Physarum Learner: A Novel Structure Learning Algorithm for Bayesian Networks inspired by *Physarum Polycephalum*

DISSERTATION ZUR ERLANGUNG DES DOKTORGRADES
DER NATURWISSENSCHAFTEN (DR. RER. NAT.)
DER FAKULTÄT FÜR BIOLOGIE UND VORKLINISCHE MEDIZIN
DER UNIVERSITÄT REGENSBURG

vorgelegt von

Torsten Schön

aus

Wassertrüdingen

im Jahr 2013

Der Promotionsgesuch wurde eingereicht am:
21.05.2013

Die Arbeit wurde angeleitet von:
Prof. Dr. Elmar W. Lang

Unterschrift:

Torsten Schön

Abstract

Two novel algorithms for learning Bayesian network structure from data based on the true slime mold *Physarum polycephalum* are introduced. The first algorithm called C-PhyL calculates pairwise correlation coefficients in the dataset. Within an initially fully connected Physarum-Maze, the length of the connections is given by the inverse correlation coefficient between the connected nodes. Then, the shortest indirect path between each two nodes is determined using the *Physarum Solver*. In each iteration, a score of the surviving edges is increased. Based on that score, the highest ranked connections are combined to form a Bayesian network. The novel C-PhyL method is evaluated with different configurations and compared to the LAGD Hill Climber, Tabu Search and Simulated Annealing on a set of artificially generated and real benchmark networks of different characteristics, showing comparable performance regarding quality of training results and increased time efficiency for large datasets.

The second novel algorithm called SO-PhyL is introduced and shown to be able to outperform common score based structure learning algorithms for some benchmark datasets. SO-PhyL first initializes a fully connected Physarum-Maze with constant length and random conductivities. In each *Physarum Solver* iteration, the source and sink nodes are changed randomly and the conductivities are updated. Connections exceeding a predefined conductivity threshold are considered as Bayesian network arcs and score of nodes included in selected connections is examined in both directions. A positive or negative feedback is given to conductivity values based on calculated scores. Due to randomness in initializing conductivities and selecting connections for evaluation, an ensemble of SO-PhyL is used to search the final best Bayesian network structure. First, a detailed analysis of the influence of configuration parameters on learning quality of SO-PhyL is presented, before the novel algorithm is compared to state of the art structure learning methods using a set of artificially generated benchmark networks. Next, seven real benchmark networks are used to further analyse the performance of SO-PhyL compared to other algorithms. It is observed that SO-PhyL is a competitive structure learning method that outperforms Simulated Annealing in most datasets, Tabu Search in some datasets and even LAGD for specific networks.

A newly generated medical dataset collecting clinical parameters of liver biopsy proven Non-alcoholic Fatty Liver Disease (NALFD) patients delivered from the Medical University of Graz is analysed using common feature selection and classification methods in order to find novel biomarker candidates for NAFLD. Magnesium is identified as promising biomarker and is forwarded to medical experts where a mouse model is used to verify the novel biomarker candidate. In addition, both Physarum based algorithms are used to learn a Bayesian network structure from the NAFLD dataset to get deeper understanding of parameter interactions.

Contents

CHAPTER 1

Introduction

Technological progress over the last decades changed the way in which knowledge is discovered and referred between and within generations by creating the ability of collecting, processing and analysing a huge amount of data automatically by computer based systems. Learning is no longer only a process performed between human beings based on their individual experiences. Automatic systems can be used to collect data for any specific task and algorithms are applied on this data in order to get insights and extract knowledge from it that can thus be understood and used by human beings or passed directly into another algorithm. Nowadays, the amount of assembled data is constantly growing and is solicited to be analysed, which is a great chance to do a next big step in technological evolution. Developing valid and useful methods to optimize knowledge discovery in any thinkable way is one of the most challenging and important tasks for our generation and will have major impact on further development of interaction between humanity and information technology. These methods and models are used for example to explain observations, understand them and, based on these understandings, predict future events.

One of these models is called Bayesian network [159] which incorporates a broad spectrum of possibilities, for example explanation, classification and prediction. Bayesian networks are probabilistic graphical models represented by a directed acyclic graph (DAG), modelling data by estimating probabilistic relations between data parameters and are expatiated in Section 2.2. Further, the illustration of this probability model as a graphical network makes it more intuitive to be used by humans than other only mathematical models. Nevertheless, learning a Bayesian networks from data representing the underlying distribution of the dataset precisely is a very hard task, where exact methods have shown to be NP-hard [53]. A set of different heuristic methods [41, 42, 1, 90, 91, 37] has been introduced that showed adequate performance for most datasets but also come with significant drawbacks. Most of these algorithms suffer from getting stuck in local performance maxima and as by definition of heuristic methods, not all possibilities are considered when searching the space of possible network structures. As this is impossible to perform in foreseeable time, a possible improvement can be given to these methods

by in either the way the search space is traversed or in the way the quality of network structures is measured.

While researching novel concepts and methods, one can often find solutions already existing in nature. No matter what outstanding technologies mankind has developed so far, compared to creatures, concepts and methods evolution showed up with, technology is still in its infancy. Fortunately, this offers the opportunity to copy solutions from nature instead of developing them from scratch. Most times, this is part of development processes in engineering. But also in computer science, there are many algorithms that originate from observations made in biological processes as for example artificial neural networks [133] that try to model brain processes by implementing interconnected neurons, also called perceptrons. Another biologically inspired method in machine learning and data mining area is the ant colony optimization algorithm [71]. The concept that ants are marking paths between their nest and located food sources by emitting pheromones along the path, which animates other ants of the same colony to follow the path, has been mathematically formulated and applied to find shortest paths through a graph or road maps.

A rather related biological mechanism has been observed by Nakagaki *et. al.* in 2000, who showed that the true slime mold *Physarum polycephalum* is able to find the shortest path through a maze [152]. In previous experiments, Nakagaki and colleagues observed that *Physarum polycephalum* is reforming its shape in response to food sources. The former sponge like body of the plasmodium reforms itself to a tubular system transporting sol once food sources are reached by spreading plasmodium. The food sources are covered by the sponge part of the slime mold and if more food sources exist, they are connected by a single tube along the shortest possible path between the food sources. Based on the primary founding, Nakagaki *et. al.* developed a maze that is fully covered by *Physarum polycephalum* and placed food sources at the entry and the exit. They discovered that after a few hours, the slime mold has disappeared in all dead end paths and only a single tube remained connecting the sponge sections at entry and exit food sources along the shortest path through the maze. The mathematician Athushi Tero and his colleagues developed in contribution with Nakagaki a mathematical model of *Physarum polycephalum*s behaviour, called *Physarum Solver*, and showed that it acts in the same manner as the real organism does [194, 195]. Since publication of this model, huge research interest has been given to applications using the *Physarum Solver* especially for shortest path finding problems (also referred to as Travelling Salesman Problem) first defined in 1930 by Karl Menger. The fact that the NP-hard [108] Travelling Salesman Problem can be solved more efficiently by using the *Physarum Solver* motivated the application of *Physarum Solver* to the also NP-hard problem of learning Bayesian network structures. While the ant colony optimization algorithms have already been successfully applied to the problem of learning Bayesian network structure from data [64], to the best of the authors knowledge, this is the first time that a concept based on *Physarum polycephalum* or a method using the *Physarum Solver* is applied to learn Bayesian network structure from data.

Overview This thesis addresses the question if the mathematical model based on the slime mold *Physarum polycephalum* can be used to learn the structure of Bayesian networks from data by introducing two novel concepts of integrating *Physarum Solver* into a Bayesian network structure learning process.

Chapter 2 provides background information about *Physarum polycephalum* and introduces basic concepts of the *Physarum Solver*. Further, an introduction is given to probabilistic graphical models, Bayesian networks and how they can be learned from data. Lastly, additional relevant methods are explained, benchmark datasets are formulated and development environment used for experiments performed in this thesis is described.

In Chapter 3, a correlation based approach called C-PhyL using the *Physarum Solver* to learn Bayesian network structure is introduced, different parameter settings are evaluated and learning performance is measured by comparing the novel algorithm to state of the art methods. A conclusion and a discussion are given at the end of this chapter.

Next, another novel algorithm called SO-PhyL is provided using a score optimization technique based on an updated version of the *Physarum Solver* in Chapter 4. First, the method is described and algorithms are presented. Next, parameters influencing SO-PhyL are studied carefully before comparing the newly developed algorithm to state of the art methods by learning structures of different benchmark datasets. Again, a brief conclusion and discussion are given at the end of this chapter.

Further, a medical dataset that has been retrieved from project partners of this thesis at the Medical University of Graz (MUG) is analysed in Chapter 5 to detect possible biomarker candidates for the Non-alcoholic Fatty Liver Disease (NAFLD). Also, the two new structure learning algorithms presented in this thesis are used to learn a Bayesian network structure to get insight in relations of biomedical parameters related to NAFLD.

Finally, Chapter 6 provides a more detailed discussion of both algorithms and possible directions for future work before Chapter 7 finally summarizes results.

CHAPTER 2

Material

2.1 Physarum Polycephalum

2.1.1 Biological background

Physarum polycephalum belongs to the superclass of *Myxomycetes*, also referred to as true slime molds. Together with the cellular slime molds they form the group of *Mycetozoa* [45]. *Physarum polycephalum* is conspicuously pigmented in yellow or orange (for examples see Figure 2.1) and does not perform photosynthesis. The vegetative nutrition consuming

Figure 2.1: Different examples of *Physarum polycephalum* in wild life. Images: Thomas H. Kent [109]

stage of the Physarum's life cycle is called plasmodium and consists of a single amoeboid like cell with multiple diploid nuclei where all nuclei divide at precisely the same time. The plasmodium can grow up to a size of several centimetres and preferably grows on wet ground, rotting leaves and logs.

Once the source of nutritions is exhausted or the Physarum's habitat desiccates, the plasmodium stops growing and differentiates to stages of its live cycle that allow reproduction. First, sporangia are built that often have a stalk. Inside the sporangia, several haploid sporocytes are built via meiosis until they are released. Once life conditions got better again, the spores germinate and develop to amoeboid or flagellated cells. Pairs of equal types of these cells combine to diploid zygotes by syngamy. The zygotes grow to a

plasmodium by repeating mitosis of their nuclei or by melting with other zygotes which closes the life cycle.

The body of the plasmodium is built of a network of tubular channels efficiently transporting nutrients and chemical signals through the organism [152, 146]. These tubes are built of actin-myosin fibers and are surrounded by a "sponge" section including distributed actin-myosin fibers where the protoplasmic sol flows in and out [194]. The sol is streamed through a complicated network of tubes within the organism. Therefore, the composition of the tube network has high influence on the transportation of information and materials. The ability of adapting this network system in response to external conditions by disassembling or reassembling tubes over time enables the organism to optimize the flow of sol in a changing environment. Further, tubes transporting a high amount of sol widen over time, whereas tubes with less flow shrink and tend to disappear [153].

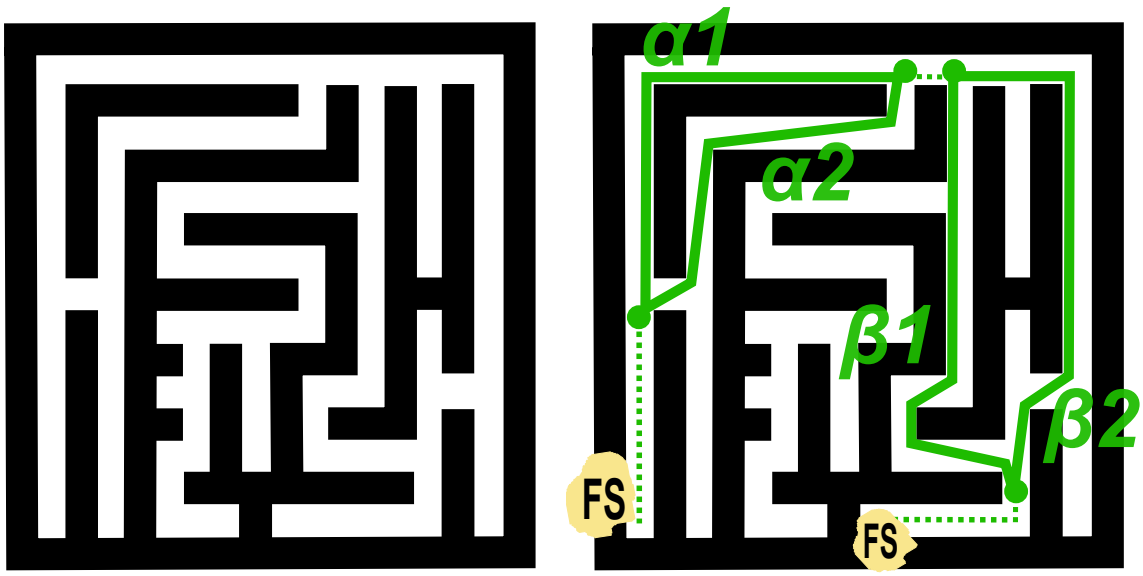
The transportation of sol is driven by variation of hydrostatic pressure along the tubes [195]. The hydrostatic pressure that streams protoplasm through the tube is caused by rhythmic contractions [154]. If sol is flowing in a given direction for a certain period, the tubular structure is formed in that direction as the actomyosin fibers that are arranged along the length of the cortex of the tube are oriented by the Stretch-Activation-Effect[1] [107, 142]. Tero *et. al.* [195] hypothesized, that the shear stress developed by fast flowing (1mm/sec) protoplasm induces a stretching effect that leads to regular orientation of the actomyosin fibers of the tubes. This ongoing stretching force widens the tube with sufficient flux. In a wider tube, resistance to the flow of sol decreases what in fact leads to an increase of flux in the tube. That means, the network has the ability to adjust its tubular system to variations of flux by giving positive feedback.

Rhythmic contractions with a period of two minutes [195] are exhibited by the actin-myosin fibers of the sponge section, exerting pressure on the protoplasmic sol and pushing it into the tube. The sol flows through the tube until it flows out into the sponge section at the other end of the tube. Please note that the direction of the flow changes periodically. Experiments by Nakagaki *et. al.* [153, 150] showed that when applying food sources (oak flakes) to the plasmodium, the slime mold grows around the food sources and disappears elsewhere. When several food sources are present, *Physarum polycephalum* keeps them connected by a single tube transporting sol between the food source areas. As the direction of the flow of sol changes periodically and sol flows between the food sources, at any time, one food source can be seen as source of the sol and the one at the end of the tube can be seen as sink of the sol [195]. This assumption is very important for building a mathematical model of the network dynamics as further described in Section 2.1.3.

2.1.2 Maze-solving by *Physarum polycephalum*

Nakagaki *et. al.* examined network dynamics of *Physarum polycephalum* at the turn of the millennium, where they showed that oscillatory reaction-diffusion type equations can be used to model cellular activities of the plasmodium [153, 139, 193, 135, 151, 210]. Former, Babloyantz and Sepulchre [178, 177] computationally showed that a simple network of non-linear oscillators is able to navigate in a complex geometrical system like a maze.

[1] Stretch-Activation-Effect: When stretching randomly oriented fibrous molecules, they tend to reorient in the direction of the stretching force.



(a) (b)

Figure 2.2: Schematic view of the maze used by experiments of Nakagaki *et. al.*. **a)** Configuration of the maze, **b)** The green lines show the possible paths through the maze connecting the two applied food sources (FS). The path forks at two points: First either way α_1 or α_2 can be chosen and second, either way β_1 or way β_2 has to be picked.

Therefore, Nakagaki supposed that the plasmodium of *Physarum polycephalum* is also able to solve a maze [152].

They cut a negative pattern of the maze shown in Figure 2.2a out of a plastic film and placed it on an agar plate as the plasmodium avoids growing on the dry surface of the plastic film. After that, they cut off several small pieces of a cultured plasmodium and distributed them equally within the maze. After a few hours, the plasmodial pieces spread and coalesced to a single organism that covered the complete maze. In the following, oat flakes where placed to serve as food sources (FS) at two specific points in the maze to simulate the start and end point, see Figure 2.2b. The green lines in Figure 2.2b illustrate the possible paths through the maze connecting the two food sources. The solution path first forks into subpath α_1 and α_2 and then bifurcates another time to subpath β_1 and β_2. Original measurements of Nakagaki *et. al.* [152] showed that subpath α_2 $(33 \pm 1mm)$ is clearly shorter than path α_1 $(41 \pm 1mm)$. In contrast, the paths β_1 $(44 \pm 1mm)$ and β_2 $(45 \pm 1mm)$ share nearly the same length. Therefore, the shortest path between the two food sources is via α_2 and either β_1 or β_2. Please note that Figure 2.2 only shows the layout of the maze and it's scale may not be in precise accordance with the original maze.

The initial experimental set up can be seen in Figure 2.3a, where the plasmodium illustrated in yellow has crowd in each path of the maze. At time $t = 0$, two food sources are placed as described above. The slime mold is initially built of sponge sections with only very short and thin tubes. Nakagaki *et. al.* call this composition of the slime mold "sheet-like plasmodium". Only four hours after applying the food sources, the plasmodium retired from dead ends, see Figure 2.3b. In the food source areas, the

(a) Initial (b) After 4 hours (c) After 8 hours

Figure 2.3: Maze-solving by *Physarum polycephalum* on basis of the figures presented by Nakagaki *et. al.* [152] . The plasmodium is illustrated in yellow. **a)** Initially, the slime mold covers the complete maze. **b)** Four hours after placing the food sources, the plasmodium disappeared in the dead ends of the maze. **c)** Another 4 hours later, the slime mold remained only at the shortest path by way of α_2 and β_1.

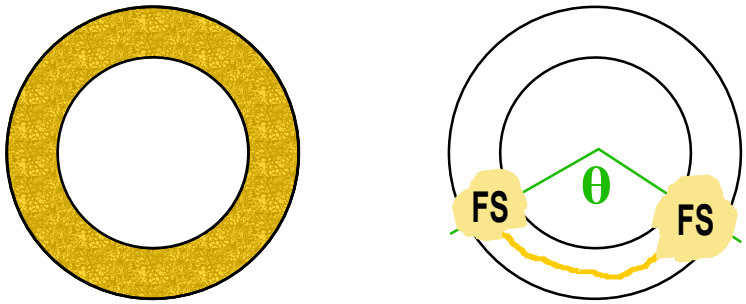
plasmodium grew and covered the food sources whereas the structure of the plasmodium that connected the food sources changed to build single thick tubes at the four possible connections α_1, α_2, β_1 and β_2. Another four hours later, only the tube at the shortest path by way of α_2 and β_1 survived (Figure 2.3c). Nakagaki *et. al.* repeated this experiment several times [152], where each time the shorter path α_2 survived and α_1 disappeared. The paths β_1 and β_2 have been selected equally often where in some runs, both connections where still present after eight hours. Nakagaki *et. al.* assumed that the difference in length between β_1 and β_2 is lost in the windings of the tubes. Nakagaki explains the reason for shrinking dead ends and developing a single thick tube connecting the food sources as follows:

> *"The addition of food leads to a local increase in the plasmodium's contraction frequency, initiating waves propagating towards regions of lower frequency [167, 128, 140, 151, 210], in accordance with the theory of phase dynamics [120]. The plasmodial tube is reinforced or decays when it lies parallel or perpendicular, respectively, to the direction of local periodic contraction [154]; the final tube, following the wave propagation, will therefore link food sites by the shortest path."* (Toshiyuki Nakagaki in [152])

It remains to be noted that in some rare cases no path survived at all, meaning that the organism has split into single plasmodia at each food source.

2.1.2.1 Further investigations

In the following years, Nakagaki *et. al.* further investigated the behaviour of path finding by *Physarum polycephalum* by using different shapes [153, 146, 150, 149]. The procedure of placing several pieces (cut from a plasmodium) into the shapes, allowing the plasmodiums to grow and coalesce, applying food sources and investigating the changes of the plasmodium, remained constant. However, they showed that the plasmodium only finds



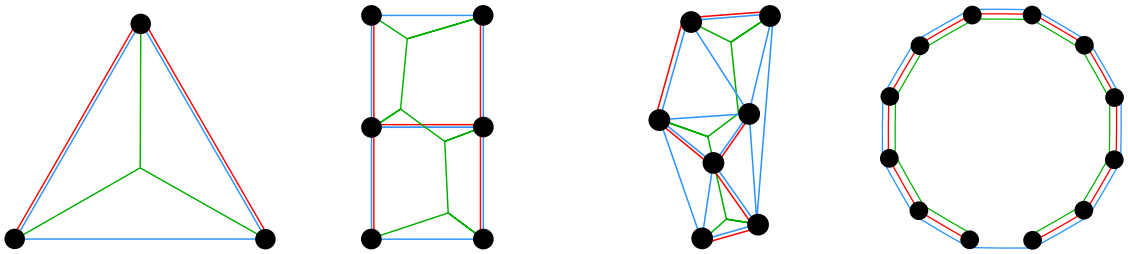
(a) Covered ring shape (b) Shortest path

Figure 2.4: Schematic view of the ring shape used by Nakagaki *et. al.* [153]. **a)** The ring shape is initially covered by the plasmodium illustrated in yellow. **b)** Four hours after the nutrients have been applied, the food sources are connected by one thick tube at the shortest path.

the shortest path if the initial composition of the plasmodium is of sheet-like structure and no major tubes have been developed so far [153]. To investigate the tube selection between two food sources, a simple ring shape has been used, see Figure 2.4a. The distance was systematically varied by changing the angle θ by which the food sources are placed in the ring with respect to the center of the circle as shown in Figure 2.4b. For adjustments with $\theta = 90°$ and $\theta = 135°$, the shorter path is always selected [153] (for experiments where exactly one path survived). For an angle of $\theta = 160°$, the distance between the two pathways gets very small and the probability of survival of both paths increases. Therefore, Nakagaki *et. al.* concluded that the plasmodium can find the shortest path only within a certain precision.

2.1.2.2 Relation between tube formation and amount of food

Another interesting discovery is that the vein selection is affected by the amount of applied food [153, 145]. When increasing the amount of food for arrangements with $\theta = 90°$, the number of tubes connecting the two points decreases. When the food is limited, two veins exist more often but when excessive food is available, there are often no tubes meaning the organism has divided into two individuals. From an evolutionary point of view, this behaviour totally makes sense. When food is available only rarely, the organism keeps connected to both food sources. In a case where one food source is exhausted, the organism is still connected to another one and continues to live. On the other hand, when enough food is provided, the organism splits into individuals to avoid wasting energy by maintaining a connecting tube and transporting sol from one end to the other. Further, Nakagaki *et. al.* reported that the amount of food also affects the time scale of vein rearrangement as the plasmodium first covers the food source completely before extending from the food sites. Therefore, also dispersion of the food sources is a determining factor of how the tubes are developed when keeping the concentration of food constant.



(a) 3 food sources (b) 6 food sources (c) 7 food sources (d) 12 food sources

Figure 2.5: Arrangement of different numbers of food sources based on the illustration presented by Nakagaki *et. al.* [150], where black dots indicate the food source positions. The red line indicates the networks of the *Minimum Spanning Tree* (MSP), the green lines show *Steiner's Minimal Tree* (SMT) and the blue networks show the *Delaunay Triangular Network* (DTN) respectively. **a)** Equilateral triangle with 3 food sources. **b)** Two adjoining squares containing 6 food sources. **c)** Irregular arrangement of 7 food sources. **d)** Regular duodecagon with a food source at each bend.

2.1.2.3 Applying multiple food sources

In 2004, Nakagaki *et. al.* extended the experimental set-up by applying more than two food sources to the plasmodium [150, 144] and showed that the network geometry meets all the requirements of a smart network: close connections between all branches; a short total length of the tubular system and tolerance to accidental uncoupling of tubes [150]. In order to examine the rather smart strategy of the plasmodium, they studied the development and formation of the tubular system within different arrangements of multiple food sources as shown in Figure 2.5 and compared them to three mathematically well defined methods: *Minimum Spanning Tree* (MST) [119, 163], *Steiner's Minimal Tree* (SMT) [54] and *Delaunay triangulation network* (DTN) [68]. The network of MSP is drawn in red, SMT in green and DTN in blue for each of the settings shown in Figure 2.5.

To evaluate the shapes of the tube networks, the average degree of separation (AS), fault tolerance (FT) and the total length of the tubular system have been measured and compared. An introduction and theoretical background to these methods can be found in literature of adaptive self-organizing networks [190, 29, 207, 22]. When applying these measures to plasmodium networks, the degree of separation is defined as the number of food sources that are passed by the shortest path between two given food sources. The degree of separation is set to zero if the two food sources are directly connected to each other. The fault tolerance (FT) is the probability that the organism is not fragmented if a tube is accidentally disconnected at a random point of the tube. As the probability of accidentally disconnecting a tube grows with the length of the tube, a combined index FT/TL has been used to measure the ratio of benefit to cost [150, 144]. It is known, that MST and SMT have short TL and low FT while DTN has higher FT but also increased TL (blue networks in Figure 2.5).

For the triangular configuration with three food sources as shown in Figure 2.5a, the shape of the surviving tubular network varied considerably on the set of several individually tested agar plates. But in all cases, the three food sources remained connected by

only a few thick tubes. Comparing the AS and FT/TL values, it can be seen that the physarum solutions have constantly high FT/TL comparable to DTN and also lowest AS values, indicating a very effective network. In most cases, the resulting network was a kind of mixture between SMT and DTM [144]. As for almost all experiments with the triangle arrangement, the junction of the connecting tube lay within a 5% boundary around the Steiner point. Further tests have been performed but showed that the plasmodium does not seek the exact Steiner point. The fact that TL was kept short leads to the assumption that the organism makes a good approximation of SMT by searching for the shortest connecting paths.

For biologically more complex arrangements with six, seven and twelve food sources as shown in Figure 2.5b-2.5d, again only a few thick tubes remained that connected all food sources. The mentioned beneficial properties of a network with low AS and high FT/TL were especially clear for those arrangements with a higher number of food sources. Thus, Nakagaki *et. al.* [150] concluded that the organism forms a sophisticated transportation network that has a better configuration than the network built by Steiner's minimum tree. To summarize, they established two empirical rules describing the formation of the plasmodium when food sources are applied [150, 144]:

1. *Open ended tubes are likely to disappear.*

2. *When two or more tubes connect the same two food sources, the longer tubes tend to disappear.*

Please note, that the second rule is only applicable assuming that enough food is provided. As already mentioned, the number of surviving tubes is decreasing with increasing amount of food.

2.1.3 *Physarum Solver*: A mathematical model of maze-solving by *Physarum polycephalum*

In 2005, Nakagaki collaborated with Atsushi Tero and Ryo Kobayashi from the Department of Mathematics and Life Science at Hiroshima University to develop a mathematical model of the maze-solving mechanism of *Physarum polycephalum* [194, 195]. Tero *et. al.* described their model by using the maze introduced by Nakagaki *et. al.* (Figure 2.2a), where each intersection and each end in the maze is represented as node and denoted by N_i. The two special path ends where the food sources have been placed in the original experiments are called *Source Node* N_1 and *Sink Node* N_2. As already mentioned in the previous section, the direction of sol flowing between two food sources reverses periodically. Therefore, at each time, one food source can be seen as a source of sol and the other one as a sink of sol. As the direction of sol flow is not crucial to the dynamics of the mathematical model, Tero *et. al.* defined one food source as source and the other one as sink. Figure 2.6 shows the graph of the maze where nodes are indicated by squares and the two special food source nodes are indicated by stars. A tube connecting two nodes N_i and N_j is referred to as section M_{ij}. Please note that the equations and assumptions made in the remainder of this section have all been introduced by Tero *et. al.* [194, 195, 198, 149].

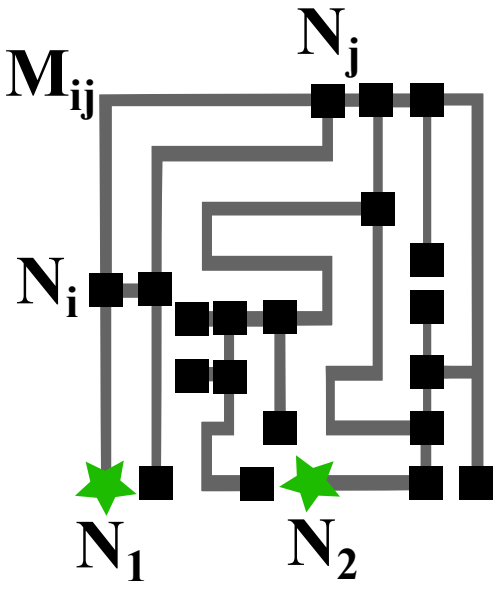


Figure 2.6: Nakagaki's maze illustrated as graph like introduced by Tero *et. al.* [194, 195]. The source node N_1 and sink node N_2 are indicated by stars. Ordinary nodes N_i are indicated by squares. The path connecting two nodes N_i and N_j is referred to as M_{ij}

The flux from N_i to N_j through M_{ij} is expressed by variable Q_{ij}. Tero *et. al.* assumed that the flow along the tube is approximately a Poiseuille flow [191] and can therefore be expressed as

$$Q_{ij} = \frac{\pi \alpha_{ij}^4}{8\kappa} \frac{p_i - p_j}{L_{ij}} \tag{2.1}$$

where L_{ij} is the length between node N_i and node N_j. Variable α_{ij} denotes the radius of the tube corresponding to the edge M_{ij} and κ is the viscosity coefficient of the sol. The pressure at node N_i is given by p_i and the pressure at node N_j by p_j, respectively. To integrate the constants into a single variable, the conductivity variable D_{ij} is defined as

$$D_{ij} = \frac{\pi \alpha_{ij}^4}{8\kappa}. \tag{2.2}$$

and is the inverse of the resistance of the tube per unit length. Therefore, Equation 2.1 can be rewritten as

$$Q_{ij} = \frac{D_{ij}}{L_{ij}}(p_i - p_j). \tag{2.3}$$

As the nodes are only abstraction units used to formulate a mathematical model, it can be assumed that the sol capacity of a node is zero. Hence, these nodes cannot absorb sol (apart form the source and sink node). Additionally assuming that the tubes itself do not push sol into surrounding sponge sections, but transport the complete sol from one node to the other, Kirchhoff's conservation law of sol can be considered. Therefore, the total flux at an ordinary node N_i can be written as

$$\sum_j Q_{ij} = 0 \quad (j \neq 1, 2) \tag{2.4}$$

and the total flux for source node N_1 and sink node N_2 is given by

$$\sum_i Q_{i1} + I_0 = 0 \tag{2.5}$$

and

$$\sum_i Q_{i2} - I_0 = 0. \tag{2.6}$$

where I_0 is the flux that flows from the source node to the sink node. It is kept constant throughout the process. In other words, I_0 is the amount of food that is absorbed by the organism from the food source.

In Section 2.1.2, the adaptive network dynamics of widening tubes with a high flux and shrinking tubes with low flow of sol have been introduced. From a more abstract point of view, one can say that these modifications do change the conductivity of the tubes over time. Hence, Tero *et. al.* proposed the following adaptation equation for the evolution of $D_{ij}(t)$:

$$\frac{d}{dt}D_{ij} = f(|Q_{ij}|) - rD_{ij} \tag{2.7}$$

where r is a decreasing rate constant of the tube. The length of the tubes has been kept constant. Therefore, all network dynamics over time are modelled by Equation 2.7. As can be seen easily, the conductivity of the tube disappears over time if there is no flux in the tube. Two types if monotonically increasing continuous function $f(Q)$ satisfying $f(0) = 0$ have been supposed:

$$f(Q) = mQ^{\mu} \tag{2.8}$$

$$f(Q) = \delta\frac{(\frac{Q}{Q_h})^{\mu}}{1 + (\frac{Q}{Q_h})^{\mu}} \tag{2.9}$$

where the exponent μ has to be positive. The function of Equation 2.9 is motivated by the previously noted observations that the formation of tubes is dependent on the amount of provided food, represented by variable I_0, and also by the assumption that there is a maximal value for the tube diameter that constrains the conductivity.

The constants m, δ and r (Equation 2.7) can be resolved by taking a characteristic magnitude of I_0 and taking a characteristic conductivity $\bar{D}$ so so that the relation $f(I_0) - r\bar{D} = 0$ holds [195]. Considering the dimensionless variables and functions, the model equation becomes

$$\frac{d}{dt}D_{ij} = f(|Q_{ij}|) - D_{ij} \tag{2.10}$$

and the two types of $f(Q)$ derive as

$$f(Q) = Q^{\mu} \tag{2.11}$$

$$f(Q) = \frac{(1+\alpha)Q^{\mu}}{1 + \alpha Q^{\mu}} \tag{2.12}$$

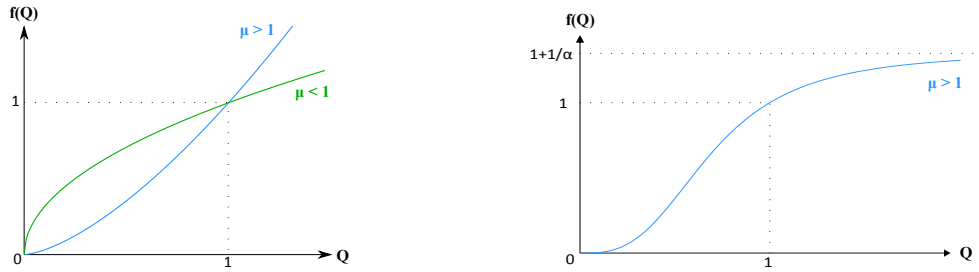
where $\alpha = (\frac{I_0}{Q_h})^{\mu}$. Detailed calculations of removing dimensions from the model equation are given by Tero *et. al.* [195]. The network partial differential equation, where the left hand side is a non-uniform discrete Laplacian of the pressure p can be derived from

Equations 2.3, 2.4, 2.5 and 2.6 as follows:

$$\sum_i \frac{D_{ij}}{L_{ij}}(p_i - p_j) = \begin{cases} -I_0 & \text{for } j = 1 \\ +I_0 & \text{for } j = 2 \\ 0 & \text{otherwise} \end{cases} \tag{2.13}$$

The pressure p_2 of sink node N_2 is set to zero as basic pressure level. Therefore, all pressures can be calculated with Equation 2.13 and each flux can be determined by solving Equation 2.3. Please note that the disappearance of a tube is expressed by a conductivity value of zero.

Next, Tero *et. al.* investigated the asymptotic behaviour of their system for a given function $f(Q)$ with different parameters and tested if their model would solve the maze of Nakagaki in the same manner as the plasmodium of the real slime mold does. The graph of Equation 2.11 and 2.12 is printed in Figure 2.7 to illustrate different behaviour for different values of μ.For $f(Q) = Q^\mu$, the value of μ can be any positive number where the function becomes linear for $\mu = 1$. For $f(Q) = \frac{(1+\alpha)Q^\mu}{1+\alpha Q^\mu}$, the parameter μ has to be in the range of $\mu > 1$ to get a sigmoid profile as shown in Figure 2.7b and parameter α needs to be positive.



(a) Graph of Equation 2.11 (b) Graph of Equation 2.12

Figure 2.7: Plot of functions for $f(Q)$. Note that as by definition of dimensionless functions, $f(1) = 1$. **a)** Function of Equation 2.11 with $\mu > 0$. **b)** Sigmoid curve of Equation 2.12 with $\mu > 1$ and $\alpha > 0$.

Each length L_{ij} of edges M_{ij} has been set with respect to measurement of the original maze shown in Figure 2.2a. The initial conductivities D_{ij} have been randomly set in a range of $D_{ij} = [0.5, 1.0]$ and are equally distributed. In all simulations done by Tero *et. al.*, paths with a dead end vanished first, further referred to as *dead end cutting*. Meaning that the immediate state, where only the four paths $\alpha_1, \alpha_2, \beta_1$ and β_2 survive (Figure 2.3b), is always observed. Tero *et. al.* further reported that dead end cutting is completed already at time step $t = 5$ [195]. The next paragraphs provide a short overview of the results obtained with different parameter configuration. Detailed simulation results are given in *"A mathematical model for adaptive transport network in path finding by true slime mold"* [195].

Simulations with $f(Q) = Q^\mu$:

Case $\mu > 1$: Rapid dead end cutting has been observed while the three short solution paths connecting food sources with the α and β paths always survived and their

conductivity quickly increased. Path α_1 always survived when the initial values of D_{ij} are initialized almost uniform. Either path β_1 or β_2 survived dependent on which of their initial conductivity was bigger. In some simulations where the initial conductivity of α_2 was sufficiently larger than the one of path α_1, α_2 could survive instead of α_1. Tero *et. al.* also reported, that the choice between competing paths is made faster if the value of μ increases. To summarize, in the final state, only one paths survives and the choice of this path is dependent on the initial values of the randomly set conductivities.

Case $0 < \mu < 1$: When setting μ in the range of $]0, 1[$, the final state of the maze is totally different: all of the four paths α_1, α_2, β_1 and β_2 survive! Therefore, the final state is the same no matter what value of μ is chosen and how the initial conductivities are set. But the final conductivities of the paths are different, where the shorter path of α_2 achieves higher conductivity than α_1. A bunch of simulations showed that the convergence to the finals state is faster if the parameter μ is taken smaller.

Case $\mu = 1$: As already mentioned, in the special case $\mu = 1$, function $f(Q) = Q^\mu$ becomes linear. While dead end cutting is performed similar to the case where $\mu > 1$, the shortest path is always selected in the final state. Different to $\mu > 1$, **the final state is always the shortest path, independent from the initial values of the conductivities!** But the convergence to the final step takes more time steps than needed in previous cases.

Summery: Tero *et. al.* suggested to use μ as follows:

1. If one wishes to reach a quick acceptable solution, use $\mu > 1$
2. If the aim is to get all solution paths, use $0 < \mu < 1$
3. If the shortest path is searched, use $\mu = 1$

Simulations with $f(Q) = \frac{(1+\alpha)Q^\mu}{1+\alpha Q^\mu}$: As $f(Q)$ always requires a value of $\mu > 1$ to build a sigmoid curve, the value of α has been examined in these simulations. Keeping the value of α small, leads to a behaviour similar to the previously presented case of $\mu > 1$ with $f(Q) = Q^\mu$ where only one of the possible solution paths survived. Setting high values of α instead, all competing paths can survive until the final state. Please note that definition of "high" and "small" α values depends on the maze under study. If using intermediate values for α, not only the smallest path survives, but also not all paths remain in the final state.

To perform detailed mathematical convergence analysis, Tero *et. al.* used the more simpler ring shape introduced in Section 2.1.2.1 and a T-shaped graph [195] not presented in this thesis. In all experiments, the behaviour of the real plasmodium could be correctly simulated. It could therefore be assumed, that Tero's model is a good approximation of the behaviour of the real slime mold. In the following years, it was mathematically proven that the Phaysarum Solver can find the shortest path in a general planar graph [136, 137, 36].

2.1.3.1 Multiple food source model

The *Physarum Solver* as introduced in the previous section has been shown to perform in the same manner as the true slime mold does when applied to any kind of maze with two food sources [194, 195]. In 2008, Tero *et. al.* introduced a slightly updated variation of the *Physarum Solver* that is able to deal with more than two food sources [198]. First, they introduced a new growth function $f(Q)$ reflecting the saturation of the tube diameter but is simpler than Equation 2.12:

$$f(Q) = \frac{Q^{\mu}}{1 + Q^{\mu}} \tag{2.14}$$

It has been observed that the tube growth is less sensitive to flow changes when the difference in pressure is small [198]. To consider these observations, $\mu > 1$ is used in Equation 2.14 to get a sigmoid curve. In experiments with two food source, it was assumed that the sol flows from one food source to the other as the plasmodium changes the direction of the flow periodically and the mathematical model equations hold in either direction. But calculating a model with more than two food sources requires to know which are sources and which are sinks in order to form the model equations of flux, pressure and conductivities. To overcome this rather complicated calculations, Tero *et. al.* supposed that only one source and one sink node exists at each time step t. Hence, the model defines a food source N_i as sink randomly at each time step. A corresponding food source N_j is then randomly defined as source with probability

$$P(N_j = \text{Source}) = \frac{d_{ij}^{\beta}}{\sum_{k \neq i} d_{ik}^{\beta}} \tag{2.15}$$

where β is a positive constant, k indexes the set of food source nodes and the distance between node N_i and N_k is given by d_{ik}. This means that the food source node with the highest distance to the previously randomly selected sink node is most likely to be selected as source node.

Experiments with three food sources reproduced the same variety of network shapes as the real plasmodium, reported in Section 2.1.2.3. The different shapes could be reproduced by varying parameters I_0, μ and β. The impact of I_0 partly agrees with the observation of the real organism by increasing the number of edges with increasing I_0. This is also observed for the real organism, but with an increasing amount of food, the slime mold starts dividing itself into smaller parts as observed by Nakagaki *et. al.*, which is not modelled in the *Physarum Solver*. Also, a network shape similar to the Steiner minimum tree [54] could be observed. The value of parameter μ was chosen such that the model fits the appropriate experimental results. Hence, there is no exact parameter configuration that can be applied to any network. This is not surprising, as the solutions of the real organism also varies. Setting the parameters strongly depends on what the model is supposed to produce. But even then, the parameter configuration for different networks and number of food sources are different [198]. Next, the network indicators TL and FT have been compared to the networks created by the real organism and demonstrated that there is good agreement. Hence, the model successfully reproduces network arrangements of the real organism.

2.1.4 *Physarum polycephalum*: Related Work

In one of their key papers, Tero *et. al.* applied the *Physarum Solver* to navigation of road map of the US interstate highways [194]. They planned a car trip form Seattle to Houston and used the *Physarum Solver* to find the shortest route. Further, they simulated a road barrier between Oklahoma City and Dallas and reran the *Physarum Solver* to find the shortest path from their actual position at Salt Lake City. This work clearly demonstrated that the *Physarum Solver* can be used for navigation systems. In 2010, Tero *et. al.* also applied the *Physarum Solver* to the Tokyo rail network [197]. A lot of work has been published in the last few years applying the strategy of *Physarum polycephalum* to real world transport networks [206, 189], especially by Adamatzky *et. al.*, who simulated motorways of several countries [14, 10, 16, 17, 8, 11, 18, 6, 19, 9, 7, 15].

Further research interest concentrated on examining how *Physarum polycephalum* solves the Steiner Problem [185, 196, 149, 103]. Miyaji *et. al.* started to investigate the theoretical and mathematical background in 2007 [136]. Thenceforth, detailed convergence analysis and mathematical examinations have been done to prove that *Physarum polycephalum* can find shortest paths and how this task is solved [137, 138, 40, 103, 36]. Johannson *et. al.* introduced a solver of linear programming problems based on these mathematical analysis in 2012 [106].

Another interesting research field used *Physarum polycephalum* as a biological computing unit, called *Physarum Machines* and showed that these machines have potential for designing novel computing devices [201, 5, 3, 4, 12, 13]. In addition, physarum strategy has been applied to a lot of different application fields. Song *et. al.* for example, successfully applied *Physarum polycephalum* to the minimal exposure problem in wireless sensor networks [184]. Nakagaki *et. al.* showed in 2007 that path finding of *Physarum polycephalum* is sensible to light [148], Masi *et. al.* used the strategy for decision making [127] and Reid *et. al.* recently succeeded in solving the Towers of Hanoi by using the *Physarum Solver* [166]. Finally, in 2012, Siriwardana *et. al.* further improved the *Physarum Solver* by integrating shuttle streaming and showed that their method is 40-11650 faster compared to Tero's *Physarum Solver* [182].

A detailed study of biological background is given by Goodman [93] where information regarding maze-solving and *Physarum Solver* can be found in review articles by Nakagaki *et. al.* [149, 147].

2.2 Bayesian networks

2.2.1 Fundamentals: Probabilities and Bayes' Rule

The term "Probability" is often used in everyday life. In weather forecasts for example, we are given a value for the probability that it will rain tomorrow. Or we say that the probability to get that awesome job is very low. Usually, when people talk about probability, their intention is to communicate a feeling on how probable it is that something will happen. This enables the listeners to estimate the certainty of the event and plan further steps. For example, if I organize a dinner with my friends and I ask them to come, a friend may answer that he can probably take part. At this point, he is not certain about coming to the dinner as he might has to check the date in his organizer or he may ask his wife first. Therefore, he doesn't appeal to my invitation but gives me an evidence that it is more probable that he will come than not. In social live, these uncertain statements are very important for interaction and communication. If my friend first accepts my invitation and cancels it later, I might be disappointed. But with saying that he will probably come, the option that he can't come is still valid and I can adapt myself to that case. It's the same with weather forecasts, when given a forecast that it will not rain tomorrow, I believe that it will be dry and I won't take an umbrella to work. I might be really upset if it rains anyway and I get wet on my way home. By giving me probability of 20% that it will rain, I know that in two out of ten times, it rains and I take my umbrella with me precautionary.

When talking about probability in a mathematical or statistical manner, the term has to be defined more precisely. Please note that notations and definitions of the following sections are based on Nir Friedman and Daphne Koller's excellent text book *Probabilistic Graphical Models* [114]. First, we have to formally define the events we want to assign a probability to. We denote the *space* of possible events by Ω. In a case where we are examining the outcome of rolling a die, the possible events might be $\Omega = \{1, 2, 3, 4, 5, 6\}$. Further, we define the set of *measurable events* S, where each event $\alpha \in S$ is a subset of Ω. In our die example, a possible set S can be the subset of odd numbers $S = \{1, 3, 5\}$. Koller and Friedman [114] defined the probability distribution as follows:

Definition 1 *A probability distribution P over (Ω, S) is a mapping from events in S to real values that satisfies the following conditions:*

- $P(\alpha) \geq 0$ *for all* $\alpha \in S$
- $P(\Omega) = 1$
- *If* $\alpha, \beta \in S$ *and* $\alpha \cap \beta = \emptyset$ *then* $P(\alpha \cup \beta) = P(\alpha) + P(\beta)$ *where* $\emptyset$ *is the empty event*

The first two conditions state that the probability value is in the range of zero and one, where $P(\alpha) = 1$ means that it is guaranteed that event α occurs and if $P(\alpha) = 0$, event α is impossible. Condition three says that the probability that either event α or event β occurs, is the sum of the probabilities $P(\alpha) + P(\beta)$ (if α and β are mutually disjoint).

When investigating real world distributions, events are often not independent. In these cases, the knowledge of the probability of event α changes our belief of event β. Consider a distribution over a population of students. Let event α denote students with

grade A, event β denote students with high intelligence and let the set of intelligent student who got grade A be $\alpha \cap \beta$. Once, evidence of event α changes, meaning that the state of α is known, we can update our belief of β by calculating the *conditional probability* which is defined as

$$P(\beta|\alpha) = \frac{P(\alpha \cap \beta)}{P(\alpha)}. \tag{2.16}$$

Equation 2.16 states that the probability of β when α is known, can be calculated by dividing the probability that α and β is true by the probability of α. Rewriting the definition of the conditional probability directly leads to *chain rule* defined as

$$P(\alpha \cap \beta) = P(\alpha)P(\beta|\alpha) \tag{2.17}$$

and in the more general form for events $\alpha_1 \dots \alpha_k$:

$$P(\alpha_1 \cap \dots \cap \alpha_k) = P(\alpha_1)P(\alpha_2|\alpha_1) \dots P(\alpha_k|\alpha_1 \cap \dots \cap \alpha_{k-1}). \tag{2.18}$$

As by definition $\alpha \cap \beta$ is equal to $\beta \cap \alpha$, it follows from Equation 2.17 that

$$P(\alpha)P(\beta|\alpha) = P(\beta)P(\alpha|\beta) \tag{2.19}$$

and therefore the definition of *Bayes' rule* [32]

$$P(\alpha|\beta) = \frac{P(\alpha)P(\beta|\alpha)}{P(\beta)} \tag{2.20}$$

which allows to calculate the conditional probability from the known reverse conditional probability.

Until know, only probabilities of events have been considered. But to handle problems in a more mathematically manner, attributes that can have different values are more adequate. With respect to the previously introduced student example, the event that a student got grade A can also be modelled by an attribute *Grade* that has value *A*. Clearly, using an attribute is more flexible as other nodes could also be applied to *Grade*. Therefore, the probability $P(GradeA)$ can be rewritten as $P(Grade = A)$. Generally, attributes and their outcomes are referred to as *random variables*. Formally, a *random variable* is defined by a function that associates each outcome of Ω with a value. Further, random variables will be denoted by upper-case letters and their values are referred to by lower-case letters. Sets of random variables are denoted in boldface. Let X be a random variable, then the values of X are given by $x^1 \dots x^k$ where $k \in |Val(X)|$. The distribution over a random variable (also called multinomial) is defined by

$$\sum_{i=1}^{k} P(X = x^i) = 1. \tag{2.21}$$

Clearly, as variable X is defined by values x^i, the sum of the probabilities of x^i has to be one. The distribution over events described by X is called *marginal distribution* and denoted as $P(X)$. As the marginal distribution indicates the knowledge of a ran-

dom variables before observing any other random variables or events, it is also called *prior* knowledge. The already introduced conditional probability is also valid for random variables and Baye's rule is given by

$$P(X|Y) = \frac{P(X)P(Y|X)}{P(Y)} \tag{2.22}$$

respectively.

In some cases, the probability of events from more than one random variable is investigated. The *joint distribution* over a set of random variables $\mathcal{X} = \{X_1 \dots X_n\}$ is expressed by $P(X_1, \dots, X_n)$.

Previously, we considered that observing an event β can update our belief of event α if the events are somehow influencing each other. We then expect $P(\alpha|\beta) \neq P(\alpha)$. In other situations, where learning about β does not change our belief in α, meaning that $P(\alpha|\beta) = P(\alpha)$, the events are called *independent*. Consider for example the two independent events "color of my car" and "get injured while playing soccer". When observing that my car is blue, no further knowledge of probability to get injured while playing soccer can be retrieved and these two events are obviously independent.

Definition 2 *Independence of event α and β in a distribution P is denoted by $P \models (\alpha \perp \beta)$ and holds if and only if either $P(\alpha|\beta) = P(\alpha)$ or $P(\beta) = 0$.*

If two events are by itself not independent when observed in isolation, but become independent when observing an additional third event, it is called *conditional independence* [61, 62]. For example, consider a graduate who is actually applying for a job and suppose that the events of interest are getting a job at Google and getting a job a Apple. These two events are independent in most reasonable distributions. But learning that the graduate student got a job at Google updates our belief of how good her programming skills are as we assume that Google only hires excellent students. Indeed, we assume that Apple also hires great programmers, only. Hence, we can increase the probability of getting a job at Apple. Let's further suppose that both Apple and Google base their decisions only on the final grade of the student. Once the grade is known to be A, the fact that the student got a job at Google does not change our belief of getting a job at Apple any more. In this case, Apple is conditionally independent of Google given grade A. The formal definition is given by Koller and Friedman [114].

Definition 3 *An event α is conditionally independent of an event β given γ in P, denoted $P \models (\alpha \perp \beta|\gamma)$, if $P(\alpha|\beta \cap \gamma) = P(\alpha|\gamma)$ or if $P(\beta \cap \gamma) = 0$.*

Applying the concept of conditional independence to random variables leads to the following definition also supplied in the textbook of Koller and Friedman:

Definition 4 *The distribution P satisfies $(\boldsymbol{X} \perp \boldsymbol{Y} \mid \boldsymbol{Z})$ if and only if $P(\boldsymbol{X}, \boldsymbol{Y} \mid \boldsymbol{Z}) = P(\boldsymbol{X} \mid \boldsymbol{Z})P(\boldsymbol{Y} \mid \boldsymbol{Z})$.*

This section did by far not introduce all fundamentals in probability theory but concentrates on the basic concepts that are needed to be able to define Bayesian networks in the following sections. For example, dealing with continuous random variables where the

values of the random variable are not categorical but numerical requires to define a probability density function (PDF) that integrates to one. But as the scope of this thesis is on dealing with Bayesian networks using only categorical values, density functions are not introduced any further. An interested reader may find a more detailed introduction to principles of probability theory in various excellent text books [114, 112, 59, 65, 169, 76, 159].

2.2.2 Probabilistic graphical networks

2.2.2.1 Graph theory

Section 2.2.1 already introduced the basic concepts of probability theory. But before being able to describe graphical networks, some basic principles of graph theory have to be defined. A *graph* is an abstract structure $\mathcal{K}$ that is built of a set of *edges* and a set of *nodes*, where the set of nodes is $\mathcal{X} = \{X_1 \dots X_n\}$ in most cases throughout this thesis. The set of edges $\mathcal{E}$ consists of connections between two nodes X_i and X_j that can either be directed $X_i \to X_j$, $X_j \to X_i$ or undirected $X_i - X_j$ (also indicated by $X_i \leftrightarrow X_j$) for $X_i, X_j \in \mathcal{X}$ and $i \neq j$. A *directed graph* $\mathcal{G}$ is a graph $\mathcal{K}$ where all edges $\mathcal{E}$ are directed. In contrast, a graph $\mathcal{H}$ that contains only undirected edges is called *undirected graph*.

When considering an directed edge $X_i \to X_j \in \mathcal{E}$, X_j is called the *child* of X_i and X_i is denoted as *parent* of X_j. Statement $Pa(X)$ is used to denote the parents of a node X, while the children of X are given by $Ch(X)$. A node X where $Pa(X) = \emptyset$ is called *orphan*. When considering a undirected edge $X_i - X_j$ instead, X_j is called the *neighbour* of X_i and the other way round. The set of neighbours of a node X is given $Nb(X)$. An example of a graph can be seen in Figure 2.8.

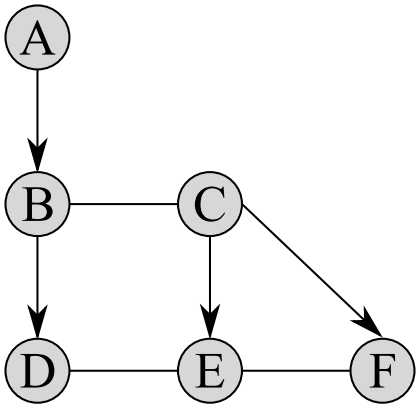


Figure 2.8: An example of a graph containing directed and undirected edges, also called partially directed graph.

In this example, graph $\mathcal{K} = (\mathcal{X}, \mathcal{E})$ consists of nodes $\mathcal{X} = \{A, B, C, D, E, F\}$ and edges $\mathcal{E} = \{A \to B, B - C, B \to D, C \to E, C \to F, D - E, E - F\}$. Clearly, node B for example has one parent $Pa(B) = \{A\}$, one child $Ch(B) = \{D\}$ and one neighbour $Nb(B) = \{C\}$.

A connection in a graph $\mathcal{K} = (\mathcal{X}, \mathcal{E})$ over nodes $X_i \dots X_k$ is called a *path* if for every $i = 1, \dots, k-1$ either $X_i \to X_{i+1}$ or $X_i - X_{i+1}$. A path is called directed if at least one edge of the path is directed. Further, a directed path $X_i \dots X_k$ where $X_i = X_k$ is called

a *cycle*. A graph containing no cycles is called *acyclic graph*. Hence, the example graph shown in Figure 2.8 is acyclic.

2.2.2.2 Introduction to Bayesian networks

A Bayesian network $\mathcal{B}$ [159, 205, 85] is a probabilistic graphical model represented by a directed acyclic graph (DAG) $\mathcal{G}$ whose nodes represent the random variables of the domain. Further, for categorical data, a Bayesian network also holds a conditional probability table (CPT) for each node. The *conditional probability distribution (CPD)* is defined by the chain rule 2.18 which factorizes the conditional probabilities. Let there be two random variables X and Y, then the joint distribution $P(X, Y)$ is factorized as $P(X, Y) = P(X)P(Y|X)$ with respect to the chain rule. Instead of specifying the joint entries $P(X, Y)$, only the prior $P(X)$ and the conditional probability distribution $P(Y|X)$ of Y given X has to be defined. The representation by conditional probability distributions of a node X has two important adventages: first, it is much more compact than the raw joint distribution if the number of nodes grows and second, it is modular. If for example a new node Z would be added, only the CPD of Z and the CPDs of nodes $Ch(Z)$ have to be updated where otherwise all entries in the joint distribution would have to be redefined. Factorizing the joint distribution into conditional probabilities of nodes given their parents and into prior distributions for orphan nodes, is a key concept of Bayesian networks.

Further, a Bayesian network can also be seen as a representation of a set of conditional independence assumptions about a distribution [87, 114]. Consider the Bayesian network $\mathcal{B}^{example}$ represented by a DAG $\mathcal{G}$ with nodes $\mathcal{X} = \{A, B, C, D, E\}$ illustrated in Figure 2.9. As can be seen, nodes A, C, D and E have binary values $\{0, 1\}$ while node B has

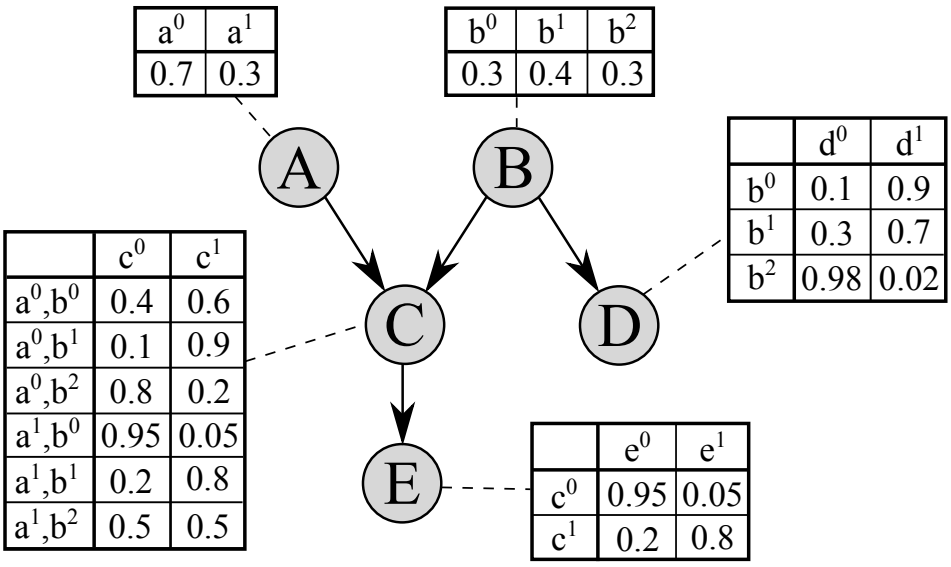


Figure 2.9: An example Bayesian network $\mathcal{B}^{example}$ with five nodes $\mathcal{X} = \{A, B, C, D, E\}$ and corresponding CPTs. Nodes A, C, D and E have binary values $\{0, 1\}$ while node B has three values $\{0, 1, 2\}$. Each node is connected to its CPT by a dashed line.

three values $\{0, 1, 2\}$. Dashed lines indicate the correspondence of the CPTs to the

nodes. Connections in the network as well as entries in the CPTs indicate the conditional dependencies. It can be seen for example, that node D only depends on its parent node B while node C is dependent on nodes A and B. Changing the point of view to independences, it can be seen that node E is conditionally independent of all other nodes given its parent C:

$$(E \perp A, B, D|C). \tag{2.23}$$

This means that once the value of C is known, no observation of nodes A, B or D changes the belief of node E. When investigating node C again under independence properties, the assumption that C depends only on its parents is not true any longer. Observing a value of E (a child of C) can apparently update the belief of node C. Thus, it cannot be expected that a node is conditionally independent of all other nodes given its parents as it can still depend on its children and even on further descendants. Thus, it can be noted that node C is only independent of node D given nodes A and B:

$$(C \perp D|A, B). \tag{2.24}$$

Following these statements, it can be further concluded that node B is independent of node A as A is neither a parent of B nor a descendant:

$$(B \perp A). \tag{2.25}$$

On the other hand, node A is obviously also independent of node B, but also of node D:

$$(A \perp B, D). \tag{2.26}$$

Considering conclusions of the previously discussed example network, a formal definition of a Bayesian network structure with respect to independence assumptions is given by Koller and Friedman [114] as follows:

Definition 5 *A Bayesian network structure $\mathcal{G}$ is a directed acyclic graph whose nodes represent random variables $X_1, \ldots, X_n$. Let $Pa^{\mathcal{G}}(X_i)$ denote the parents of X_i in $\mathcal{G}$ and $NonDescendants_{X_i}$ denote the variables in the graph that are not descendants of X_i. Then $\mathcal{G}$ encodes the following set of conditional independence assumptions, called the* local independencies*, and denoted by $\mathcal{I}_l(\mathcal{G})$:*

$$\textit{For each variable } X_i : (X_i \perp NonDescendants_{X_i}|Pa^{\mathcal{G}}(X_i)) \tag{2.27}$$

Namely, Definition 5 states that each node X_i is conditionally independent of its non-descendants given its parents.

Finally before formally defining a Bayesian network, the association between conditional independences and conditional probability distributions has to be clarified. Considering chain rule for probabilities from Equation 2.18, joint distribution $P(A, B, C, D, E)$ of the Bayesian network $\mathcal{B}^{example}$ can be decomposed as

$$P(A, B, C, D, E) = P(A)P(B|A)P(C|A, B)P(D|A, B, C)P(E|A, B, C, D) \tag{2.28}$$

without relying on any assumptions. Obviously, the decomposition of Equation 2.28 does not bring any advantages compared to the joint distribution itself. But the decomposed

form on the right hand side allows to incorporate independence assumptions given for example in Equations 2.23 - 2.26. For example from $(B \perp A)$ immediately follows that $P(B|A) = P(B)$. Hence, the second term on the right hand side of Equation 2.28 can be simplified. Following this concept, the simplified decomposition becomes

$$P(A, B, C, D, E) = P(A)P(B)P(C|A, B)P(D|B)P(E|C) \tag{2.29}$$

which is exactly in line with the defined conditional probability tables. Thus, for each variable, a factor can be computed that represents its conditional probability and each entry in the joint distribution can be calculated by building a product of these factors, [114]. The chain rule for Bayesian networks concludes as follows:

$$P(X_i, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|Pa^{\mathcal{G}}(X_i)) \tag{2.30}$$

where $\mathcal{G}$ is a Bayesian network graph over variables $X_i, \ldots, X_n$ and the factors $P(X_i|Pa^{\mathcal{G}}(X_i))$ are the individual CPDs. If a distribution P can be expressed as demonstrated in Equation 2.30, P *factorizes* according to $\mathcal{G}$ [101, 183].

Finally, the formal definition of a Bayesian network follows from the chain rule for Bayesian networks also presented by Koller and Friedman [114]:

Definition 6 *A* Bayesian network *is a pair $\mathcal{B} = (\mathcal{G}, P)$ where P factorizes over $\mathcal{G}$, and where P is specified as a set of CPDs associated with $\mathcal{G}$'s nodes. The distribution P is often annotated $P_{\mathcal{B}}$.*

D-separation The concept of d-separation [158, 159, 86, 203] describes the relationship between the graph structure of a Bayesian network and the probabilistic independences. Two variables X and Y in a Bayesian network $\mathcal{B}$ are d-separated given variable Z if for all path between X and Y,

- Z is a node of a diverging $(X \leftarrow Z \rightarrow Y)$ or a serial path $(X \leftarrow Z \leftarrow Y$ or $X \rightarrow Z \rightarrow Y)$ between X and Y and Z is observed, or
- Z is a node of a v-structure (converging connection $X \rightarrow Z \leftarrow Y$) and neither Z nor any of its descendent is observed.

In case of a v-structure $X \rightarrow Z \leftarrow Y$, node Z is also called a *collider*. Therefore, the conditional dependencies and independence relations in the probability distribution over a set of random variables are described by the DAG of a Bayesian network. Verma and Pearl [203, 204] as well as Chickering [51] showed that the d-separation criterion encodes not a unique DAG, but can encode several DAGs if and only if they share the same skeleton and the same set of colliders. A set of DAGs with equal skeleton and colliders is thus called *equivalence class* and its members are called to be *structure equivalent*. In other words, the same probability distribution and therefore also the same set of d-separations can be expressed in equivalent DAGs even if some edges are differently directed.

2.2.3 Parameter Learning for Bayesian networks

Section 2.2.2.2 introduced the formal definition of Bayesian networks and provided an example Bayesian network in Figure 2.9. Bayesian networks can either be created by human experts by designing the structure of the network based on their subjective beliefs of independences and filling the CPTs based on their experiences. For sure, when creating a network of considerable size, the hand-crafting approach is no longer applicable. Another possibility of creating a Bayesian network is by estimating the structure and CPDs of a probability distribution from a provided dataset. This section introduces how CPDs of a Bayesian network can be derived from a dataset when the structure is already known and the next section will address the more advanced task of learning the structure of a Bayesian network from data. As the main focus of this thesis is on structure learning, this section only gives a brief introduction to parameter learning. A more detailed overview is given in related publications [44, 41, 104, 95, 112, 114, 59].

For the task of estimating the parameters for a Bayesian network $\mathcal{B} = (\mathcal{G}, P)$ with $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ from a dataset $\mathcal{D}$, we assume that the network structure $\mathcal{G}$ is already known and that the values $x^i, \ldots, x^n$ of any node X are defined. The dependencies and independence relationships for each node are mapped by $\mathcal{G}$ and therefore the compositions of the individual CPTs for each node are known, too. The goal of parameter estimations is thus to fill in the conditional probability values in the CPTs by estimating these values from a given dataset $\mathcal{D}$. Obviously, the dataset has to "match" the network, meaning that each node in $\mathcal{G}$ is represented by a parameter in $\mathcal{D}$. More formally, $\mathcal{D}$ is defined by a list of *instances* defining example cases for the parameters X (also called *features*), where the values of X are $x^i, \ldots, x^n$ according to $\mathcal{B}$ and reflecting the independence assumptions of $\mathcal{G}$ in the parameter distribution. In other words, it can be assumed that the dataset has been randomly sampled from the original (yet unknown) probability distribution of B. In addition, the dataset is assumed to be fully observed, i.e. does not contain any missing variables, and that the data instances are independent and identically distributed (IID). An example of a dataset that has been randomly sampled from the Bayesian networks illustrated in Figure 2.9 is shown in Table 2.1. Note that it can already be seen from

Table 2.1: Randomly sampled dataset with ten instances from the Bayesian network of Figure 2.9. Each row contains an instance I_i with a value for each random variable $\mathcal{X} = \{A, B, C, D, E\}$ (columns).

Instances	**A**	**B**	**C**	**D**	**E**
I_1	a^1	b^1	c^1	d^1	e^1
I_2	a^1	b^1	c^1	d^1	e^1
I_3	a^0	b^2	c^1	d^0	e^1
I_4	a^0	b^1	c^1	d^1	e^0
I_5	a^0	b^1	c^1	d^0	e^1
I_6	a^1	b^0	c^0	d^1	e^0
I_7	a^0	b^2	c^1	d^0	e^1
I_8	a^1	b^1	c^1	d^0	e^1
I_9	a^0	b^1	c^1	d^1	e^1
I_{10}	a^0	b^0	c^1	d^1	e^1

these few instances that C is somehow related to E. For almost all instances, if $C = c^1$ then $E = e^1$ and if $C = c^0$ then $E = e^0$ which is totally in line with the CPT of node E in $\mathcal{B}^{example}$. Only for instance I_4, c^1 and e^0 do not match, but as the probability that $E = e^1$ if $C = c^1$ is only 0.8, this is not too surprising. Please adhere that with respect to the assumption that the instances have been sampled under IID, the probability distribution of the dataset converges to $\mathcal{B}_P$ with a growing number of instances m. Hence, the more instances a dataset contains, the better $\mathcal{B}_P$ is mapped and therefore the better the parameters can be estimated.

2.2.3.1 Maximum Likelihood Estimation

Probably the most native parameter estimator is the *Maximum Likelihood Estimation* (MLE) [77, 23]. Let the sampling of the values of each variable be controlled by an unknown constant θ which describes the frequency of the outcomes (values) of the parameter. For example for node A, θ describes the frequency of a^0 denoted as $M[a^0]$ in a dataset. Obviously as A is binary, it follows that the frequency of a^1 denoted by $M[a^1]$ is $1 - \theta$. The Likelihood function $L(\theta : \mathcal{D})$ for A is therefore defined as

$$L(\theta : \mathcal{D}) = \theta^{M[a^0]}(1 - \theta)^{M[a^1]}. \tag{2.31}$$

It practice, maximizing the logarithm of the likelihood function is more applicable:

$$l(\theta : \mathcal{D}) = M[a^0] \log \theta + M[a^1] \log(1 - \theta). \tag{2.32}$$

Equation 2.31 defines the Likelihood with respect to the introduced example of parameter A while the Likelihood function in the general case is defined by

$$L(\boldsymbol{\theta} : \mathcal{D}) = \prod_k \theta_k^{M[k]} \tag{2.33}$$

given the vector of counts. Once the Likelihood function is defined, parameters can be chosen to maximize the function

$$L(\hat{\boldsymbol{\theta}} : \mathcal{D}) = \max_{\boldsymbol{\theta} \in \Theta} L(\boldsymbol{\theta} : \mathcal{D}) \tag{2.34}$$

where Θ is the hypothesis space. Fortunately, it has been shown that the likelihood decomposes as a product of independent terms, one for each CPD in the Bayesian network [187, 57, 186]. Hence, each local likelihood can be maximized independently and finally combined to get the global MLE solution. Considering a random variable X with parents $Pa(X)$ where $P(X|Pa(X))$ is represented as a CPT, a parameter $\theta_{x|\boldsymbol{u}}$ can be defined for each entry in the table where $x \in Val(X)$ and $\boldsymbol{u} \in Val(Pa(X))$. Please note that $Val(Pa(X))$ denotes the set of all possible combinations of values of each parent of X. The maximum likelihood estimation for each parameter is therefore

$$\hat{\theta}_{x|\boldsymbol{u}} = \frac{M[\boldsymbol{u}, x]}{M[\boldsymbol{u}]} \tag{2.35}$$

where $M[\boldsymbol{u}, x]$ is the number of times, the parents of X are in configuration u and $X = x$ and $M[\boldsymbol{u}]$ is the frequency of $\boldsymbol{u}$. For clarifying this statement, Table 2.1 is considered as dataset and the CPT of node C is determined. The parent configuration set u is therefore $\{\{a^0, b^0\}, \{a^0, b^1\}, \{a^0, b^2\}, \{a^1, b^0\}, \{a^1, b^1\}, \{a^1, b^2\}\}$. Suppose parameter $\hat{\theta}_{c^0|a^0,b^0}$ should be estimated, then $\hat{\theta}_{c^0|(a^0,b^0)} = \frac{M[(a^0,b^0),c^0]}{M[(a^0,b^0)]}$ where $M[(a^0, b^0), c^0]$ is simply the number of instances where $A = a^0, B = b^0$ and $C = c^0$. Frequency $M[(a^0, b^0)]$ is the number of instances where $A = a^0, B = b^0$ respectively. In other word, entries of the CPTs can simply be calculated by counting the frequencies in $\mathcal{D}$.

As can be seen from the example dataset, $M[(a^0, b^0), c^0] = 0$ and therefore $\hat{\theta}_{c^0|a^0,b^0} = 0$ what immediately leads to $\hat{\theta}_{c^1|a^0,b^0} = 1$. This observation is different from the known probabilities given in Figure 2.9 where $\hat{\theta}_{c^0|a^0,b^0} = 0.4$ and $\hat{\theta}_{c^1|a^0,b^0} = 0.6$ concluding that the dataset is far to small to represent $\mathcal{B}_P$ correctly. From this example, a big disadvantage of MLE can be seen, namely that if a specific configuration is not present in the dataset, its probability is strictly set to zero meaning that it is impossible to happen. Other approaches are considering this drawback by incorporating a prior distribution for each configuration.

2.2.3.2 Bayesian Parameter Estimation

Maximum Likelihood estimation does neither consider prior distributions of variables nor update of the strength of belief if the amount of data grows. Another approach based on Bayesian statistics introduced in this section is considering these drawbacks, called *Bayesian Parameter Estimation* (BPE) [117, 114]. With BPE, the prior knowledge of parameters $\boldsymbol{\theta}$ is encoded by a prior probability distribution. The posterior distribution of parameters $\boldsymbol{\theta}$ with respect to the dataset $\mathcal{D}$ is therefore defined as

$$P(\boldsymbol{\theta}|\mathcal{D}) = \frac{P(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\mathcal{D})} \tag{2.36}$$

where $P(\mathcal{D}|\boldsymbol{\theta})$ is simply the Likelihood as defined in the previous section and $P(\boldsymbol{\theta})$ is the prior distribution over the possible values in Θ. The denominator $P(\mathcal{D})$ is the integration of the likelihood over all possible parameter assignments and is used as a normalization factor [114]. As with MLE, the posterior can be decomposed into a product of local terms:

$$P(\boldsymbol{\theta}|\mathcal{D}) = \prod_i P(\boldsymbol{\theta}_{X_i|Pa(X_i)}|\mathcal{D}). \tag{2.37}$$

Using a Dirichlet distribution as a conjugate prior to the multinomial distribution [114], the posterior of the local likelihood is described by

$$P(x|\boldsymbol{u}, \mathcal{D}) = \frac{M[x, \boldsymbol{u}] + \alpha_{x|\boldsymbol{u}}}{M[\boldsymbol{u}] + \alpha_{\boldsymbol{u}}} \tag{2.38}$$

where $\alpha_{x|\boldsymbol{u}}$ and $\alpha_{\boldsymbol{u}}$ are the Dirichlet prior parameters. Using the same example as with MLE, $P(c^0|(a^0, b^0), \mathcal{D}) = 0.\bar{3}$ when setting any $\alpha_{x|\boldsymbol{u}} = 1$ ($\alpha_{\boldsymbol{u}} = \sum_i \alpha_{x_i|\boldsymbol{u}} = 2$), what is far closer to the real value of 0.4 compared to the result with MLE. It can also be seen from Equation 2.38 that the more instances the dataset has, the less influence is given to the

prior and therefore the more belief is given to the data observations.

2.2.4 Structure Learning for Bayesian networks

In the previous section, estimating the parameters of a Bayesian network with respect to a given structure has been introduced. But in a more truthful setting, the structure is not known in advance and has to be learned before being able to estimate the parameters. There are several good reasons for learning a Bayesian network from a dataset instead of performing plain correlation analysis. First, Bayesian networks are intuitively considering containing higher order correlations and provide them in a form that is especially suitable for human understanding. Roughly speaking, a graph is just a picture mapping relations between objects and human beings can learn relations more efficient from illustrations than from raw numbers or text [130]. Further, Bayesian networks provide correlations in a combined structure which enables inference calculations for explanation or prediction [56, 141, 114]. In addition, the structure can help in determining new domain knowledge by discovering relations between nodes which is often used in in biological or medical domains where influence factors for specific diseases or reactions are searched [97, 83, 67, 28, 35, 164, 102]. Probably the most beneficial characteristic of Bayesian networks is that they try to estimate a statistical model of the underlying distribution of the dataset and are thus able to generalize [114]. In consequence, the network is able to reason instances that are not in the dataset which is important for example for classification tasks. Hence, Bayesian networks provide a broad application spectrum.

But as usual in real life, these advantages do come with some remarkable difficulties which make learning the "correct" structure of a Bayesian network nearly impossible. One reason is that as mentioned in Section 2.2.2.2, the independence assumptions of the dataset and the d-separation criterion do not lead to a unique graph but to a set of graphs sharing the same skeleton and colliders. Thus, the task of structure learning should be refined from searching the true structure to searching a structure that best represents the parameter independences of the dataset. The accuracy of estimating independences grows with growing number of instances and shrinks with the number of features due to basic probability theory [114]. Hence, the true independences can only be derived with a dataset of infinite instances. A limited dataset can thus only provide estimations. Another key aspect is the number of parameters and the number of their values. During parameter estimation, the Likelihood has been introduced that fragmented the dataset into subsets that can be counted for filling the fields of the CPTs. The more parents a node has and the more values these node have, the more fields are defined in the table where each configuration requires to be sufficiently often represented in the dataset to provide a good estimation. As the configuration grows exponentially, increasing the number of parents and the parameter values quickly results in a dataset with an insufficient amount of instances. Other major problems derive from the assumption that the instances are IID which is not true in most common domains and that the dataset is fully observed. In the purpose of this thesis, the dataset is further required to include categorical features only. While the IID assumption has to be made, the latter two drawbacks can be resolved. The structure of a partially observed dataset can either be learned by special algorithms summarized by Koller and Friedman [114] or by preliminary estimating the missing values which can be done by a variety of different methods [200, 70].

Although, structure learning algorithm that are handling datasets with continuous variables resulting in Gaussian networks have be proposed [84], the most common solution is to discretize [75, 81] continuous parameters into discrete bins.

Dependent on desired result, a Bayesian network can be learned to include many edges or to be sparse. Adding more edges to the network increases the chance to better consider the underlying distribution while a sparse network may miss some important edges. On the other hand, when the goal is to investigate interesting relations between parameters, too many edges can dissemble wrong relations of the domain. But even when doing density estimation, a sparse network structure should be preferred if the data is limited due to the exponential growth of the parent sets.

In general, there are three different domains of structure learning algorithms for Bayesian networks:

Constraint-based structure learning Constraint-based structure learning methods [160, 204, 131, 49, 188, 123] assume a Bayesian network to be a representation of independences. The network structure is built by reconstructing the independence statements from the dataset by performing independence queries of the form "Does P satisfy $(X_1 \perp X_3|X_2)$?" and provide an algorithm to answer these queries. Examples of these query answering approaches are *Single-Sided Hypothesis Tests* [78, 114] or *Deviance Measures* [129, 114]. For datasets with a growing number of parameters, the required independence queries are infeasible. The number of allowed parents for any node is hence often limited to reduce the query space. As the approaches for answering queries usually perform standard statistical hypothesis tests, their reliability can be parametrized. When setting the significance level too high, too many queries might be negated and on the other hand when setting the significance level too low, too many wrong independence assumptions might be accepted. These methods are thus sensitive to errors in these independence tests, where single mistakes are sufficient in some cases to mislead the whole network construction process [114].

Score-based structure learning When learning the structure of a Bayesian network based on scores, a space of all possible network structures that can result from the parameters in the dataset is built. This space of different structures is then searched to find the structure that matches the dataset best. Evaluation of the "best" network is performed by calculating a score for each network indicating how well the probability distribution in the structure fits to the one in the dataset. As the space of possible structures is super exponential [114], search methods are used to find the best structure within the space. For growing datasets with respect to the number of nodes, heuristic approaches are used as searching the whole search space is no longer feasible. Score-based structure learning is probably the most common approach where most state of the art structure learning algorithms belong to. Thus, section 2.2.5 introduces this approach in more detail and provides some of the most common score and search methods.

Learning an ensemble of structures Another possibility of structure learning is to learn a set of different structures instead of just a single one [114]. The prediction is then averaged over the set of possible structures as with Bayesian reasoning [55, 176]. Hence, these methods are also called *Bayesian model averaging*. In an ensemble of structures, a

single structure that deviates from the "true" structure is less severe to the result as it can be compensated by the other structures. Ensemble learning methods are less prone to failures and to overfitting.

2.2.5 Score-based structure learning

Score-based structure learning methods try to find the ideal structure for a given dataset by optimizing a score while navigating through the space of possible structures. To do so, a score function has to be defined that can determine a score for each individual structure that increases with increasing match of the probability distribution of the structure and the training data. Thus, the choice of scoring function is crucial to the learning result and a set of different scoring functions has been published in the last decades [37, 192, 121, 174, 20, 186]. Section 2.2.5.1 introduces some of the most common scoring functions which are also used in the novel algorithms presented in this thesis later on. Evaluating the scores of all possible networks and picking the one with the highest score as final result would be the first idea when using score methods for structure learning. But this approach is only applicable for datasets with a very reduced set of parameters and their values. As David Chickering showed in 1996 that learning Bayesian networks from data is $\mathcal{NP}$-complete [52] and that the number of possible structures is growing super exponential, the only manageable possibility of score optimization for most datasets is to use heuristic search methods. An introduction to structure search in provided in Section 2.2.5.2 and some interesting representatives of these methods are described at the end of this Section.

2.2.5.1 Scores

In Section 2.2.3.1, the Maximum Likelihood Estimation has been used to determine the values of the nodes conditional probability tables. It thus seems intuitive to use the Likelihood function as scoring function and search the network structure that maximizes the likelihood function. The Likelihood score $score_L$ of a graph $\mathcal{G}$ with respect to the training data $\mathcal{D}$ could be defined as

$$score_L(\mathcal{G} : \mathcal{D}) = l(\hat{\boldsymbol{\theta}}_{\mathcal{G}} : \mathcal{D}) \tag{2.39}$$

where $\hat{\boldsymbol{\theta}}_{\mathcal{G}}$ are the maximum likelihood parameters for $\mathcal{G}$ and $l(\hat{\boldsymbol{\theta}}_{\mathcal{G}} : \mathcal{D})$ is the logarithm of the likelihood. Koller and Friedman [114] provide a prove that the Likelihood score never prefers a simpler network structure over a more complex structure and thus always results in learning the network with most possible edges. As previously mentioned, learning goal is to get a sparse network to better generalize and avoid overfitting. Hence, using the Likelihood as a score metric for structure learning is not useful in most cases. Fortunately, the research community has presented a bunch of more advanced scoring metrics:

BIC/MDL Score The *Minimum Description Length (MDL) score* [37, 192, 121], which is also called *Bayesian Information Criterion (BIC) score* [174] extends the Log-

Likelihood score by a penalization factor to avoid overfitting. The score is defined as

$$score_{MDL} = score_{BIC} = l(\hat{\boldsymbol{\theta}}_{\mathcal{G}} : \mathcal{D}) - \frac{\log M}{2} Dim[\mathcal{G}] \tag{2.40}$$

where M is the total number of instances in $\mathcal{D}$ and $Dim[\mathcal{G}]$ is the number of independent variables in $\mathcal{G}$ and can be calculated by

$$Dim[\mathcal{G}] = \sum_i (|Val(X_i)| - 1)|Val(Pa(X_i))| \tag{2.41}$$

AIC Score A simpler form of BIC score is the *Akaike Information Criterion (AIC) score* [20] where only the dimension of $\mathcal{G}$ is used as penalization factor:

$$score_{AIC} = l(\hat{\boldsymbol{\theta}}_{\mathcal{G}} : \mathcal{D}) - Dim[\mathcal{G}] \tag{2.42}$$

Bayesian Score As usual in the Bayesian approach, a prior distribution is placed over possible network structures. The *Bayesian score* was introduced by Buntine in 1991 [42] and Cooper and Herskovits in 1992 [57] and generalized by Spiegelhalter *et. al.* in 1993 [186] and is defined as

$$P(\mathcal{D}|\mathcal{G}) = P(\mathcal{G}) \prod_i \prod_{\boldsymbol{u}_i \in Val(Pa^{\mathcal{G}}(X_i))} \frac{\Gamma(\alpha^{\mathcal{G}}_{X_i|\boldsymbol{u}_i})}{\Gamma(\alpha^{\mathcal{G}}_{X_i|\boldsymbol{u}_i} + M[\boldsymbol{u}_i])} \prod_{x_i^j \in Val(X_i)} \left[\frac{\Gamma(\alpha^{\mathcal{G}}_{x_i^j|\boldsymbol{u}_i} + M[x_i^j, \boldsymbol{u}_i])}{\Gamma(\alpha^{\mathcal{G}}_{x_i^j|\boldsymbol{u}_i})} \right] \tag{2.43}$$

where

$$\alpha^{\mathcal{G}}_{X_i|\boldsymbol{u}_i} = \sum_j \alpha^{\mathcal{G}}_{x_i^j|\boldsymbol{u}_i} \tag{2.44}$$

and $\alpha^{\mathcal{G}}_{x_i^j|\boldsymbol{u}_i}$ is the hyperparameter of the Dirichlet prior of value j of node X_i under parent condition $\boldsymbol{u}_i$. Symbol $\Gamma(\dots)$ represents the Gamma function[2] and $P(\mathcal{G})$ is the prior on the network structure that is constant over all structures and can therefore be ignored for score optimization. Again, $\boldsymbol{u}_i$ is a configuration of the values of the parents of node X_i and thus $M[\boldsymbol{u}_i]$ is the number of instances in the dataset where this configuration occurs. Term $M[x_i^j, \boldsymbol{u}_i]$ is the number of instances where parent configuration $\boldsymbol{u}_i$ is given and node X_i has value j.

In practice, $P(\mathcal{D}|\mathcal{G})$ can become small very fast, hence the logarithmic form is often used as scoring metric and thus $score_{Bayes}$ is

$$\sum_i \sum_{\boldsymbol{u}_i \in Val(Pa^{\mathcal{G}}(X_i))} \left[log \left[\frac{\Gamma(\alpha^{\mathcal{G}}_{X_i|\boldsymbol{u}_i})}{\Gamma(\alpha^{\mathcal{G}}_{X_i|\boldsymbol{u}_i} + M[\boldsymbol{u}_i])} \right] + \sum_{x_i^j \in Val(X_i)} \log \left[\frac{\Gamma(\alpha^{\mathcal{G}}_{x_i^j|\boldsymbol{u}_i} + M[x_i^j, \boldsymbol{u}_i])}{\Gamma(\alpha^{\mathcal{G}}_{x_i^j|\boldsymbol{u}_i})} \right] \right]. \tag{2.45}$$

In other words, for each node X_i, the parent configurations $\boldsymbol{u}_i$ are first determined from the structure where the score is calculated for. Then, for each composition of parent

[2]The Gamma function is an extension of the factorial ($n!$) to real and complex number arguments and is defined as $\Gamma(n) = (n-1)!$ [27]

values, the number of times they occur together in the dataset $M[\boldsymbol{u}_i]$ is counted and the first fraction in Equation 2.45 is calculated. Further, for each value j of X_i, the number of instances where $X_i = j$ and parent configuration $\boldsymbol{u}_i$ occurs together is counted. The results are summed up and added to the previously determined term. Please note that for all of these terms, the logarithmic value is used. The final score of the network structure is then simply the sum of all node scores.

It immediately follows, that the final score can be decomposed into local node scores as they are simply added together. The local score for a node can thus be calculated by knowing the nodes' parents only, independent of the rest of the structure. By investigating the score function of the Bayesian score, it can be seen that the score is biased to simpler structures, but with more data, the chance to build a more complex structure grows. Meaning that the Bayesian score balances model complexity and fit to data and thus reduces the risk of overfitting [114].

K2 Score *K2 score* [57] is a variation of the previously described Bayesian score where the hyperparameters $\alpha^{\mathcal{G}}_{x_i^j|\boldsymbol{u}_i}$ are all set to 1 and therefore $\alpha^{\mathcal{G}}_{X_i|\boldsymbol{u}_i} = |Val(X_i)|$. The right most fraction shown in Equation 2.45 then simplifies to $(M[x_i^j, \boldsymbol{u}_i])!$.

BDeu Score Another widely used variation of the Bayesian score is the *Bayesian Dirichlet equivalent uniform (BDeu) score* [96] where

$$\alpha^{\mathcal{G}}_{x_i^j|\boldsymbol{u}_i} = \frac{c}{|Val(X_i)||Val(Pa(X_i))|} \tag{2.46}$$

and thus $\alpha^{\mathcal{G}}_{X_i|\boldsymbol{u}_i} = c/|Val(Pa(X_i))|$. Variable c is a constant value that is set to $c = 1$ in most cases. The BDeu score yields a unique distribution over the hyper parameters and is score equivalent [168].

Score Decomposability As already mentioned while defining the Bayesian score, a big advantage of the scores introduced in the previous paragraphs is that these scores are all decomposable [114]. A local score of a node that only depends on its parents will be called *family score* for the rest of the thesis. The score of the whole network is thus defined as the sum of all family scores. When considering search methods in the next section, score decomposability will be proved to be very helpful as when navigating through the space of possible structures by altering a single connection in the graph, only the family scores of the affected nodes have to be updated instead of recalculating the complete network score.

2.2.5.2 Structure Search

Searching the network structure that best fits the training data can be defined as an optimization problem where the score is to be optimized. The search space is given by the set of all possible network structures that can be built from the parameters in the dataset [51, 97]. Koller and Friedman [114] described the search space as a graph over candidate solutions where each node is a possible network structure and the transitions between the nodes are defined by the following operators:

- *Add edge;*
- *Delete edge;*
- *Reverse edge;*

A search algorithm can then move through the graph by starting at a random structure and moving in the direction with the highest score improvement. It is clear that the three presented operators are sufficient to reach every structure in the graph. Indeed, only edge addition and edge removal would be sufficient to reach any structure from any point in the graph as edge reversal is nothing else than removing an edge and inserting the edge in reversed direction. But as the search method is supposed to only move in directions that increase the score, the operation of edge reversal is necessary. Consider for example an edge that is not part of collider, but the edge is "true", meaning the edge is also in the correct network structure where the dataset has been sampled from but in reversed direction. As the two nodes connected by the edge are not independent from each other, removing the edge in the first step of edge reversal will decrease the score. Adding the edge again in reversed order will increase the score to a higher level as in the previous direction. But as the score is first decreased, a naive search method might not consider going in that direction. If the edge is reversed in a single step, the increasing score is noticed by the search method and is considered to be a promising direction.

The vast majority of search methods are local search methods such as the *Greedy Hill-Climber* [42, 41] where at an initial step, a random network structure is selected, usually the empty network. In each step, the score of all neighbours in the search space graph is evaluated and the structure with the highest score improvement is chosen. This procedure is resumed until no neighbour has a higher score than the current structure. Within this procedure, neighbours are only evaluated if they do not extend a possibly defined in-degree (allowed number of parents per node) and don't violate the acyclicity of the network. Obviously, this simple approach has two significant drawbacks. First, the number of available operators at each point is very large. Second, as the algorithm terminates once no neighbour is available with better score, the search procedure has high risk to get stuck in local maxima. One approach to avoid local maxima is to restart the search procedure several times from randomly selected initial structures, called *Repeated Hill Climber*.

2.2.5.3 LAGD

The *Look Ahead in Good Directions Hill Climbing* algorithm (LAGD) [1] is a hill climbing search method that considers a set of directions instead of only one at each search step. Initially, an arbitrary DAG $\mathcal{G}_{old}$ is selected. Then, as with the naive hill climbing algorithm, the score differences to all neighbours of $\mathcal{G}_{old}$ is calculated but not only the single neighbour with highest score improvement is selected, but a set of l best neighbour structures is stored in a specific list. The same procedure is performed for the DAGs in the list where each DAG again produces a list of l best neighbours. This process is repeated recursively until the look ahead depth k is reached. In the last recursion step, only the network with the highest score improvement is chosen. After that, the structure with the highest score win rate is selected and set as $\mathcal{G}_{old}$ for the next iteration. Once no

score improvement can be found in all look ahead directions with depth k, the algorithm terminates and $\mathcal{G}_{old}$ is the final solution.

Obviously, the quality of learning result increases with increasing look ahead depth. On the other hand, execution time increases, too. The challenge is thus to find a good balance between quality and execution time. The LAGD algorithm has been shown to outperform Greedy Hill-Climbing, Repeated Hill Climbing and Simulated Annealing by Neubach *et. al.* [1]. Salehi and Gras [170] performed a detailed comparison of local search algorithms on a variety of different networks and concluded that LAGD performs best in nearly all cases and is also more time efficient in general.

2.2.5.4 Tabu Search

Another modification of the common hill climbing approach is called *Tabu Search* and has been developed by Glover *et. al.* [92, 90, 91]. Tabu Search performs hill climbing by adding and removing where structures that have been visited in the previous n steps are stored in a tabu list. Once a local optimum is reached, the least worse candidate of neighbour structures that is not in the tabu list is chosen. Tabu Search can further be parametrized by setting the number of runs used to traverse the search space. Experiments by Salehi and Gras [170] showed that Tabu Search is a comparable state of the art structure learning method that even outperformed LAGD in some rare cases.

2.2.5.5 K2

Efficiency of network structure learning can be boosted by predefining an order of variables X_i. Ordering variables means that if a variable X_i precedes another variable X_j in the ordering, structures containing edges $X_j \rightarrow X_i$ are prohibited what reduces the search space enormously. The *K2* structure search algorithm [57] uses a greedy hill climbing among the networks compatible with an ordering. In most cases, the order of the variables is not known and a search across the orders has to be performed first. Finding the best order can itself be seen as an optimization problem that can be solved by using scores and search methods [63, 208].

2.2.5.6 Simulated Annealing

Another often used heuristic search method is *Simulated Annealing* [38, 111] which ties an error boundary across network structures. As with other local search algorithms, a initial network structure is selected first. A candidate network structure is then generated randomly by performing one of the valid operations. If the score of the candidate network is higher that the score of the actual network, the candidate is accepted. Otherwise, the candidate is also selected, but only with a probability of

$$P = e^{-\left(\frac{\Delta_{score}}{t_i}\right)} \tag{2.47}$$

where Δ_{score} is the score difference between the actual and the candidate network and t_i is the *temperature* at iteration i. The temperature is initialized with a higher value t_0 and is slowly decreased with each iteration to tie down the error acceptance. In other words, with growing iterations, the acceptance of candidate networks is getting more and more

stricter. The procedure of controlled cooling has been inspired from material engineering where metal is cooled down slowly to avoid defects in its crystal structure. Hence, the name simulated annealing has also been inherited from the area of metal engineering when it was first used as optimization technique [111].

Salehi and Gras [170] showed that Simulated Annealing does perform adequate for small networks with less instances compared to K2 and repeated hill climbing but had the worst results in all test networks with a sufficient number of instances in the dataset. On the other hand, Laarhoven and Aarts [202] showed that if t is annealed logarithmically, Simulated Annealing is guaranteed to converge to the global optimum although the computational cost is of prohibitive expense in that case.

2.2.6 Variable order

A variable order for a Bayesian network is an ordered list of all nodes of the network so that if a node X precedes another node Y, node X can not be a descendent of Y. Getting a valid order for a Bayesian network from data is computationally infeasible for larger networks and thus, heuristic methods have to be used. Most of these methods try to order the variables by building weights for each variable in the dataset. The scoring function approach [156] for example tries to determine the weights of the variables by the naive assumption, that a parent in the true network has better chances to be a parent in the learned network than a child in the true network. Thus, the algorithm simply counts the positive, zero and negative score gains of any variable X_i to be a single parent of an other variable.

$$w_i = \sum_{X_j \in X \setminus i} sgn(s_{X_j, Pa(X_j)=\{X_i\}} - s_{X_j, Pa(X_j)=\emptyset}) \tag{2.48}$$

The symbol $s_{X_j, Pa(X_j)=\{X_i\}}$ is the score determined by any scoring function of variable X_j having only parent X_i and $s_{X_j, Pa(X_j)=\emptyset}$ is the score of X_j without a parent. The signum function is indicated by sgn and is used as local score of different structure cannot be compared. The final ordering is determined by sorting the list of nodes according to the determined weights.

2.2.7 Bayesian networks: Related work

Since Pearl [159] formulated the notion of a Bayesian network as a data structure encoding independence relationships in 1988, remarkable research interest has been given to Bayesian networks resulting in a multitude of publications and applications. Heckerman *et. al.* used Bayesian networks to build their famous Pathfinder diagnostic tool [98, 99, 100] that had the ability to outperform human expert physicians. This application as well as several other large diagnostic networks [24, 179, 134] had major impact on the rise of popularity of Bayesian networks.

The theoretical properties of Bayesian networks have been studied in several different works where attention was in particular given to the computational feasibility of structure learning [52, 60, 132, 53]. In 1990, Cooper [56] was the first who formally analysed the computational complexity of probabilistic inference in Bayesian networks before Kim and Pearl [110] proposed a message passing method to challenge the general problem of inference in graphical models.

With growing networks, the feasibility of sufficiently good parameter estimation reduces dramatically when conditional probability tables are used due to the combinatorial problem previously described in this thesis. Many approaches have been published that try to resolve these problems, for example by using structured CPTs [187, 42, 69]. Another method originally performed by Buntine for classification tasks [43] and applied to Bayesian networks by Friedman and Goldszmidt [80] is to replace the CPT with a tree representing the conditional probability distribution. Another approach is to learn Bayesian networks with linear Gaussian dependencies, which was analysed by Heckerman and Geiger [94] in 1995.

Section 2.2.5.2 briefly addresses the task of structure learning for Bayesian networks and a set of local search methods has been introduced that try to resolve the problem of finding the global optima. A different approach of avoiding local optima by perturbing the training data has been presented by Elidan *et. al.* [73]. Much work has been done in reducing the computational expense of search methods, for example by intelligent caching of sufficient statistics to accelerate score calculations and by holding already calculated scores in cache tables [25, 115, 46]. Other methods try to reduce the search space by considering only a subset of structures where most of these approaches are using an Markov chain Monte Carlo (MCMC) strategy over the space of Bayesian networks [125, 124, 89]. Friedman and Koller further improved this idea by using MCMC on orderings instead of structures and showed that their methods outperforms the structure variant [82]. Teyssier and Koller [126] also showed that finding the highest scoring network consistent with a given order is not $\mathcal{NP}$-hard and published an algorithm that outperformed state of the art methods in score and time. Another type of methods has been investigated where finding the correct network structure can be guaranteed and performs in polynomial time if a bounded in-degree is defined [188, 49]. For example, Koivisto *et. al.* [113] presented an exact structure learning algorithm based on an ordering approach that is feasible for networks with less than 30 nodes. Two years later, in 2006, Silander *et. al.* [181] showed another exact method that also uses variable ordering but is much simpler to implement. A further development of Silander's methods by using dynamic programming in combination with a divide and conquer strategy was introduced by Parviainen *et. al.* in 2009 [157]. Nevertheless, both of these algorithms are also limited to a maximum of around 30 nodes. And yet another improvement has been made by Jaakkola *et. al.* [105] who used LP-relaxations to finally overcome the limitations to 30 nodes and showed that their method nearly always finds the global solution in short time. In 2012, Silander *et. al.* presented a novel advancement of their methods that can handle 32 nodes and is restricted by memory only [180].

Barabasi *et. al.* presented basic network topology analysis where they showed that at general, their exist two possible network architectures, *Random networks* and *Scale-Free networks* [31, 30]. While for random networks, the number of edges each node is connected to follows a Poisson distribution with a bell shape, the distribution in Scale-Free networks is *L*-Shaped. Meaning that there are just a few nodes that have a high amount of connections where most other nodes have only around two to three edges. They further detected, that most metabolic networks follow the Scale-Free topology.

Bayesian networks have also been used for prediction and classification tasks [79, 50], especially in medical and biological domains in the last decade. Extensive studies of using Bayesian networks with gene expression data and genetic regulatory networks have

been performed by Stetter *et. al.* [67, 66, 143, 47]. Gevaert *et. al.* used a Bayesian network learned from a mixture dataset of clinical and microarray data to predict the prognosis of breast cancer [88]. Bodén *et. al.* [35] built a Bayesian network that uses support vector machines to represent the conditional probability distributions and modelled protein associations related to Leukemia. Bayesian networks have also been used to analyse lifestyle and metabolic predictors of obesity by Aussem *et. al.* in 2010 [28]. Isci *et. al.* [102] performed pathway analysis of high throughput biological data by using a Bayesian network framework. Another interesting approach that uses Bayesian network theory combined with MCMC method is given by Qingfeng [164] *et. al.* where they apply the MCMC method to generate a sequence of samples from a probability distribution, by which to approximate the distribution.

Obviously, this section is a highly limited selection of publications related to Bayesian networks, showing only the publications where the author got in touch with. Searching for "Bayesian network" on Googles scientific search engine *Google Scholar* results in approximately 754000 matches[3] which clearly underlines the research interest in that area. A very good reference for Bayesian networks is for example the book of Koller and Friedman [114] as they present detailed literature reviews to all of its chapters.

[3]Search performed at http://scholar.google.com on 09. February 2013.

2.3 Correlation

2.3.1 Pearson product-moment correlation coefficient

The Pearson product-moment correlation coefficient [155] denoted as ρ is a measure of linear dependence between two variable X and Y and its value is in the range of -1 to 1. The coefficient is defined as the covariance divided by the product of their standard deviations:

$$\rho(X,Y) = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{2.49}$$

where $cov(X,Y)$ is defined as

$$cov(X,Y) = E\left[(X-\mu_X)(Y-\mu_Y)\right] = E[XY] - \mu_X\mu_Y. \tag{2.50}$$

Letter μ denotes the mean value and $E[\dots]$ describes the expected value, meaning the weighted average of all possible values. A value of $\rho(X,Y) = 0$ implies that there is no linear correlation between X and Y while values in the range of $[-1;0[$ and $]0;1]$ denote the probability that X and Y are linear dependent. If $\rho(X,Y) = 1$, all data points of X and Y are lying on a straight line in the X,Y space where Y increases with increasing X. On the other hand, if $\rho(X,Y) = -1$, data points also lie on a straight line but Y decreases when X increases.

2.3.2 Cramér's V correlation coefficient

A specialized correlation coefficient for nominal data has been introduced by Harald Cramér in 1946 [58] and is computed by using the χ^2 statistics to measure independence between two variables X and Y defined by Pearson [161] as

$$\chi^2 = \sum_{i=1}^{r}\sum_{j=1}^{c} \frac{(O_{ij} - E_{ij})^2}{E_{ij}} \tag{2.51}$$

where r is the number of values of X, c is the number of values of Y, O_{ij} is the observed frequency count where variable X has value i and variable Y has value j and E_{ij} is the expected frequency count respectively. Cramér's V correlation coefficient between x and Y is thus defined as

$$\phi_c(X,Y) = \sqrt{\frac{\chi^2}{N(k-1)}} \tag{2.52}$$

where N is the total number of instances and k is either the number of columns or the number of rows, whichever is less. The range of ϕ_c is between 0 and 1 and is interpreted the same as the Pearson correlation coefficient.

2.4 Weka - a machine learning framework in Java

As machine learning attracts much attention in computer science, several frameworks have been developed for Java. Probably, the most established one is the open source li-

brary *Weka* [72]. Weka has been developed at the University of Waikato in New Zealand and provides a collection of algorithms for data classification, preprocessing, regression, clustering, association rules and data visualization. These techniques can either be accessed from the provided shell via a graphical user interface, from the command line or directly from Java code via a transparent API provided. To handle datasets, Weka defines a new data format called ARFF (Attribute-Relation File Format) where each ARFF file describes a single dataset.

In addition, Weka also provides support for using Bayesian networks as classifiers, randomly sampling data from Bayesian networks and learning Bayesian networks from data [39]. Further, a Bayesian network editor is available where networks can be visualized, nodes and arcs can be manipulated and entries of the CPTs can be edited. For learning networks from data, Weka offers a variety of different score implementations including Bayes, BDue, MDL and AIC scores. Local search methods introduced in Section 2.2.5.2, namely hill climber, repeated hill climber, K2 and simulated annealing are available, too. Parameters can be estimated for example by Bayesian parameter estimation.

2.5 Benchmark networks

The novel structure learning algorithms introduced in upcoming chapters are compared to state of the art structure learning methods by validating their learning performance on a set of commonly used benchmark Bayesian networks. The networks have all been downloaded in BIF (Bayesian Interchange Format) from the *bnlearn* website [175] which provides a repository for Bayesian networks, and have been translated into a XML based version readable by Weka called BIF XML by using a self-developed parser. Illustrations of these benchmark networks are provided in Appendix A. Further, a collection of artificially generated networks with different structure characteristics is used to study the behaviour of the novel methods under different conditions.

2.5.1 Cancer and Earthquake

The two smallest benchmark networks *Cancer* and *Earthquake* both include only 5 nodes which are connected by 4 arcs and have been introduced as example Bayesian networks in the text book of Korb and Nicholson in 2010 [116]. Both networks share the same structure and have a maximal in-degree (number of parents) of 2 and an average degree (number of arcs connected to a node) of 1.6. The conditional probability table of node Cancer of the Cancer network is strongly biased to the value *False* in all cases of parent values and thus seems to be independent of both parents. Meaning, that the CPT does not fit well to the structure. It is not sure if this was intended by the authors or if there is an error in the database where the network has been retrieved from. Nevertheless, the Cancer network was kept unchanged to be comparable to other publications.

2.5.2 Asia

The Asia network, which is also called Chest Clinic network, has been introduced as an example network by Lauritzen and Spiegelhalter in 1988 [122] and is often used when explaining the principles of Bayesian networks. As this network has been generated for

example purposes only, it should not be used for real medical decision making. Nevertheless, it can serve as a benchmark network due to its simple structure containing only 8 nodes connected by 8 edges and its clear distributions. The network is characterized by an average degree of 2.00 and a maximum in-degree of 2.

2.5.3 Insurance

Binder *et. al.* [34] introduced a novel approach to learn networks with known structure and hidden variables in 1997. Within this work, they published a network generated for car insurance risk estimation. The *Insurance* network became an often used benchmark networks for learning Bayesian networks from data in the last years. The network is built of 27 nodes which are connected by a total of 52 arcs. The network structure leads to an average degree of 3.85 and a maximum in-degree of 3.

2.5.4 Alarm

One of the most famous benchmark networks for learning Bayesian networks from data is the ALARM (A Logical Alarm Reduction Mechanism) network [33] developed by Beinlich *et. al.* in 1989. ALARM was initially introduced as an alarm message system to monitor patients where probabilities are calculated for eight different diagnoses by encoding 16 findings and 13 intermediate nodes. The network therefore consists of 37 nodes that are connected by 46 arcs. The average degree over all nodes is 2.49 and the maximum in-degree is 4.

2.5.5 Barley

The *Barley* network has been created to design a prototype decision support system for growing malting barley without the need of pesticides by Kristensen and Rasmussen [118] in 2002. The network consists of 48 nodes and 84 arcs with an average degree of 3.5 and a maximum in-degree of 4. In contrast to other benchmark networks where nodes are defined on average by 2.68 categories, nodes of Barley network have 8.7 parameters on average. The large number of parameters make the Barley network to one of the most challenging networks for structure learning used in this thesis as the number of instances required to reflect the underlying distribution grows with the number of possible parameter combinations.

2.5.6 Hailfinder

Lastly, the biggest real benchmark network used in this thesis is the *Hailfinder* network created 1996 by Abramson *et. al.* [2] in order to forecast severe weather in North-Eastern Colorado. The network has been build from meteorological data and expert judgement on both experience and physical knowledge and was the first Bayesian network used in the field of meteorology. It is built of 65 nodes connected via 66 arcs with an average degree of 2.36 and a maximal in-degree of 4.

2.5.7 Artificially generated networks

In order to validate the behaviour of the novel structure learning algorithms under different network configurations, a collection of pseudo-random networks is generated synthetically. The networks are pseudo-random in the way that they are randomly sampled but are forced to comply with predefined characteristics. These structure characteristics are:

- Number of nodes `[N]`
- Number of arcs `[A]`
- Cardinality (Number of values) of the nodes `[C]`
- Maximum in-degree `[P]`

The networks are generated by a modified version of the `BayesNetGenerator` class provided by Weka which first generates a tree structure to ensure a connected graph and afterwords adds more arcs if specified. As Wekas original java class lacks the functionality of specifying a maximum in-degree, the class has been modified. The generated networks are named by their characteristics using the pattern `n[N]_p[P]_a[A]_c[C]` where `[N]` is replaced by the number of nodes, `[P]` by the number of maximal parents per node, `[A]` by the total number of arcs in the network and `[C]` by the cardinality of the nodes. If more than one network with the same configuration is generated, the template is preceded by an upper-case alphabetic letter. An example network with five binary nodes, four edges and a maximal in-degree of two would therefore be named as `A_n5_p2_a4_c2`, where a second network with same configuration is named as `B_n5_p2_a4_c2`. A possible disadvantage of this method is that all nodes have the same cardinality. Nevertheless, networks that require to have different number of variables for their nodes can be generated manually in the Weka Bayesian network editor.

2.5.8 Sample datasets from networks

Datasets are sampled by using Wekas `BayesNetGenerator` class which first determines an ordering that guarantees that all parents of a node precede the node in the ordering. The values of the nodes can therefore be randomly picked from its conditional probability distribution by starting with the first node in the ordering. For each upcoming node, the values of all its parents are ensured to be known by definition of the ordering.

2.6 Development environment

All experiments presented in this thesis have been performed on an Intel(R) Core(TM) i7-2600K CPU @ 3.40 GHz processor running the 64 bit version of Windows 7 with service pack 1 installed and uses 16 GB RAM. The Weka API has been used in version 3.7.3. All code presented in this thesis has been written in Java 1.6 using the Eclipse Indigo development environment. The *Physarum Solver* algorithm has been implemented with respect to the theorems presented by Tero *et. al.* [194] described in Section 2.1.3 using the Apache Commons Mathematics library [199] for solving the linear equation system.

CHAPTER 3

Correlation based Physarum Learner (C-PhyL)

This chapter introduces the C-PhyL algorithm [172, 173], which is a novel method for learning the structure of a Bayesian network from a given dataset by using a totally new approach on how to determine the structure. For the scope of this thesis, datasets are restricted to be categorical.

3.1 The C-PhyL algorithm

Instead of using constraints or optimizing a score, the basic idea of the C-PhyL algorithm is to initially build a fully connected Bayesian network graph from which unconsolidated connections are pruned by using the *Physarum Solver*. In the following of this thesis, a graph describing a maze as it has been done by Tero *et. al.* (see Figure 2.6 and Section 2.1.3) that can be solved by the *Physarum Solver* is called *Physarum-Maze*. Tero *et. al.* indicated nodes by N and connections by M. As nodes for Bayesian networks have been referred to as X in the previous sections, nodes will be indicated by X for the rest of this thesis. Connections in the Physarum-Maze are further referenced by M as long as they are not directed. A directed edge is indicated by E.

3.1.1 Building a Physarum-Maze from data

The first step of C-PhyL is to initialize a fully connected Physarum-Maze from the dataset. A Physarum-Maze is generated by adding a node for each attribute in the dataset and adding an undirected connection between each pair of nodes, so that the maze is fully connected. The example dataset shown in Figure 3.1 contains four attributes $\boldsymbol{X} = \{A, B, C, D\}$ and five instances. A Physarum-Maze of four nodes is created and each node is connected to each other node in the graph. Please note, that at this point, no node is considered to act as a food source. In order to get a valid Physarum-Maze, each connection between two nodes X_i and X_j needs to have a defined length L_{ij} and a initial conductivity D_{ij}. As has been done by Tero *et. al.*, initial conductivity values are set randomly in the range of D_{min} and D_{max}. Defining length L_{ij} of the connection between

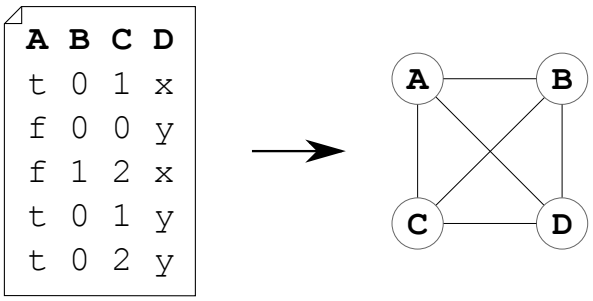


Figure 3.1: An example dataset containing four nodes is transformed into a fully connected Physarum-Maze by inserting a node for each variable.

two nodes is the key task of learning a Bayesian network structure with the *Physarum Solver*. Here it is suggested, that the length of the connection should represent the level of independence of the two nodes in the dataset. Therefore, by approximating true independence, a correlation coefficient between the connected nodes is calculated. Nodes in Bayesian networks considered here are taken to be nominal or ordinal and the connections in the *Physarum Solver* are undirected. Therefore, a symmetric correlation coefficient for nominal data is used, the Cramér's V correlation coefficient ϕ_c [58]. The correlation coefficient is in the range of 0 and 1 where a higher value indicates higher correlation. The *Physarum Solver* is aimed to keep edges for nodes that are correlated to each other. Hence, the length L_{ij} should be short for higher correlated variables and large for uncorrelated variables. The length is thus calculated regarding ϕ_c by

$$L_{ij} = (10(1 - \phi_{c_{norm}}(X_i, X_j) + l))^{\gamma} \tag{3.1}$$

where $l \leq 0.1$ is a minimum distance as the *Physarum Solver* cannot handle distances of zero length. The normalized correlation coefficient

$$\phi_{c_{norm}}(X_i, X_j) = \frac{\phi_c(X_i, X_j) - \phi_{c_{min}}}{\phi_{c_{max}} - \phi_{c_{min}}} \tag{3.2}$$

is subtracted from 1 to get short length for high correlations. Further, the result is multiplied by 10, so that the length is growing if γ is greater than one. The exponent $\gamma \geq 1$ can be used to increase the differences between the length values. Assuming a high correlation between node A and node B in the example of Figure 3.1 and low correlation between node A and node D, the final Physarum-Maze might look similar to the one shown in Figure 3.2, where the difference in line thickness indicates different conductivities. The procedure of building a Physarum-Maze from a dataset can be seen in Algorithm 1, where $calculateCramersV(X_i, X_j)$ returns Cramér's V correlation coefficient and function $Rand(D_{min}, D_{max})$ returns a random value between D_{min} and D_{max}.

First, an empty nodes array and an empty connections array is initialized and the minimum and maximum value of ϕ_c is set to 1 and 0. For each attribute X_i of the dataset, a new node is created and added to the nodes array. For each other attribute X_j, the correlation coefficient is calculated and minimum or maximum values are updated if required. After all nodes have been added to the nodes array, for each possible pair

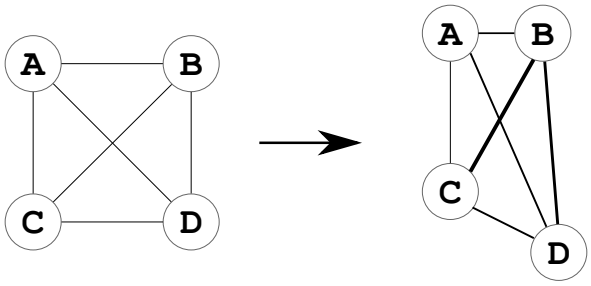


Figure 3.2: The initialized Physarum-Maze illustrated in Figure 3.1 is updated by applying length and conductivity values. Note that for this figure, L_{ij} and D_{ij} have been set to randomly chosen values to give an example.

of nodes, the previously determined value $\phi_c(X_i, X_j)$ is normalized and the length L_{ij} is calculated according to Equation 3.1. The conductivity D_{ij} is chosen randomly in the range of D_{min} and D_{max}. A new connection M_{ij} is created with L_{ij} and D_{ij} and is added to the connections array. The result of Algorithm 1 is a list of nodes and a list of connections defining a fully connected Physarum-Maze.

3.1.2 Rank connections by using the *Physarum Solver*

Once the Physarum-Maze has been built, the *Physarum Solver* can be used to find connections that should be part of the final Bayesian network by iterating over all possible pairs of nodes. For each pair $X_i \leftrightarrow X_j$, the direct connection between X_i and X_j is removed. The two nodes are set to be food source nodes and the *Physarum Solver* is applied to the Physarum-Maze to find the shortest path from X_i to X_j. The rank of the connections that remained in the Physarum-Maze after the *Physarum Solver* has terminated is increased by 1. Figure 3.3 shows an example iteration where $X_i = A$ and $X_j = B$ in the Bayesian network example.

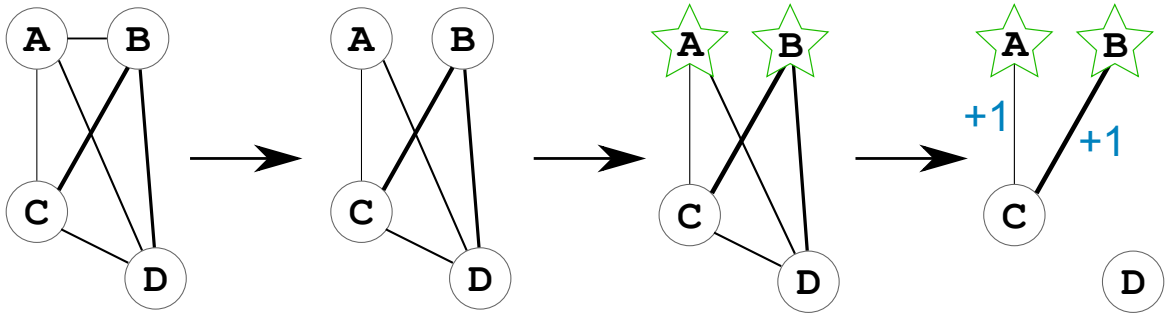


Figure 3.3: The *Physarum Solver* is used to find the shortest path between nodes $X_i = A$ and $X_j = B$. First, the direct connection between A and B is removed. Next, nodes A and B are set to be food sources and the *Physarum Solver* is applied. Finally, ranks of remaining connections after the *Physarum Solver* has terminated are increased by 1.

The previously removed connection between X_i and X_j is reinserted. The nodes X_i and X_j are reset to be normal nodes and the conductivities of the connections are reset

Algorithm 1 C-PhyL: Building Physarum-Maze from dataset

Require: nodes[], connections[]
Require: dataset
  $\phi_{c_{min}} = 1$
  $\phi_{c_{max}} = 0$
  nodes ← Empty
  connections ← Empty
  for all variable X_i in dataset **do**
    Add X_i to nodes[]
    for all variable $X_j \neq X_i$ in dataset **do**
      $\phi_c(X_i, X_j) =$ `calculateCramersV`(X_i, X_j)
      if $\phi_c(X_i, X_j) > \phi_{c_{max}}$ **then**
        $\phi_{c_{max}} = \phi_c(X_i, X_j)$
      end if
      if $\phi_c(X_i, X_j) < \phi_{c_{min}}$ **then**
        $\phi_{c_{min}} = \phi_c(X_i, X_j)$
      end if
    end for
  end for
  for all node pair X_i and X_j in nodes[] **do**
    $\phi_{c_{norm}}(X_i, X_j) = \frac{\phi_c(X_i, X_j) - \phi_{c_{min}}}{\phi_{c_{max}} - \phi_{c_{min}}}$
    $L_{ij} = (10(1 - \phi_{c_{norm}}(X_i, X_j) + l))^{\gamma}$
    $D_{ij} = Rand(D_{min}, D_{max})$
    $M_{ij} =$ new connection(L_{ij}, D_{ij})
    Add M_{ij} to connections[]
  end for

to their initial values before the next node pair is evaluated. This procedure can also be seen in Algorithm 2, where $Index(X_j) > Index(X_i)$ means that each pair is considered only once as in an undirected graph $M_{ij} = M_{ji}$.

After all iterations have finished, each node has been set as food source node exactly $|\boldsymbol{X}| - 1$ times, where $|\boldsymbol{X}|$ is the total number of nodes. The result after all iterations is an undirected fully connected graph, where each connection has a rank between 0 and $\frac{|\boldsymbol{X}|(|\boldsymbol{X}|-1)}{2} - 1$. A rank of zero means that the connection has never survived in any of the iterations. As the length of connections are indirectly proportional to the correlation between the connected nodes, a connection between nodes that are highly correlated is expected to survives more often than a connection between uncorrelated nodes. In other words, C-PhyL forces the *Physarum Solver* to find indirect paths that explain the correlation between two nodes, by blocking the direct path between these two nodes.

3.1.3 Build Bayesian network from ranked connections

The final step of the C-PhyL algorithm is to build the Bayesian network from the ranked connections. The basic problem on transforming the fully connected ranked Physarum-Maze into a Bayesian network is that the connections of the *Physarum Solver*, and there-

Algorithm 2 C-PhyL: Applying *Physarum Solver* on Physarum-Maze

```
Require: nodes[], connections[]
  Initialize rank for all connections with 0
  for all nodes X_i do
    for all nodes X_j do
      if Index(X_j) > Index(X_i) then
        Disable connection X_i ↔ X_j
        Set X_i and X_j as food source nodes
        survived[] = PhysarumSolver()
        for all survived do
          Increase connection rank by 1
        end for
        Reset X_i and X_j to normal nodes
        Activate connection X_i ↔ X_j
        Reset conductivities to initial values
      end if
    end for
  end for
```

fore the ranked connections in the Physarum-Maze, are undirected whereas a Bayesian network requires directed edges. This issue is solved by calculating a variable ordering as described in Section 2.2.6 and directing the edges so that the ordering is not violated. In other words, if weight w_i of node X_i is higher than weight w_j of nodes X_j, connection M_{ij} is directed to edge $E_{ij} = X_i \rightarrow X_j$.

Another difficulty is to determine the connections that should be added to the final Bayesian network. Obviously, the connections are added in the order of their rank where the highest ranked connection is added first. But the amount of connections to be added is unclear and not easy to define. Thus, the average rank value of all connections that have survived at least once (meaning that their rank is not zero) is calculated and connections are only considered to be added as long as their rank is higher than the average value. Further, a connection is only added to the Bayesian network if the score of the network increases when adding the connection. Nevertheless, connections with a rank lower than the average rank or connections resulting in a score decrease can be added if the child node of the connection is yet unconnected to any other node to avoid isolated nodes in the graph, see Algorithm 3.

The algorithm requires as input a list of connections sorted in descending order by their ranks and a Bayesian network $\mathcal{B}$ having the same nodes $\mathcal{X}$ as the Physarum-Maze and an empty set of edges $\mathcal{E}$. Once connections have been added according to the previously described procedure, the conditional probability distributions of $\mathcal{B}$ regarding $\mathcal{G}$ are estimated using any parameter estimation method (indicated by function estimateCPT()) implemented in Weka.

Algorithm 3 C-PhyL: Generate Bayesian network from ranked Physarum-Maze

Require: ranked connections
Require: variable order **w**
Require: Bayesian network $\mathcal{B}$ defined by graph $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ where $\mathcal{E} = \emptyset$
Ensure: connections sorted by rank, highest first
  $\theta \leftarrow$ average of connection ranks where $rank \neq 0$
  for all connection M_{ij} **do**
    if $w_i > w_j$ **then**
      $E_{ij} = X_i \rightarrow X_j$
    else
      $E_{ij} = X_i \leftarrow X_j$
    end if
  end for
  for all connection E_{ij} **do**
    $s_{E_{ij} \in \mathcal{E}}$ = score of network including E_{ij}
    $s_{E_{ij} \notin \mathcal{E}}$ = score of network without E_{ij}
    if (Rank of $E_{ij} > \theta$ **and** $s_{E_{ij} \in \mathcal{E}} > s_{E_{ij} \notin \mathcal{E}}$) **or** node X_j is isolated **then**
      Add E_{ij} to $\mathcal{E}$
    end if
  end for
  `estimateCPT()`

3.2 Parameter examination

The C-PhyL algorithm depends on a variety of configuration parameters including those of the *Physarum Solver* that have been shown by Tero *et. al.* to be highly sensible. Table 3.1 gives an overview of these parameters:

Column *Component* indicates to which module the parameter belongs to and column *Source* reminds in which algorithm or equation the parameter is used. The last column defines default values for parameters which are either derived from Tero's experiences or from preliminary experiments. For tests presented in this section, parameters are set to their default values if not stated otherwise.

For validating learning results of C-PhyL under different parameter configurations, datasets of 1000 instances are sampled from artificially generated networks described in Section 2.5.7. Next, C-PhyL is applied to a sampled dataset for learning a Bayesian network structure. The learned structures are compared to each other by determining their Bayesian scores which are calculated using Weka's built in scoring function for Bayesian networks.

3.2.1 Influence of D_{min} and D_{max}

The influence of D_{min} and D_{max} on the final learning result is determined by comparing learning performance of 75 artificial networks with different settings. The networks vary in their number of nodes from 5, 10, 15, 20 to 25. Further, nodes in these networks can have a maximal number of parents of either 1, 2, 3, 4 or 5 and a cardinality of 2, 3 or 4,

Parameter	Component	Source	Initial value
D_{min}	*Physarum Solver*	Algorithm 1	0.5
D_{max}	*Physarum Solver*	Algorithm 1	1.0
$f(Q)$	*Physarum Solver*	Equation 2.10	Q^{μ}
μ	*Physarum Solver*	Equations 2.11, 2.12	1.0
α	*Physarum Solver*	Equation 2.12	22.0
I_0	*Physarum Solver*	Equation 2.13	1.0
l	C-PhyL	Equation 3.1	0.1
γ	C-PhyL	Equation 3.1	2.0

Table 3.1: Overview of configuration parameters for the C-PhyL algorithm. Columns *Component* and *Source* show where the parameter is used within C-PhyL. The last column *Initial value* defines the default value of the parameter that is used for experiments if not stated otherwise.

resulting in a total of 75 networks. Five ranges of D_{min} and D_{max} are compared, namely $[0.2, 1.0]$, $[0.5, 1.0]$, $[0.8, 1.0]$, $[0.5, 0.8]$ and $[0.7, 0.8]$. These settings lead to a total number of 375 experiments running the C-PhyL algorithm where the goal is to determine if any of the given ranges for D_{min} and D_{max} can be preferred.

For 73 out of 75 tested networks, all five ranges of D_{min} and D_{max} delivered exactly the same results. For the other two datasets, the difference in score improvement and other validation metrics varied in a negligible range. Thus, it can be concluded that the initial values for D_{min} and D_{max} do not have any influence on learning results. They are therefore set to $D_{min} = 0.5$ and $D_{max} = 1.0$ according to experiments presented by Tero *et. al.*. This result is not surprising as Tero *et. al.* [195] reported that the initial state of the conductivities is not important when $f(Q) = Q^{\mu}$ with $\mu = 1.0$ and that the shortest path always survives in that case.

3.2.2 Comparing Equation 2.11 and 2.12 as $f(Q)$ method

To find out which of the two expressions for $f(Q)$, Equation 2.11 and 2.12, is more suitable for learning Bayesian networks from data using C-PhyL, the same set of 75 networks introduced in Section 3.2.1 is used. The network structures are either learned by using $f(Q) = Q^{\mu}$ or $f(Q) = \frac{(1+\alpha)Q^{\mu}}{1+\alpha Q^{\mu}}$ with $\mu = 1.0$. Parameter α is varied to be either 10, 15, 20, 22, 25 or 30. Hence, each of the 75 networks is tested with seven different configurations resulting in 525 experiments. Next, it is counted how often a method reached best learning results for a network with respect to the learned Bayesian score. A summary of the counts is given in Table 3.2.

The first row *All* shows how often the seven tested configurations (columns) could learn the network with the highest Bayesian score. Note that the sum of row one is not 75 but 159 as for some networks more than one configurations shared the best result. The

	Eq. 2.11	Eq. 2.12					
		$\alpha = 10$	$\alpha = 15$	$\alpha = 20$	$\alpha = 22$	$\alpha = 25$	$\alpha = 30$
All	23	26	20	21	20	19	**30**
n5	4	10	9	9	9	10	**12**
n10	3	5	4	5	6	5	**8**
n15	2	3	5	**6**	5	3	**6**
n20	**7**	3	1	1	0	1	2
n25	**7**	5	1	0	0	0	2
p1	**10**	5	7	3	3	4	4
p2	6	**9**	3	4	4	3	4
p3	3	5	2	2	2	2	**8**
p4	3	2	3	4	4	4	**7**
p5	1	5	5	**8**	7	6	7
c2	**10**	6	5	6	5	4	9
c3	5	11	7	6	7	9	**12**
c4	8	**9**	8	**9**	8	6	**9**

Table 3.2: Comparison of Equation 2.11 and 2.12 as function $f(Q)$ using $\mu = 1.0$ and different values for α. Line *All* counts the number of times where a method delivered best learning results regarding the Bayesian score out of 75 benchmark networks. Further, counts are shown for networks fragmented by their number of nodes, number of maximal parents and cardinality.

following lines show the counts for specific networks. Row 2 for example, indicates the sum of all networks with 5 nodes. Thus, the sum over all `n[N]` rows is equal to the value in row *All*. Rows with `p[P]` fragment networks with respect to the number of parents and rows named with `c[C]` show results for different cardinalities respectively.

It can be seen from Table 3.2 that there is no absolute benefit for one of the two $f(Q)$ methods, although Equation 2.12 with $\alpha = 30$ performs best on average over all cases. But, only in 30 out of 75 networks, Equation 2.12 with $\alpha = 30$ could learn the highest scored network structure. Further, by investigating the datasets fragmented by their number of nodes, it can be observed that Equation 2.12 with $\alpha = 30$ performs best especially for networks with a small number of nodes whereas Equation 2.11 performs better for network including more nodes. In contrast, there is no clear indication that there is a benefit for a configuration regarding the number of parents or cardinality.

3.2.3 The exponent μ of Equation 2.11

As already mentioned, using $f(Q) = Q^{\mu}$ with $\mu = 1.0$ stabilizes the learning result and ensures that the shortest path is selected accepting higher execution time. Preliminary results with $\mu > 1.0$ showed that higher scored networks can be learned in some cases. But learning performance is very unstable in these situations, meaning that the quality varies unacceptable. Thus, an increase in execution time is henceforth accepted to ensure stable and reproducible learning results by using $\mu = 1.0$.

3.2.4 Influence of food amount I_0

In order to examine if the amount of food given to the *Physarum Solver* has influence on learning result, especially with respect to the number of learned edges, the structures of the previously described 75 artificial benchmark datasets are learned with six different values for parameter I_0: 1.0, 1.5, 2.0, 5.0, 7.5 and 10.0. One could assume that with increasing amount of food, the number of surviving edges increases, too. In preliminary experiments, especially for crowded networks i.e. where the number of allowed parents and therefore the ratio between the number of edges and the number of nodes is high, C-PhyL suffered from learning too few edges. This problem can have two reasons. First, not enough connections are surviving the *Physarum Solver* or second, not enough connection are passing the restrictions to be added to the final Bayesian network from the rank list.

From 75 benchmark networks, the learning result was constant for 67 networks and varied for only 8 networks regarding the Bayesian score where for 7 networks, one additional arc was learned with increasing I_0. On the other hand, for one network, the number of arcs was increased by 1 for $I_0 = 1.0$. Further, all 8 networks have 20 or 25 nodes, meaning that larger network are more likely to have additional edges when increasing I_0. Nevertheless, the increase in learning performance is not significant. It remains to verify if the *Physarum Solver* or the restrictions of C-PhyL are responsible for the constant learning results. Therefore, the list of connection ranks for network `A_n25_p4_a57_c4` is compared for $I_0 = 1.0$ and $I_0 = 7.5$, see Table 3.3. For both settings, the top 20 ranked connections are in almost the same order varying only at position 8 and 9 where 15 of these connections have the same rank. Further, except from the highest ranked connection $N_6 \leftrightarrow N_8$, the difference in rank between $I_0 = 1.0$ and $I_0 = 7.5$ is only 1.

These observations indicate that variation of I_0 does not have strong impact on the number of learned arcs already at *Physarum Solver* level, except from minor fluctuations. The result is not unexpected, as equations used with the *Physarum Solver* are dimensionless and thus rather independent of food amount.

3.2.5 Investigation of length prior l

The length of the *Physarum Solver* connections are set by Equation 3.1, where a prior length l is added to avoid connections of zero length if $\phi_{c_{norm}} = 1$. Parameter l is added to a value in the range of 0 and 1 and is thus set to a value of either 0.1, 0.05 or 0.01. It is assumed that a value of $l > 0.1$ adds too much bias to the length. On the other hand, choosing a very small value of l could lead to remarkably short connections which are not representative for the correlation any more.

The C-PhyL algorithm has been applied to the same 75 networks that have been used in previous experiments with $l = 0.1, l = 0.05$ and $l = 0.01$ and compared with respect to the Bayesian score. For all 75 networks, there was no difference in Bayesian score for using C-PhyL with $l = 0.1, l = 0.05$ or $l = 0.01$. These results clearly indicate, that the length prior l can be set to any of these values without influencing quality of learned network structures.

Position	$I_0 = 1.0$		$I_0 = 7.5$	
	Rank	**Connection**	**Rank**	**Connection**
1	97	$N_6 \leftrightarrow N_8$	100	$N_6 \leftrightarrow N_8$
2	43	$N_8 \leftrightarrow N_{10}$	43	$N_8 \leftrightarrow N_{10}$
3	38	$N_1 \leftrightarrow N_5$	39	$N_1 \leftrightarrow N_5$
4	36	$N_8 \leftrightarrow N_9$	36	$N_8 \leftrightarrow N_9$
5	32	$N_{23} \leftrightarrow N_{22}$	32	$N_{23} \leftrightarrow N_{22}$
6	29	$N_6 \leftrightarrow N_7$	29	$N_6 \leftrightarrow N_7$
7	23	$N_6 \leftrightarrow N_3$	24	$N_6 \leftrightarrow N_3$
8	20	$\boldsymbol{N_6 \leftrightarrow N_2}$	20	$\boldsymbol{N_3 \leftrightarrow N_2}$
9	19	$\boldsymbol{N_3 \leftrightarrow N_2}$	20	$\boldsymbol{N_6 \leftrightarrow N_2}$
10	19	$N_9 \leftrightarrow N_{19}$	20	$N_9 \leftrightarrow N_{19}$
11	18	$N_6 \leftrightarrow N_{11}$	18	$N_6 \leftrightarrow N_{11}$
12	18	$N_6 \leftrightarrow N_{20}$	18	$N_6 \leftrightarrow N_{20}$
13	18	$N_8 \leftrightarrow N_{14}$	18	$N_8 \leftrightarrow N_{14}$
14	18	$N_9 \leftrightarrow N_{16}$	18	$N_9 \leftrightarrow N_{16}$
15	17	$N_4 \leftrightarrow N_7$	17	$N_4 \leftrightarrow N_7$
16	16	$N_3 \leftrightarrow N_{23}$	16	$N_3 \leftrightarrow N_{23}$
17	12	$N_{10} \leftrightarrow N_1$	12	$N_{10} \leftrightarrow N_1$
18	12	$N_1 \leftrightarrow N_{12}$	12	$N_1 \leftrightarrow N_{12}$
19	10	$N_6 \leftrightarrow N_{21}$	10	$N_6 \leftrightarrow N_{21}$
20	10	$N_{10} \leftrightarrow N_{13}$	10	$N_{10} \leftrightarrow N_{13}$

Table 3.3: Connection ranks of network `A_n25_p4_a57_c4` are compared for learning with $I_0 = 1.0$ and $I_0 = 7.5$ where the 20 top ranked connections are shown. Note that for both cases, the order of the connections is the same except from position 8 and 9. Further, ranks vary only in five of the 20 top ranked connections where except from the highest ranked connection, the rank difference is only 1.

3.2.6 Influence of exponent γ

Lastly, influence of exponent γ in Equation 3.1 on learning performance is investigated. Exponent γ stretches the length values and is thus considered to have high impact on structure learning quality as the length relations of the Physarum-Maze are changed. Consider for example three connections M_{ij}, M_{jk} and M_{ik} where $L_{ij} = L_{jk} = 0.5$ and $L_{ik} = 0.75$ (before applying γ) and suppose that the *Physarum Solver* tries to find the shortest connection between node X_i and X_k. With $\gamma = 1$, the shortest path is the direct connection M_{ik} as the length of the path over node X_j is 1.0. Changing the value of γ to 2, path $X_i \leftrightarrow X_j \leftrightarrow X_k$ has length $0.5^2 + 0.5^2 = 0.5$ whereas path $X_i \leftrightarrow X_k$ has length $0.75^2 = 0.5635$. Thus, increasing γ enables the choice of longer paths through the Physarum-Maze.

Preliminary results using the 75 test networks showed that γ has influence on learning result. Therefore, nine other randomly generated sets of 75 similar configured networks have been tested. This leads to a benchmark database of 750 networks where each configuration is present 10 times for example (A_n5_p1_a4_c2, B_n5_p1_a4_c2,. . . ,J_n5_p1_a4_c2).

These 750 artificial benchmark networks have been used to analyse the impact of γ

on learning performance by setting γ equal to 1.0, 2.0, 3.0, 4.0 and 5.0. The number of times where C-PhyL learned the highest scoring network is counted for each of the possible values of γ. The result is shown in Table 3.4.

	$\gamma = 1.0$	$\gamma = 2.0$	$\gamma = 3.0$	$\gamma = 4.0$	$\gamma = 5.0$
All	245	**364**	356	323	302
n5	78	114	123	131	**133**
n10	46	57	68	**69**	**69**
n15	49	**67**	54	51	43
n20	37	**65**	52	35	31
n25	35	**61**	59	37	26
p1	49	90	103	112	**117**
p2	46	**72**	61	57	55
p3	44	**70**	62	53	46
p4	54	63	**67**	51	37
p5	52	**69**	63	50	47
c2	69	104	**117**	104	99
c3	80	**126**	118	96	92
c4	96	**134**	121	123	111

Table 3.4: Different values of γ are compared regarding the Bayesian score. Counts how often each configuration has learned the highest scoring network compared to the other configurations are given. Row *All* shows the total count for all 750 benchmark networks while other rows provide fragmented counts for networks with the same number of nodes, parents and cardinality.

Results presented in Table 3.4 indicate a benefit for $\gamma = 2.0$ which clearly outperforms the other values except from $\gamma = 3.0$ which performed almost as good as $\gamma = 2.0$. Although, it can be seen that $\gamma = 5.0$ performs better for smaller networks having 5 or 10 nodes and for networks restricted to only one parent per node. The average score difference for networks where the score was not equal between the five values of γ is only 0.7% where the maximal score improvement was 9.8%. These results show that γ has influence on learning performance but for most networks, the score improvement is not compelling.

3.3 Experiments and analysis with benchmark networks

Examination of different parameter configurations showed that there is no setting that performs best for the majority of networks, instead parameters have to be optimized for each network individually to reach good structure learning performance. The results are in line with the "No Free Lunch" theorem [209], stating that there is no method that performs best for all datasets in an area under study. Nevertheless, experiments also showed, that learning results do not vary extremely with changing parameters.

The quality of different structure learning algorithms is evaluated by comparing the

learned structure to the original structure (where the data has been sampled from) using the following metrics:

- **A** - total number of learned arcs
- **T** - number of true/correct arcs
- **R** - number of reversed arcs
- **M** - number of missing arcs
- **E** - number of extra arcs
- **Bayes** - Bayesian score
- **t** - execution time in seconds

The total number of arcs **A** is retrieved by counting arcs in the learned network structure. The number of true arcs **T** indicates how many arcs of the learned network structure are also present in the original network and are directed equally. Arcs that are shared in the true and the learned structure but differ in their direction are called to be reversed arcs **R**. Arcs form the true network not present in the learned network are called missing arcs **M** whereas arcs from the learned network that are missing in the original network are called extra arcs **E**. The score metric **Bayes** is measured using Weka's built in Bayesian score function for Bayesian networks. The execution time is retrieved by calculating the difference of Java's `System.nanoTime()` method before and after running the C-PhyL algorithm.

3.3.1 Artificial benchmark networks

To validate the quality of the learned structures using C-PhyL, a comparison to three other state of the art score-based structure learning methods is performed. The LAGD algorithm implemented in Weka is used with 5 good operations and a look ahead step size of 2 according to performances in preliminary experiments. Further, Tabu-Search also implemented in Weka is used with tabu list size of 5 performing 500 runs. The third state of the art learning algorithm used is Simulated Annealing where initial temperature t_0 is set to 10 and is reduced in each iteration by 0.999. Simulated Annealing is performed using 10,000 runs. All of these three algorithms have been learned with using arc reversal and not initializing the network as a naive Bayes network which would assume that all nodes would be initially connected to a predefined node. Further, as benchmark networks do not contain nodes with more than five parents, the maximum in-degree was set to 5 for all learning algorithms.

Learning algorithms are compared by using a selection of 18 artificial benchmark networks including different numbers of nodes, parents and cardinalities to get sparse, normal and crowded networks (see Table 3.5 and 3.6). For all networks, datasets containing 1000 instances are sampled and used for structure learning.

C-PhyL is used to learn the structure of the benchmark networks with using the optimal parameters for each dataset according to the results presented in the Section 3.2. Table 3.5 and 3.6 show learning performance of the four structure learning methods,

where the highest score for each network is printed in bold face and Simulated Annealing is abbreviated by SA.

Dataset	Learner	Bayes	A	T	R	M	E	t
A_n5_p1_a4_c2	C-PhyL	-3357.09	3	3	0	1	0	0.638
	LAGD	**-3357.04**	3	1	2	1	0	0.045
	Tabu	-3357.08	3	1	2	1	0	0.271
	SA	-3357.08	3	2	1	1	0	0.479
A_n5_p3_a7_c3	C-PhyL	-4657.20	6	3	3	1	0	0.647
	LAGD	**-4575.17**	7	7	0	0	0	0.243
	Tabu	-4613.24	7	3	4	0	0	0.276
	SA	-4614.23	7	2	5	0	0	0.390
A_n5_p5_a7_c4	C-PhyL	-5718.30	6	3	3	1	0	2.047
	LAGD	**-5426.88**	7	7	0	0	0	0.101
	Tabu	-5454.76	7	6	1	0	0	0.288
	SA	-5477.31	8	4	3	0	1	0.598
A_n10_p1_a9_c2	C-PhyL	-4167.78	6	5	1	3	0	1.043
	LAGD	**-4159.06**	9	4	2	3	3	0.074
	Tabu	-4161.09	9	3	2	4	4	0.330
	SA	-4161.61	11	5	2	2	4	0.310
A_n10_p3_a17_c3	C-PhyL	-7641.00	14	9	5	3	0	1.597
	LAGD	**-7318.79**	17	17	0	0	0	0.110
	Tabu	-7364.46	20	13	4	0	3	0.184
	SA	-7439.09	22	14	3	0	5	2.697
A_n10_p5_a22_c4	C-PhyL	-12524.83	11	9	2	11	0	3.180
	LAGD	**-12087.45**	14	11	3	8	0	0.926
	Tabu	-12154.45	13	7	5	10	1	0.434
	SA	-12247.75	16	6	8	8	2	1.233
A_n15_p1_a14_c2	C-PhyL	-8121.97	11	6	4	4	1	2.609
	LAGD	**-8022.11**	15	9	4	1	2	0.115
	Tabu	-8026.04	16	8	5	1	3	0.221
	SA	-8039.21	17	8	5	1	4	0.617
A_n15_p3_a27_c3	C-PhyL	-13030.65	21	13	5	9	3	5.403
	LAGD	**-12391.77**	27	25	2	0	0	0.217
	Tabu	-12414.65	29	24	3	0	2	0.337
	SA	-12592.11	32	18	7	2	7	1.675
A_n15_p5_a37_c4	C-PhyL	-19229.76	13	9	2	26	2	3.933
	LAGD	**-18927.73**	12	12	0	25	0	1.775
	Tabu	-19081.44	13	9	3	25	1	0.557
	SA	-19071.30	11	9	2	26	0	0.653

Table 3.5: Comparison of four different structure learning algorithms for networks with 5, 10 or 15 nodes. The highest score for each network is written in bold face.

Results clearly show that LAGD outperforms the other methods for nearly all networks, except from two networks where Tabu Search learned the highest score. Further, C-PhyL learned the smallest score for all networks except from network A_n25_p1_a24_c2

Dataset	Learner	Bayes	A	T	R	M	E	t
A_n20_p1_a19_c2	C-PhyL	-10016.26	15	5	9	5	1	8.524
	LAGD	**-9814.13**	24	18	0	1	6	0.251
	Tabu	-9819.85	24	11	6	2	7	0.546
	SA	-9867.50	34	5	11	3	18	0.486
A_n20_p3_a37_c3	C-PhyL	-18986.04	28	25	2	10	1	10.804
	LAGD	**-18012.18**	37	36	1	0	0	0.458
	Tabu	-18212.56	37	33	3	1	1	0.376
	SA	-18924.24	38	13	13	11	12	2.451
A_n20_p5_a52_c4	C-PhyL	-25979.45	16	11	5	36	0	8.645
	LAGD	**-25553.45**	19	17	2	33	0	0.871
	Tabu	-25705.61	22	11	7	34	4	0.396
	SA	-25705.49	21	10	7	35	4	1.273
A_n25_p1_a24_c2	C-PhyL	-12630.48	22	10	10	4	2	18.553
	LAGD	**-12595.36**	25	19	1	4	5	0.440
	Tabu	-12600.83	27	15	6	3	6	0.674
	SA	-12669.60	38	9	11	4	18	1.090
A_n25_p3_a47_c3	C-PhyL	-22631.85	42	37	4	6	1	25.544
	LAGD	**-21797.08**	47	44	3	0	0	0.908
	Tabu	-22440.99	52	31	11	5	10	1.016
	SA	-22354.81	57	28	16	3	13	11.883
A_n25_p5_a67_c4	C-PhyL	-33133.58	10	7	2	58	1	19.592
	LAGD	-32328.02	23	15	5	47	3	0.723
	Tabu	**-32101.45**	27	25	1	41	1	0.848
	SA	-32494.87	24	16	4	47	4	1.090
A_n50_p1_a49_c2	C-PhyL	-25631.64	39	17	17	15	5	249.300
	LAGD	-25059.52	72	31	10	8	31	2.948
	Tabu	**-25058.40**	70	29	10	10	31	5.034
	SA	-25167.02	107	23	15	11	69	511.160
A_n50_p3_a97_c3	C-PhyL	-45932.40	74	52	22	23	0	273.611
	LAGD	**-43520.83**	95	83	6	8	6	6.795
	Tabu	-43773.03	98	76	15	6	7	5.046
	SA	-44917.91	105	54	26	17	25	0.823
A_n50_p5_a142_c4	C-PhyL	-64613.60	30	25	5	112	0	326.879
	LAGD	**-63318.81**	44	41	3	98	0	2.472
	Tabu	-63605.19	46	32	8	102	6	4.730
	SA	-63939.13	41	28	8	106	5	1.004

Table 3.6: Comparison of four different structure learning algorithms for networks with 20, 25 or 50 nodes. The highest score for each network is written in bold face.

where Simulated Annealing performed worse. But, the average difference between the highest score for each network and the score learned by C-PhyL is only 2.72% which is surprising as C-PhyL does not consider the score in its learning procedure directly. The score is only used for determining if a connection from the ranked list is added to the final Bayesian network or not. In addition, it is noticed that C-PhyL does not learn

many extra arcs compared to the other methods indicating high specificity of C-PhyL. It can also be seen, that C-PhyL does learn less connections for bigger networks which is one reason for the high specificity. Thus, it can be concluded, that C-PhyL does not learn enough arcs, but the arcs learned can be assumed to be correct. Investigating the number of true arcs and number of reversed arcs, it can be suggested that the usage of a variable ordering to determine the directions of the arcs does not work very well. For many benchmark networks, C-PhyL was able to find the correct arcs, but suffered from finding the valid direction of the arcs. Obviously, C-PhyL is much slower compared to the other methods as no efforts have been given so far to optimize C-PhyL for speed.

To summarize results of this section, it was shown that C-PhyL can be used to learn the structure of Bayesian networks with nearly comparable results to state of the art methods. Thus, the novel concept of using the *Physarum Solver* for learning network structures of Bayesian networks has been shown to be usable.

3.3.2 Real benchmark networks

Artificially generated networks have been used to study the behaviour of C-PhyL under different parameter settings and to compare learning quality to other state of the art structure learning algorithms. In this section, C-PhyL is tested on real world benchmark networks introduced in Section 2.5 and compared to Tabu Search and LAGD. Please note that Simulated Annealing is not applied for real world networks as preliminary results showed that Weka's implementation throws exceptions due to array overflows for some networks. Further, experiments with artificial networks showed that Simulated Annealing does not perform as good as Tabu Search and LAGD. Again, datasets of 1000 instances have been sampled from the benchmark networks. The datasets are passed as input to C-PhyL, Tabu Search and LAGD and the learned network structures are compared using the same metrics as have been used with artificial benchmark networks. For the three smaller networks *Cancer*, *Earthquake* and *Asia*, parameter γ was set to 5.0 and Equation 2.12 was used with $\alpha = 30.0$ according to results of Section 3.2. Hence, larger networks *Insurance*, *Alarm*, *Barley* and *Hailfinder* have been learned using C-PhyL with $\gamma = 2.0$ and Equation 2.11. Performance of C-PhyL could be improved by fitting the parameters to each network individually, but this is not done in order to avoid overfitting. Results are shown in Table 3.7.

For the *Cancer* network, all three algorithms learned the same network structure. The fact that two of the four arcs are reversed in all three cases is explained by a bias in the CPT of node Cancer to value *False* no matter what values its parent has. In general, result for real networks are in line with results of experiments done with artificial networks. C-PhyL performs comparable for sparse networks where the relation between the number of nodes and arcs is small. On the other hand, if there are many arcs per node resulting in a more crowded network, C-PhyL performs worse and suffers from the fact that less arcs are learned. But again, the quality of the arcs is good, meaning that C-PhyL does not learn many wrong i.e. extra arcs. Hence, for seven of the eight networks, C-PhyL learned the least number of extra arcs. It can also be seen, that the relation between correctly directed and reversed arcs is very bad for results determined by C-PhyL which is again pursuant to results with artificial networks and indicates that the usage of a variable ordering is not appropriate to determine connection directions. Investigating execution

Dataset	Nodes	Arcs	Learner	Bayes	A	T	R	M	E	t
			C-PhyL	-2235.90	4	2	2	0	0	2.916
Cancer	5	4	LAGD	-2235.90	4	2	2	0	0	0.076
			Tabu	-2235.90	4	2	2	0	0	0.252
			C-PhyL	-534.73	3	2	1	1	0	0.041
Earthquake	5	4	LAGD	**-521.43**	4	4	0	0	0	0.015
			Tabu	**-521.43**	4	4	0	0	0	0.129
			C-PhyL	-2388.99	7	1	4	3	2	0.437
Asia	8	8	LAGD	**-2318.45**	8	4	3	1	1	0.030
			Tabu	**-2318.45**	8	4	3	1	1	0.099
			C-PhyL	-18651.63	35	16	10	26	9	19.007
Insurance	27	52	LAGD	-15772.38	55	25	13	14	17	0.912
			Tabu	**-15740.36**	58	26	14	12	18	1.087
			C-PhyL	-13115.39	45	11	22	13	12	106.12
Alarm	37	46	LAGD	**-11247.43**	54	38	4	4	12	1.003
			Tabu	-11299.76	55	34	8	4	13	1.485
			C-PhyL	-71716.49	33	14	15	55	4	245.68
Barley	48	84	LAGD	**-61790.39**	81	23	28	33	30	16.488
			Tabu	-63021.69	80	15	33	36	32	10.386
			C-PhyL	-54171.01	55	21	20	25	14	828.42
Hailfinder	56	66	LAGD	**-51322.70**	75	35	14	17	26	4.469
			Tabu	-51374.87	76	32	17	17	27	4.952

Table 3.7: C-PhyL is compared to Tabu Search and LAGD using seven real benchmark networks. First column shows the name of the networks. Column *Nodes* shows how many nodes the network has and column *Arcs* indicated the number of arcs in the original network. Remaining columns show the metrics by which learning methods are evaluated.

times, C-PhyL clearly needs more time than other methods as the *Physarum Solver* has to solve the linear equation system many times, which grows fast if the number of nodes is increased.

The reason why C-PhyL does not learn enough connections for larger networks is further investigated by examining the list of ranked connections for *Insurance* network as an example case. Figure 3.4 shows a histogram of the top ranked connections in descending order determined by the C-PhyL algorithm. Green histogram bars indicate connections that are also present in the original true *Insurance* network where the direction is not considered so far and red histogram bars show wrong connection that aren't present in the true network.

Figure 3.4 clearly shows, that C-PhyL is able to find the most important connections but is unable to indicate connections that are not clearly represented by the pair-wise correlation of the nodes. The true network includes 52 connections and the position where ranks fall below average rank $\theta = 2.95$ is 51 indicating that the border θ is adequate. But, roughly only the upper two third of ranked connections where rank is higher θ do include correct connections in majority. Thus, C-PhyL does not find enough connections to perform comparable to other state of the art learning algorithms. Please note, that the nodes of the first seven wrong connections at position 3, 8, 14, 23, 24, 27 and 31 are

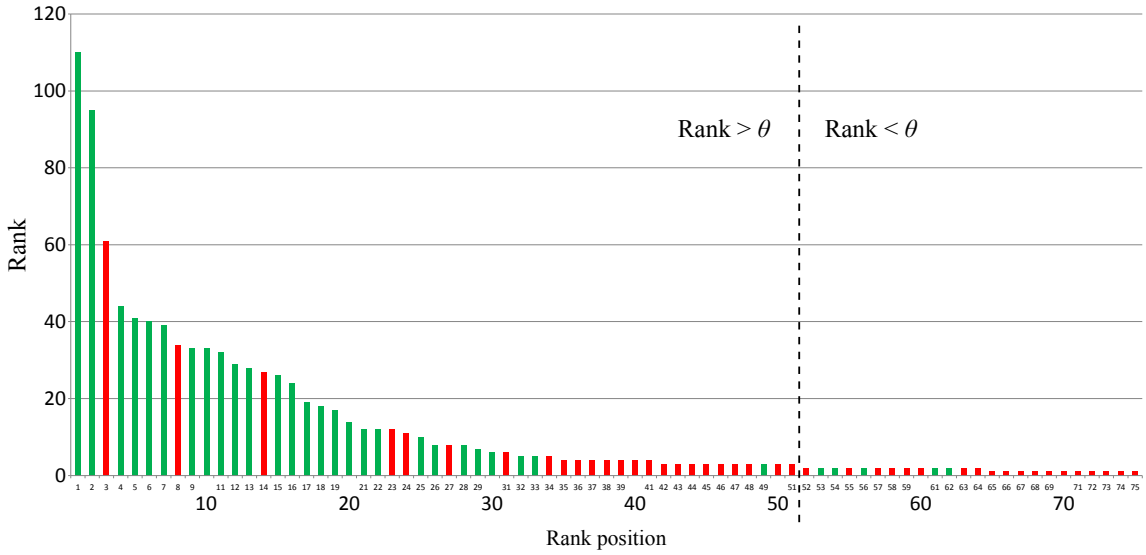
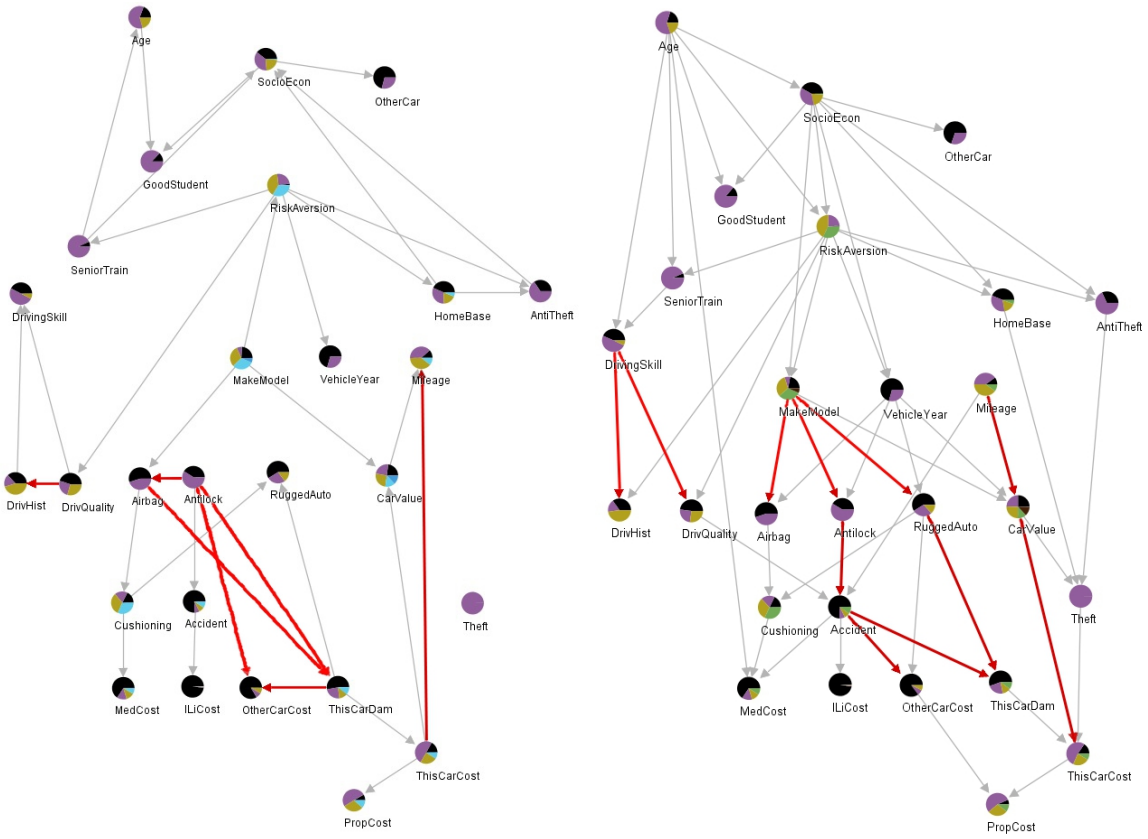


Figure 3.4: Histogram of the 75 top ranked connections for the *Insurance* network. The x-axis shows the position in the connection rank list where the y-axis shows the rank value. Connections that can also be found in the original *Insurance* network are painted green, wrong connections are marked red. Bars left to the dashed line have rank values higher than the average rank value $\theta = 2.95$ of Algorithm 3 whereas connections right to the dashed line have rank values less tan θ.

connected in all seven cases via a diverging or serial path explaining their high correlation values, see Figure 3.5. Figure 3.5a shows the network structure learned by C-PhyL for the insurance network and Figure 3.5b shows the original network. Edges marked red in Figure 3.5a illustrate the seven high ranked incorrect edges also marked red in histogram of Figure 3.4. Red edges of Figure 3.5b illustrate the diverging and serial paths that lead to the wrong connections. For example edge `DrivQuality` $\rightarrow$ `DrivHist` in 3.5a can be explained by a diverging connection over `DrivingSkills` in the original network. Connection between `Antilock` and `ThisCarDam` is the only one that is represented in the correct structure by both, a diverging path over `MakeModel` and a serial path over `RuggedAuto`. Further, it can be clearly seen from the comparison of the learned and the true network structure, that C-PhyL tends to learn a more sparse network.

Results presented in Tables 3.5, 3.6 and 3.7 showed that the execution time of C-PhyL grows much faster for networks with more nodes than execution time of Tabu Search and LAGD. With increasing number of nodes, the linear equation system of the *Physarum Solver* grows quadratically which explains the longer execution time. On the other hand, when the number of instances is increased, C-PhyL is supposed to terminate in nearly the same time as the correlation coefficients have to be calculated only once at initialization. Scoring based methods instead have to touch the dataset at each search step while C-PhyL touches the dataset only once. Thus, the C-PhyL is compared to LAGD and Tabu Search in order to study the execution time with increasing number of instances by using the *Asia* network. Execution time is plotted logarithmically as a function of the number of instances for *Asia* in Figure 3.6.

Theoretically, the advantage in execution time for C-PhyL with high numbers of



(a) (b)

Figure 3.5: Comparison of the insurance network structure learned by C-PhyL (a) to the original structure (b). Red edges in (a) indicate the seven top ranked wrong connections at positions 3, 8, 14, 23, 24, 27 and 31 also printed red in Figure 3.4. Red edges in (b) show that all these connections are represented as diverging or serial paths in the correct network. Note that connection between `Antilock` and `ThisCarDam` is represented in the correct structure by both, a diverging path over `MakeModel` and a serial path over `RuggedAuto`. Please note that graph format description is provided in Appendix A

instances is bigger than shown in Figure 3.6 as the implementation of C-PhyL is not optimized for speed and memory usage. Nevertheless, even the naive implementation of C-PhyL is faster than LAGD and Tabu Search if more than 1,000,000 instances are used for structure learning.

3.4 Conclusion and future work

A novel structure learning algorithm for Bayesian networks has been introduced which uses a totally new approach on estimating independences from data. A bio-inspired mathematical model called *Physarum Solver* is used to find paths over pairwise correlations that explain a correlation between two variables best, assuming that the direct correlation between the two variables is unknown. Using this strategy, the algorithm tries to find higher order correlations. The novel method called C-PhyL was described and algorithm

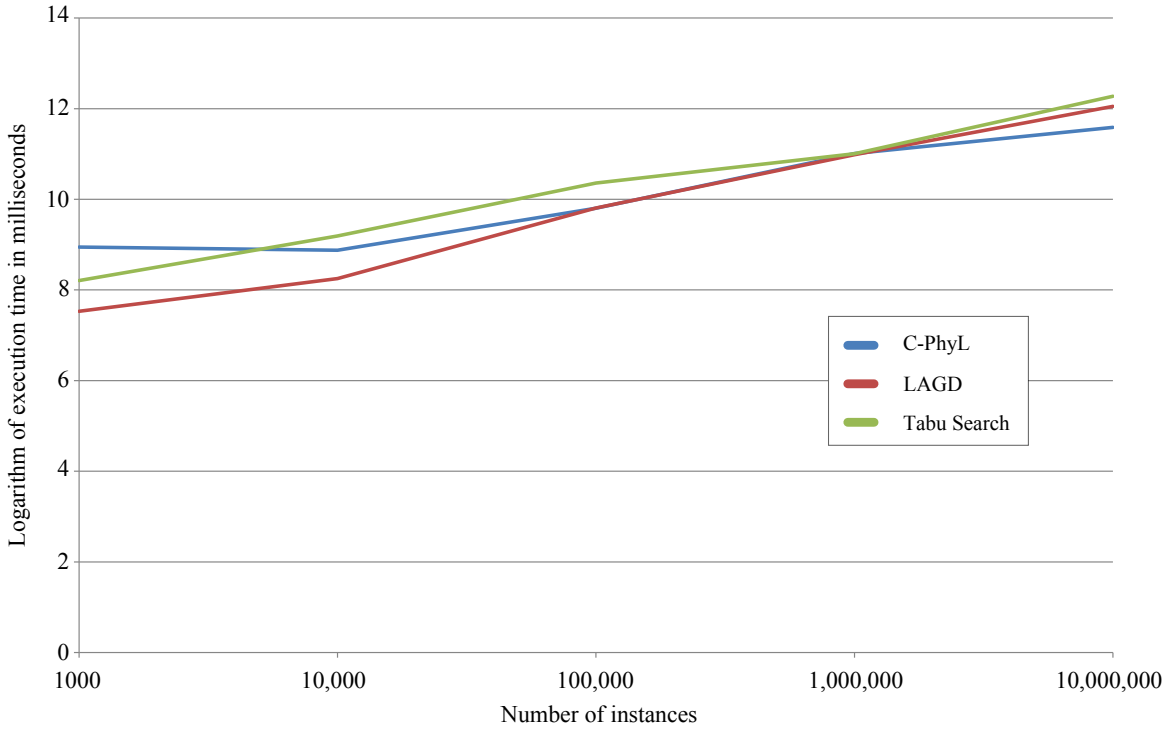


Figure 3.6: Logarithm of execution time in milliseconds as a function of the number of instances for benchmark network *Asia* using C-PhyL, LAGD and Tabu Search.

parameters have been studied. It was observed that there are two major parameters influencing the learning quality of C-PhyL, γ and function $f(Q)$ which is used within the *Physarum Solver*. Next, C-PhyL has been compared to three state of the art structure learning methods on a set of artificially generated benchmark networks with different network characteristics. It turned out, that the novel algorithm performs adequate for some networks but could not reach the quality of the state of the art learning methods regarding the Bayesian score and the quality of learned arcs. But, results showed that C-PhyL learns less extra arcs compared to other methods indicating a higher specificity of arcs. It was further observed, that for most correctly identified arcs, the correct direction could not be estimated by using an ordering based approach. To determine the ordering, the simple score based approach was used. In future work, more advanced methods might be tested to find a better way of determining a valid ordering. Another method to direct arcs within the C-PhyL algorithm is to use the PC-Algorithm introduced by Spirtes *et. al.* [188] which tries to find directions of a network skeleton by performing independence tests. Another, and probably the most nearby technique (as fluxes in the real slime mold do have directions) is to update the *Physarum Solver* algorithm by considering Kirchhoff's second law saying that the directed sum of potential differences around any closed network is zero. This law has to be valid also for the flow of flux in the *Physarum Solver* where the flow in a closed loop of tubes has to flow in the same direction. By updating the transportation equation accordingly to consider this law, the *Physarum Solver* itself would be able to determine the directions in which the sol flows. The direction of the final edges in the Bayesian network could therefore be derived from the *Physarum Solver*. A problem that has to be solved considering this approach is that Kirchhoff's

law needs to be applied to closed circles while Bayesian networks are forbidden to include circles by definition. In the initial state, where the Physarum-Maze is fully connected, each connection is automatically part of several loops. But, with getting connections cut out of the Physarum-Maze in each *Physarum Solver* iterations, the Physarum-Maze transforms into a graph with less and less loops until the final state includes only a path. Thus, defining Kirchoff's laws becomes more difficult with proceeding iterations and is not obvious once most loops have been cut out. For that reason, a modified *Physarum Solver* which is able to also learn directions by using Kirchoff's second law is referred to as future work. Another problem of the C-PhyL algorithm can be observed in Figure 3.5a, where the learned network structure of the insurance benchmark network is shown. It can be noticed, that node `Theft` is not connected to any other node although Algorithm 3 tries to avoid unconnected nodes. The reason for that is that node `Theft` never occurs as a child node. For future experiments, Algorithm 3 needs to be updated so that connections are also added to the final Bayesian network if a connections parent node is yet unconnected.

Execution time analysis showed a disadvantage for C-PhyL with increasing number of nodes but a benefit if the number of instances grows dramatically. C-PhyL has to determine the pairwise correlation coefficients only once while score based algorithms have to touch the dataset in each optimization step. Weka's implementation does store bayesian networks and datasets within objects while the self-made implementation of the *Physarum Solver* holds Physarum-Maze and dataset informations in arrays. Thus, within C-PhyL, several copies from Weka objects to arrays have to be made which is tremendously time and memory intensive for larger datasets. Further, the linear equation system that has to be solved for each iteration within the *Physarum Solver* to calculate pressure values is solved by using an implementation of a singular value decomposition method while Tero *et. al.* reported that they have used Incomplete Cholesky Conjugate Gradient which might be faster. Another approach to speed up C-PhyL is to use shuttle streaming as introduced by Siriwardana *et. al.* [182] who showed that the *Physarum Solver* could be much faster when using their proposed modifications. Once unprofitable implementation issues are resolved, the C-PhyL algorithm has high potential to be used for datasets including many instances as for example streaming data.

Another benefit of C-PhyL is that the algorithm does not require a maximum in-degree. Score based structure learning methods are restricted to maximum number of parents per nodes as score calculation becomes infeasible with a growing set of parents. Further, C-PhyL can be easily adopted to be used with continuous variables as Cramér's V correlation coefficient can be replaced for example by the common Pearson correlation coefficient. Of course, the representation of conditional probability distributions has to be updated to work for continuous data, too.

CHAPTER 4

Score optimizing Physarum Learner (SO-PhyL)

The previous chapter demonstrated that the *Physarum Solver* introduced by Tero *et. al.* can be used to learn the structure of a Bayesian network from data accepting some drawbacks. Due to its correlation based approach, C-PhyL poorly determines directions of connections and learns more sparse networks than score optimizing competitors and is thus unable to learn networks with competitive score. Resolving these problems motivated the development of another structure learning algorithm for Bayesian networks based on *Physarum Solver* that considers score already within learning iterations. The score optimizing Physarum Learner (SO-PhyL) uses the *Physarum Solver* for learning the structure of a Bayesian network from data in a different way than C-PhyL. Instead of searching paths in a Physarum-Maze where connection lengths are determined by pairwise correlations between connected nodes, SO-PhyL uses the *Physarum Solver* to search the space of possible network structures in order to optimize a score. Thus, SO-PhyL belongs to the group of search and score based algorithms including also LAGD and Tabu Search.

4.1 The SO-PhyL algorithm

The basic idea of SO-PhyL is that the Physarum-Maze can be transformed to a Bayesian network in each iteration of the *Physarum Solver* by applying a filter that is cutting out connections with a conductivity value below a predefined threshold. The influence of each connection to the score of the Bayesian network is evaluated in each iteration and feedback is given to the connections by increasing conductivity of a score improving connection and decreasing conductivity otherwise. Obviously, the common *Physarum Solver* is no longer applicable as there are no longer only two food sources, but each node is considered as a food source to avoid nodes being cut out from the Bayesian network. Hence, SO-PhyL uses the *Physarum Solver* updated for multiple food sources further referred to as *MFS-Physarum Solver* that has been introduced by Tero *et. al.* [198]. Please remind that for multiple food sources, a new function $f(Q)$ (see Equation 2.14)

has been supposed by Tero *et. al.* and is thus used with the *MFS-Physarum Solver*.

4.1.1 Initialize Physarum-Maze

As a first step, a Physarum-Maze has to be initialized based on the dataset for which the structure is to be learned. Therefore, for each parameter in the dataset, a node is inserted into the Physarum-Maze and is connected to any other node so that the final Physarum-Maze is fully connected as has been done in the first step of C-PhyL, see Figure 3.1. Again, conductivity values of connections are set randomly in the range of D_{min} and D_{max}. In the original model of the *Physarum Solver*, the change in conductivity D_{ij} is calculated using a function of the flux Q_{ij} which is dependent on both, the actual conductivity and the length of the connection. Setting a constant value of length L_{ij} for all connections implies that all network dynamics are dependent only on the conductivities. Hence, it is easier for SO-PhyL to influence network adaptation by giving positive or negative feedback to a single parameter. For that reason, all connections M_{ij} are given the same constant value of $L_{ij} = 1.0$. Please note that the value of L_{ij} can be set to any positive number without influencing results. The procedure of initializing a Physarum-Maze from data to be used in SO-PhyL is also shown in Algorithm 4.

Algorithm 4 SO-PhyL: Building Physarum-Maze from dataset

Require: nodes[], connections[]
Require: dataset
  nodes ← Empty
  connections ← Empty
  for all variable X_i in dataset **do**
    Add X_i to nodes[]
    for all variable $X_j \neq X_i$ in dataset **do**
      $L_{ij} = 1.0$
      $D_{ij} = Rand(D_{min}, D_{max})$
      M_{ij} = new connection(L_{ij}, D_{ij})
      E_{ij} = set initial direction of M_{ij} randomly
      Add E_{ij} to connections[]
    end for
  end for

As SO-PhyL transforms the Physarum-Maze to a Bayesian network in each iteration, connections need to be directed at any time. Initially, connections are directed randomly where it is not necessary to check if directions are valid in the sense that the resulting Bayesian network is acyclic or the number of parents per node is exceeded. Initial directions are only given to have the initial Physarum-Maze in a valid state to be used by SO-PhyL, directions are evaluated correctly after the first iteration is completed.

4.1.2 Evaluate connections by score

Tero et al. [198] demonstrated how to run the *Physarum Solver* with multiple food sources by switching the source and sink node in each *MFS-Physarum Solver* iteration. First, a

source node is chosen randomly out of the set of nodes. Then, the sink node is chosen randomly too, but the probability of each node to be selected as sink node is growing with distance to the source node. As nodes in a Physarum-Maze used by SO-PhyL are all connected to each other directly and length is constant over all connections, distances between nodes are equal. Selecting both, source and sink node, at random would lead to an imbalance of preferred paths as the chance that a connection survives grows with the number of times it is part of the shortest path. Therefore, a list of all possible pairs of nodes is generated from which a node pair is chosen randomly in each *MFS-Physarum Solver* iteration. The two nodes are set as food source nodes and are removed from the list. It follows that the number of total *MFS-Physarum Solver* iterations has to be defined to be a multiple of the number of possible node pairs. Hence, parameter r that can be set as configuration parameter does not define the total number of *MFS-Physarum Solver* iterations, but how often the list of all possible node pairs is processed. The total number of *MFS-Physarum Solver* iterations can be calculated by

$$r_{total} = r\frac{|\boldsymbol{X}|(|\boldsymbol{X}|-1)}{2} \tag{4.1}$$

where $|\boldsymbol{X}|$ is the number of nodes in the Physarum-Maze. This procedure guarantees that each node has been used as food source the same number of times.

As in each of these iterations, two food source nodes are defined, pressures, fluxes and conductivities can be updated by using equations of the common *Physarum Solver* where adaptation equation for D_{ij} has been extended by a weighting constant w to manipulate the speed of adapting conductivity as shown in Equation 4.2 where λ is constant.

$$D_{ij_{new}} = wf(|Q_{ij}|) + (1-\lambda w)D_{ij_{old}} \tag{4.2}$$

In each *MFS-Physarum Solver* iteration, a filter is applied that transforms the Physarum-Maze into a Bayesian network by filtering out connections with conductivity below a parametrizable threshold D_τ. Parameter D_τ is adjusted linearly from its initial value D_{τ_0} to $D_{\tau_{end}}$ over r_{total} iterations, see Equation 4.3 where i is the current iteration.

$$D_\tau = D_{\tau_0} + \frac{i}{r_{total}}(D_{\tau_{end}} - D_{\tau_0}) \tag{4.3}$$

A Bayesian network $\mathcal{B}$ represented by graph $\mathcal{G}$ containing a node for each parameter in the dataset and an empty set of edges $\mathcal{E}$ is initialized. Algorithm 5 summarizes the complete SO-PhyL procedure and Figure 4.1 shows one SO-PhyL iteration.

In each *MFS-Physarum Solver* iteration, pressures, fluxes and conductivities are updated first. If the conductivity value of the connection between the two nodes selected as food sources in this iteration is less than D_τ, conductivity is increased to be $D_\tau + 0.01$. This ensures that each connection is considered to be part of the Bayesian network at least once in each r iterations. Next, connections that have been added to $\mathcal{E}$ in former iterations but do no longer exceed D_τ are removed from $\mathcal{E}$, see Algorithm 6. Further, connections with conductivity greater than D_τ are added to a list called *whitelist*. A greedy hill climbing like search algorithm is used to add connections from the whitelist to the Bayesian network one by one. The connection that leads to highest score improvement is added first to the Bayesian network and removed from the whitelist. Note that

Algorithm 5 SO-PhyL: Applying *MFS-Physarum Solver* on Physarum-Maze

Require: nodes[], connections[]
Require: r
  $\mathcal{B}_{best} \leftarrow$ null
  for Size of Ensemble **do**
    for $i < r$ **do**
      nodePairs[] = list of all possible pairs of nodes
      while nodePairs[] **not** empty **do**
        Select random pair (X_i, X_j) out of nodePairs[]
        Set X_i and X_j as food sources
        Remove pair (X_i, X_j) from nodePairs[]
        Perform one *Physarum Solver* iteration
        Reset food sources
        if $D_{ij} < D_\tau$ **then**
          $D_{ij} = D_\tau + 0.01$
        end if
        $\mathcal{B}$ = `evaluateConductivityByScore()`
        if $score_{\mathcal{B}} > score_{\mathcal{B}_{best}}$ **then**
          $\mathcal{B}_{best} = \mathcal{B}$
        end if
        $D_\tau = D_{\tau_0} + \frac{i}{r_{total}}(D_{\tau_{end}} - D_{\tau_0})$
      end while
    end for
  end for
  return $\mathcal{B}_{best}$

the search algorithm evaluates connections in both directions. The score improvement is measured by calculating the score of the connections child node with and without the additional parent defined by the edge. This score difference term is determined for either direction. Consider for example connection M_{ij} which can either be edge $E_{ij} = X_i \rightarrow X_j$ or $E_{ji} = X_j \rightarrow X_i$. For both directions, difference in score is calculated as

$$score_{Diff(E_{ij})} = score_{X_j}(X_i \in Pa(X_j)) - score_{X_j}(X_i \notin Pa(X_j)) \tag{4.4}$$

$$score_{Diff(E_{ji})} = score_{X_i}(X_j \in Pa(X_i)) - score_{X_i}(X_j \notin Pa(X_i)) \tag{4.5}$$

where the score of the child node without the parent node is subtracted from the score of the child including the parent in its parent set. A positive value of $score_{Diff}$ means that adding the connection increases the score while a negative value means that the score is decreased. The more profitable direction can easily be determined by comparing $score_{Diff(E_{ij})}$ and $score_{Diff(E_{ji})}$ where the direction with higher score improvement is chosen. This procedure is repeated as long as the whitelist is not empty or no connection from the whitelist further increases the score, see Algorithm 6.

Next, a positive feedback is given to the conductivity value of the connections that have been added to the Bayesian network and a negative feedback is given to connections that remained in the whitelist based on the relation between score with and without the

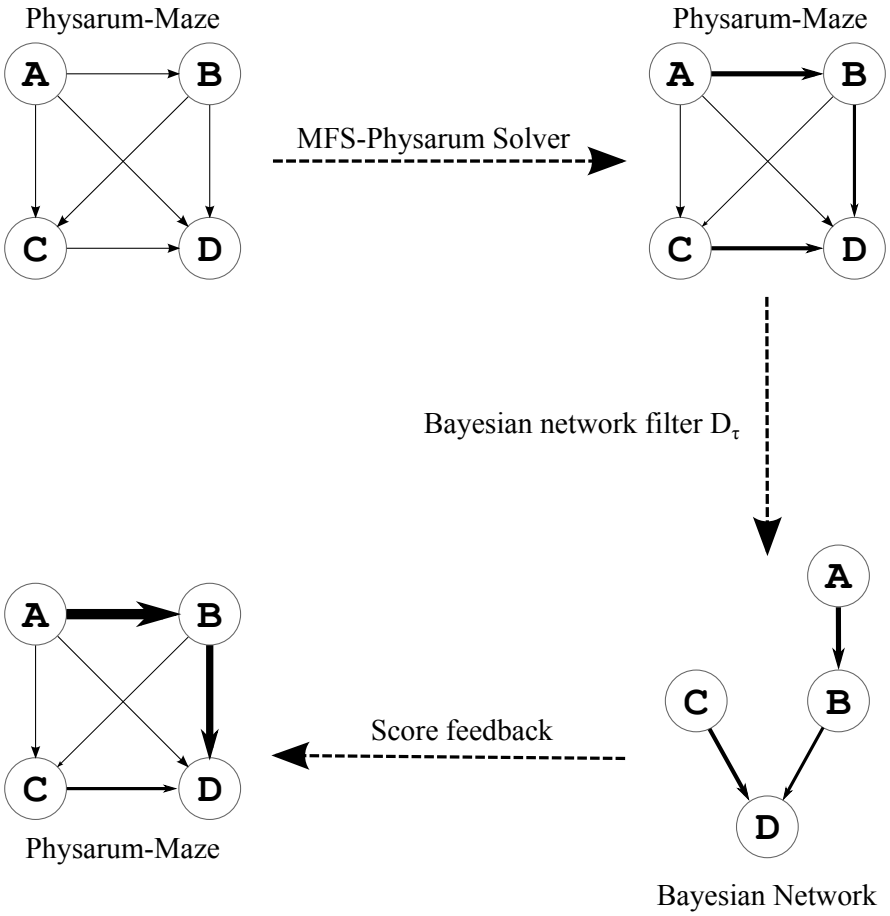


Figure 4.1: One SO-PhyL iteration is shown where first, *MFS-Physarum Solver* is used to update conductivity values of the Physarum-Maze which is further transformed into a Bayesian network by considering threshold D_τ. Next, score feedback is given to the conductivity levels of connections present in the Bayesian network and the Physarum-Maze is updated.

extra parent with respect to the selected direction, see Equation 4.6.

$$\beta = \begin{cases} \frac{score_{X_j}(X_i \in Pa(X_j))}{score_{X_j}(X_i \notin Pa(X_j))} & \text{if } score_{Diff(E_{ij})} > score_{Diff(E_{ji})} \\ \frac{score_{X_i}(X_j \in Pa(X_i))}{score_{X_i}(X_j \notin Pa(X_i))} & else \end{cases} \tag{4.6}$$

If $\beta < 1$, the score with the connection added is higher than the score without extra connection as scores are in negative range. If $\beta > 1$, adding the connection would result in a decrease in score. The value of β is next used to update the conductivity for the following *MFS-Physarum Solver* iteration as shown in Equation 4.7.

$$D_{ij} = D_{ij} + k(1 - \beta) \tag{4.7}$$

The feedback is given to D_{ij} by adding a positive or negative value that is proportional to the increase or decrease in score when adding the connection. Hence, if a connection was selected by *MFS-Physarum Solver* that would decrease the score, D_{ij} is lowered so

that the probability that this connection is chosen by the next iteration of the *MFS-Physarum Solver* is decreased. On the other hand, if adding the connection increases score, conductivity is increased too and the connection is more likely to be part of a shortest path in the next *MFS-Physarum Solver* iteration. The weight of decrease and increase on D_{ij} can be set by constant k. An upper limit for D_{ij} indicated by D_{limit} is defined to avoid endless growing in conductivity. Further the value of D_{ij} cannot be less than zero. Please note, that before a connection is considered to be added to the Bayesian network, it is verified that the connection does not produce a cycle and the child node is valid to accept another parent, indicated by function `addArkMakesSense()` in Algorithm 6. Algorithm 7 shows the procedure of providing score feedback to conductivities in a more compact way, where `calculateBeta()` represents Equation 4.6.

Before continuing with the next *MFS-Physarum Solver* iteration, the global score of Bayesian network $\mathcal{B}$ is determined where SO-PhyL remembers the highest scoring network over all r_{total} iterations. Selecting connections is initially based on randomly set initial conductivity values and on the order in which nodes are chosen to act as food sources leading to variations in learning results. To get stable structure learning performance, an ensemble of SO-PhyLs is used from which the highest scoring network is picked as final Bayesian network. Finally conditional probability distributions of $\mathcal{B}$ regarding $\mathcal{G}$ are estimated using any parameter estimation method implemented in Weka.

Algorithm 6 SO-PhyL: Implementation of function evaluateConductivityByScore()

Require: connections[]
Require: Bayesian network $\mathcal{B}$ defined by graph $\mathcal{G} = (\mathcal{X}, \mathcal{E})$
  whitelist[] $\leftarrow$ Empty
  for all connections M_{ij} **do**
    if $D_{ij} > D_\tau$ **then**
      Add M_{ij} to whitelist
    else
      Remove M_{ij} from $\mathcal{E}$
    end if
  end for
  while whitelist **not** empty **do**
    bestScoreImprovement = 0.0;
    bestConnection;
    for all connections M_{ij} in whitelist **do**
      Remove M_{ij} from $\mathcal{E}$
      if `addArkMakesSense`(E_{ij}) **then**
        $score_{Diff(E_{ij})} = score_{X_j}(X_i \in Pa(X_j)) - score_{X_j}(X_i \notin Pa(X_j))$
      end if
      if `addArkMakesSense`(E_{ji}) **then**
        $score_{Diff(E_{ji})} = score_{X_i}(X_j \in Pa(X_i)) - score_{X_i}(X_j \notin Pa(X_i))$
      end if
      scoreChange = `Max`($score_{Diff(E_{ij})}$, $score_{Diff(E_{ji})}$)
      if scoreChange > bestScoreImprovement **then**
        bestScoreImprovement = scoreChange
        bestConnection = E_{ij} or E_{ji} based on better score
      end if
    end for
    if not bestConnection **then**
      for connections M_{ij} in whitelist **do**
        `giveFeedback`(M_{ij})
      end for
      if $score_{\mathcal{B}} > score_{\mathcal{B}_{best}}$ **then**
        $\mathcal{B}_{best} = \mathcal{B}$
      end if
      return $\mathcal{B}_{best}$
    end if
    Add bestConnection to $\mathcal{E}$
    Remove bestConnection from whitelist
    `giveFeedback`(bestConnection)
  end while
  if $score_{\mathcal{B}} > score_{\mathcal{B}_{best}}$ **then**
    $\mathcal{B}_{best} = \mathcal{B}$
  end if
  return $\mathcal{B}_{best}$

Algorithm 7 SO-PhyL: Implementation of function giveFeedback(E_{ij})

$\beta =$ `calculateBeta()`
$D_{ij} = D_{ij} + k(1 - \beta)$
if $D_{ij} > D_{limit}$ **then**
  $D_{ij} = D_{limit}$
end if
if $D_{ij} < 0$ **then**
  $D_{ij} = 0$
end if

4.2 Parameter examination

The SO-PhyL algorithm is parametrized by set of variables that have been shown to have impact on learning result in preliminary experiments. An overview of configuration parameters is given in Table 4.1 where the first column indicates the parameter notation, the second column shows in which component the parameter is used and column *Source* indicates in which equation or algorithm the parameter appears. A default values derived from preliminary experiments that is used in the following experiments unless stated otherwise is given in the last column. For all structure learning algorithms used within

Parameter	**Component**	**Source**	**Initial value**
r	SO-PhyL	Algorithm 5	2
Ensemble size	SO-PhyL	Algorithm 5	10
μ	*MFS-Physarum Solver*	Equation 2.14	1.2
λ	*MFS-Physarum Solver*	Equation 4.2	0.2
w	*MFS-Physarum Solver*	Equation 4.2	0.5
I_0	*MFS-Physarum Solver*	Equation 2.13	3.0
D_{min}	*MFS-Physarum Solver*	Algorithm 1	0.78
D_{max}	*MFS-Physarum Solver*	Algorithm 1	0.79
D_{τ_0}	SO-PhyL	Equation 4.3	0.8
$D_{\tau_{end}}$	SO-PhyL	Equation 4.3	0.8
D_{limit}	SO-PhyL	Algorithm 6	2.5
k	SO-PhyL	Equation 4.7	3.0

Table 4.1: Overview of configuration parameters for the SO-PhyL algorithm. Columns *Component* and *Source* show where the parameter is used within SO-PhyL. The last column *Initial value* defines the default value of the parameter that is used for experiments if not stated otherwise.

this chapter, the number of maximum parents per nodes is set to five, as no benchmark network is used with a higher in-degree. Learning results of SO-PhyL under different parameter configurations are validated using datasets of 1000 instances that have been sampled for validation of C-PhyL. SO-PhyL is applied to a sampled dataset in order to learn a Bayesian network structure. Quality of structures is evaluated by measurements introduced in Section 3.3.

4.2.1 Number of *MFS-Physarum Solver* iterations r

The SO-PhyL algorithm optimizes a score by giving feedback to connections' conductivities in each of the r_{total} *MFS-Physarum Solver* iterations. In each of these iterations,

a bunch of scores has to be calculated leading to an increase in execution time with increasing iterations. On the other hand, a minimum number of iterations is needed to be able to optimize the score far enough. Thus, the minimum number of iterations is searched that is still high enough to find an optimal scoring network structure. Please remind that parameter r does not set the total number of iterations, but is used to calculate r_{total} as described in Equation 4.1. SO-PhyL is used with eight different values of r (1, 2, 3, 4, 5, 10, 50 and 100) to learn network structures of 75 artificial networks with different characteristics. The networks vary in their number of nodes from 5, 10, 15, 20 to 25. Further, nodes in these networks can have a maximal number of parents of either 1, 2, 3, 4 or 5 and a cardinality of 2, 3 or 4, resulting in a total of 75 networks. Each network is learned with each of the values of r and the number of times where a value of r could learn the highest scoring network regarding the Bayesian score compared to other r values is counted and presented in Table 4.2. The first rows show the number of times

	$r=1$	$r=2$	$r=3$	$r=4$	$r=5$	$r=10$	$r=50$	$r=100$
All	22	27	28	32	31	35	49	**58**
n5	**15**	14	14	14	14	14	14	14
n10	6	8	8	9	10	10	11	**12**
n15	1	3	4	6	5	5	10	**11**
n20	0	2	2	3	2	2	6	**12**
n25	0	0	0	0	0	4	8	**9**
p1	6	9	9	10	9	12	13	**14**
p2	5	6	7	8	7	9	10	**12**
p3	4	4	4	6	7	6	10	**14**
p4	3	3	3	3	4	4	**8**	**8**
p5	4	5	5	5	4	4	8	**10**
c2	7	7	9	9	9	10	13	**19**
c3	6	11	10	12	11	12	18	**20**
c4	9	9	9	11	11	13	18	**19**

Table 4.2: Different values of r are compared regarding the Bayesian score. The counts how often each configuration has learned the highest scoring network compared to the other configurations are given. Row *All* shows the total counts for all 75 benchmark networks while other rows provide fragmented counts for networks with the same number of nodes, parents and cardinality.

SO-PhyL using r (as shown in the corresponding column) was able to find the network structure with highest Bayesian score. Following rows show results for specific networks sharing the same number of nodes, parents or the same cardinality. Investigating the first row, it is not surprising that with increasing r and thus increasing iterations, the highest scoring network structure has been found more often. More iterations mean that there are more chances to optimize the score. It can also be seen, that for networks with a small number of nodes, less iterations are needed to learn an optimal score. With increasing size of nodes, as expected, counts get higher for larger values of r as can be seen in rows `n5`-`n25`. The number of possible edges between nodes grows quadratically with the number of nodes and thus more options have to be validated with increasing node size leading to the need of more optimization iterations. On the other hand, variations

in number of parents or cardinality do not have high effect on learning performance. As expected, increasing r leads to an increase in structure quality, but also increases execution time dramatically. Thus, three different networks `A_n5_p3_a7_c3`, `A_n20_p3_a37_c3` and `A_n50_p3_a97_c3` are used to further study the influence of r on learning quality by plotting the score development as a function of the number of iterations in order to see in which iterations major score optimization is performed. Figure 4.2 shows score development for one SO-PhyL with $r = 10$ resulting in $r_{total} = 100$ iterations for network `A_n5_p3_a7_c3`. Please note that values presented on the horizontal axis of Figure 4.2 show the number of total iterations at points of $r = 1$ (10), $r = 2$ (20) and so on.

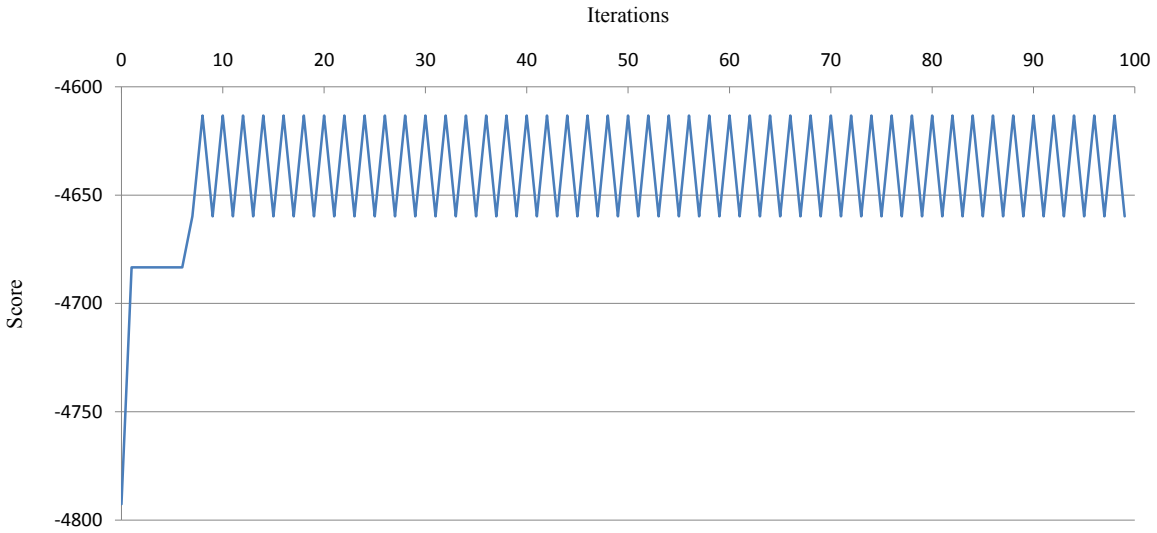


Figure 4.2: Score development of network `A_n5_p3_a7_c3` over 100 iterations.

It can be seen that the highest scoring network structure has already been found after eight iterations. This result is in line with the results presented in Table 4.2 where $r = 1$ was enough for networks with 5 nodes to find the optimal network structure. The following iterations do not provide any score optimization. It can be assumed that within these iterations, score changes are made due to noise artefacts within the dataset or fluctuations of the *MFS-Physarum Solver*. Thus, learning small networks with a high value of r could cause overfitting.

Analysing Figure 4.3, which shows score development for `A_n20_p3_a37_c3` of one SO-PhyL with $r = 10$ resulting in $r_{total} = 1900$ iterations, indicates a similar result. Within the first iterations, the score improvement is strong. But already at $r = 1$, a local maxima is reached although after about 406 iteration, the highest scoring network is found. This graphic does not show clearly if there is a constant score improvement until $r = 3$ or if the larger step at iteration 406 is a result of a lucky choose. But one can see that there is no further score optimization after $r = 2$ except from that at iteration 406.

Development of score for the largest network with 50 nodes `A_n50_p3_a97_c3` is shown in Figure 4.4 where again one SO-PhyL with $r = 10$ resulting in $r_{total} = 12250$ iterations has been performed. Once more, major score optimization is reached within the first $r = 3$ iterations (at iteration 3675). But for network `A_n50_p3_a97_c3`, a slight continuous improvement of score can be observed. Nevertheless, score increase in higher iterations

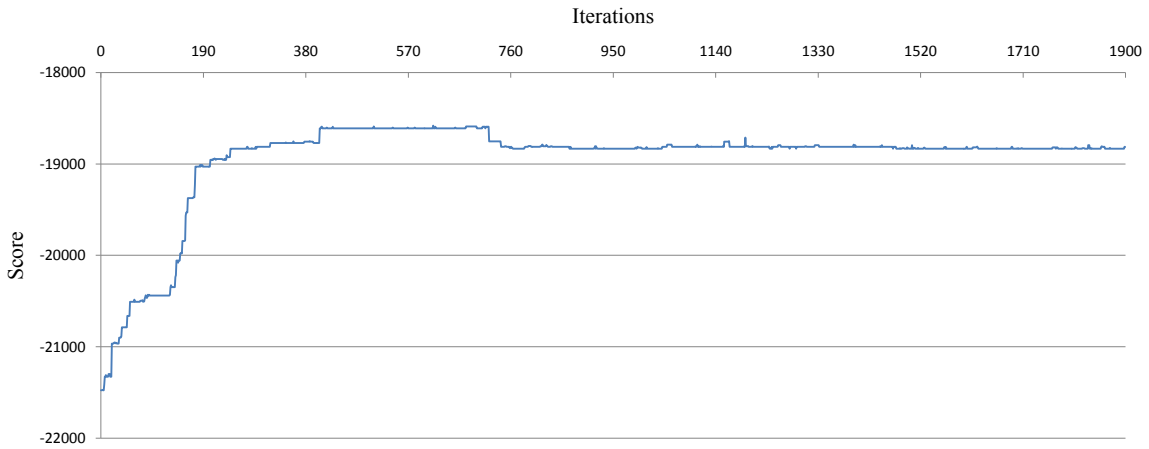


Figure 4.3: Score development of network `A_n20_p3_a37_c3` over 1900 iterations.

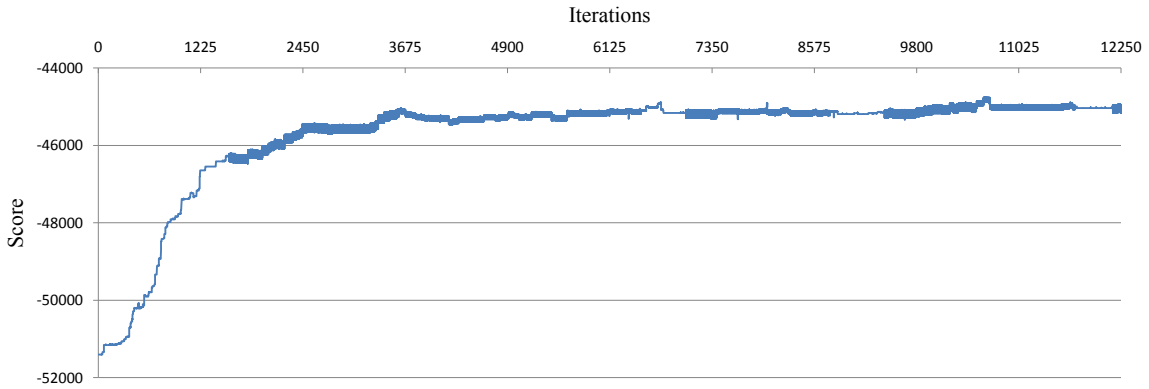


Figure 4.4: Score development of network `A_n50_p3_a97_c3` over 12250 iterations.

is low whereas execution time increases fast. It can thus be concluded, that learning network structures using SO-PhyL with very low values of $r < 5$ offers a good relation between optimizing score and execution time. However, the amount of score increase and how fast score increases depends on the random selection of food source nodes and the random initialization of conductivities. Preliminary results showed that the random selection of food sources is more crucial to the final score than the number of iterations r. It is thus necessary to use an ensemble of SO-PhyLs with a low number of r and chose the best network out of them.

4.2.2 Ensemble size

The same three networks that have been used to analyse score development with increasing number of iterations are used to test if an ensemble size of 10 is appropriate. Obviously, increasing the size of the ensemble increases the chance of getting a higher scoring network as more individual SO-PhyLs are run. On the other hand, execution time increases linearly with ensemble size. Thus, it is studied if using the default value of 10 SO-PhyLs is enough to get network structures of good quality. It is further examined

how stable learning results are by performing the same experiment for each dataset ten times. With respect to result of the previous section, r is set to 3. For `A_n5_p3_a7_c3`, all ten runs delivered exactly the same network structure, concluding that an ensemble size of 10 is enough to get stable learning performance for networks with only five nodes. Learning results for 10 individually performed runs for network `A_n20_p3_a37_c3` is given in Table 4.3 and for network `A_n50_p3_a97_c3` in Table 4.4 respectively.

Run	Bayes	A	T	R	M	E
1	-18571.53	28	24	3	10	1
2	-18579.65	28	25	3	9	0
3	-18557.86	28	25	3	9	0
4	-18571.53	28	24	3	10	1
5	-18588.57	27	23	3	11	1
6	-18536.84	29	25	3	9	1
7	-18579.65	28	25	3	9	0
8	-18585.01	28	25	3	9	0
9	-18554.21	29	25	3	9	1
10	-18557.86	28	25	3	9	0

Table 4.3: Results of ten individual runs of network `A_n20_p3_a37_c3` with $r = 3$ and an ensemble size of 10.

Run	Bayes	A	T	R	M	E
1	-44696.75	72	56	12	29	4
2	-44727.80	72	55	13	29	4
3	-44811.98	71	55	12	30	4
4	-44787.64	72	55	13	29	4
5	-44724.93	71	55	13	29	3
6	-44836.09	70	54	12	31	4
7	-44815.21	69	55	11	31	3
8	-44799.84	71	55	12	30	4
9	-44902.99	69	53	13	31	3
10	-44746.24	71	55	12	30	4

Table 4.4: Results of ten individual runs of network `A_n50_p3_a97_c3` with $r = 3$ and an ensemble size of 10.

Results presented in Table 4.3 show that an ensemble of 10 learners is still large enough for networks with 20 nodes. Although, there are slight variations between the ten runs. Analysis of results for network `A_n50_p3_a97_c3` show that variation between individual runs grow with the number of nodes. Here, learned structures vary already at most in 3 arcs that have notable influence on the Bayesian score. Increasing both, r and the ensemble size could stabilize performance, but comes with major increase in execution time. Hence, minor variations are accepted in order to keep execution time short and the number of learners used is set to 10 for the following experiments.

4.2.3 Exponent μ of Equation 2.14

Exponent μ describes how sigmoid the shape of function defined by Equation 2.14 is and thus how fast the function converges to one. In the common *Physarum Solver*, a value of $\mu = 1.0$ ensured to find the shortest path. Unfortunately, no reports on behaviour of μ are given by Tero *et. al.* for the *MFS-Physarum Solver* except that they used a different value of μ for each of their experiments. For that reason, five different values of μ (1.0, 1.2, 1.5, 2.0 and 3.0) are tested in order to find the one where SO-PhyL performs best regarding the Bayesian score. Hence, a bunch of 90 benchmark networks including the 75 already introduced ones plus 15 networks with 50 nodes in the same configuration as the other networks is analysed. Each of the networks has been learned with each of the values of μ and the number of times where SO-PhyL with a specific value of μ learned the highest scoring network is counted. Results are presented in Table 4.5. Again, results

	$\mu = 1.0$	$\mu = 1.2$	$\mu = 1.5$	$\mu = 2.0$	$\mu = 3.0$
All	**60**	45	42	32	27
n5	13	**14**	**14**	**14**	12
n10	**14**	11	10	7	6
n15	**9**	5	6	2	5
n20	**11**	5	5	4	3
n25	4	**7**	4	5	1
n50	**9**	3	3	0	0
p1	11	12	**13**	9	10
p2	**14**	10	9	10	9
p3	**12**	8	7	4	3
p4	**12**	7	7	4	2
p5	**11**	8	6	5	3
c2	**19**	12	11	12	9
c3	**23**	18	13	9	8
c4	**18**	15	**18**	11	10

Table 4.5: Counts where SO-PhyL using different values of μ learned the highest scoring networks structure among 90 benchmark datasets.

are grouped by the number of nodes, parents and cardinality as described in previous experiments. Table 4.5 shows a clear benefit for $\mu = 1.0$ where results are getting worse with increasing values of μ. From raw score values not presented in this thesis, it can also be seen that differences in score are very small for values of $\mu = 1.0$ and $\mu = 1.2$ but get bigger for higher values of μ. With respect to these observations, a value of $\mu = 1.0$ is used for further experiments presented in this thesis.

4.2.4 Influence of λ

Parameter λ defines the weight that is given to the value of D_{ij} of the former iteration when calculating the updated value of D_{ij} and is also dependent on parameter w. The influence of λ on learning quality with respect to the learned Bayesian score is tested by using the introduced 90 benchmark networks with setting λ to either 0.01, 0.1, 0.2, 0.5.

0.75 and 1.0 using $w = 0.5$. Regarding previous experiments, parameters are set to $r = 3$, $\mu = 1.0$ and ensemble size is set to 10. For 84 of the 90 networks, $\lambda = 0.01$ learned the network structure with highest score. In remaining six networks, $\lambda = 0.1$ reached the highest score. It could further be observed that the difference in score between different values of λ is very high. Recalling Equation 4.2, a very low value of λ leads to a total weight of almost 1 for $D_{ij_{old}}$ where weight w of value $f(Q)$ is 0.5. Thus, a relation between the former value of conductivity and function of flux is defined so that conductivity counts twice the value of flux in adaptation. A configuration like that results in a continuous grow of most conductivity values and thus to a state where a high amount of connections are considered as possible Bayesian network connections as threshold D_τ is exceeded. Hence, the algorithm performed in that case converges to do greedy hill climbing. Obviously this leads to dramatic increase in execution time as nearly all possible connections have to be checked in each iteration. See for example Table 4.6 providing execution times for networks `A_n10_p3_a17_c3` and `A_n50_p3_a97_c3` for six different values of λ. Figure 4.5 illustrates

λ	Execution time in seconds	
	`A_n10_p3_a17_c3`	`A_n50_p3_a97_c3`
0.01	1.14	780.08
0.1	0.70	568.13
0.2	0.57	358.70
0.5	0.30	114.90
0.75	0.22	90.79
1	0.22	80.67

Table 4.6: Execution time in seconds for networks `A_n10_p3_a17_c3` and `A_n50_p3_a97_c3` using different values of λ

the number of connections higher than D_τ before `evaluateConductivityByScore()` is called over r_{total} iterations for network `A_n10_p3_a17_c3` and Figure 4.6 provides same corresponding values for network `A_n50_p3_a97_c3`. It can clearly be seen, that with lower values of λ, the number of connections considered as valid Bayesian network connections is growing. For `A_n10_p3_a17_c3`, using $\lambda = 0.01$ results in a setting where nearly all of the 45 possible connections are exceeding D_τ for some iterations. Hence, a lot of scores have to be calculated in each iteration increasing execution time, see Table 4.6. For network `A_n50_p3_a97_c3`, 1225 possible connections do exist, but even with $\lambda = 0.01$, at most 110 connections have higher conductivity than D_τ. This is because value of I_0 is set to only 3.0. This means that a low amount of sol is flowing through a relatively large Physarum-Maze resulting in fluxes in the range of 10^{-5} for single connections. Thus, increase caused by term $f(Q)$ in adaptation Equation 4.2 is still lower than decrease caused by term $(1 - \lambda w)D_{ij_{old}}$. When increasing I_0 to an appropriate size, the number of connections exceeding D_τ would also converge to the number of possible connections. But, influence of λ on number of connections with $D_{ij} > D_\tau$ can already by seen clearly form graph shown in Figure 4.6.

Purpose of SO-PhyL is not to evaluate all possible connections in each iteration, but using the benefit of the *MFS-Physarum Solver* in combination with score optimization. For that reason, λ is set to 0.2 for further experiments in order to inhibit continuous connection growing and considering only connections as Bayesian network connections

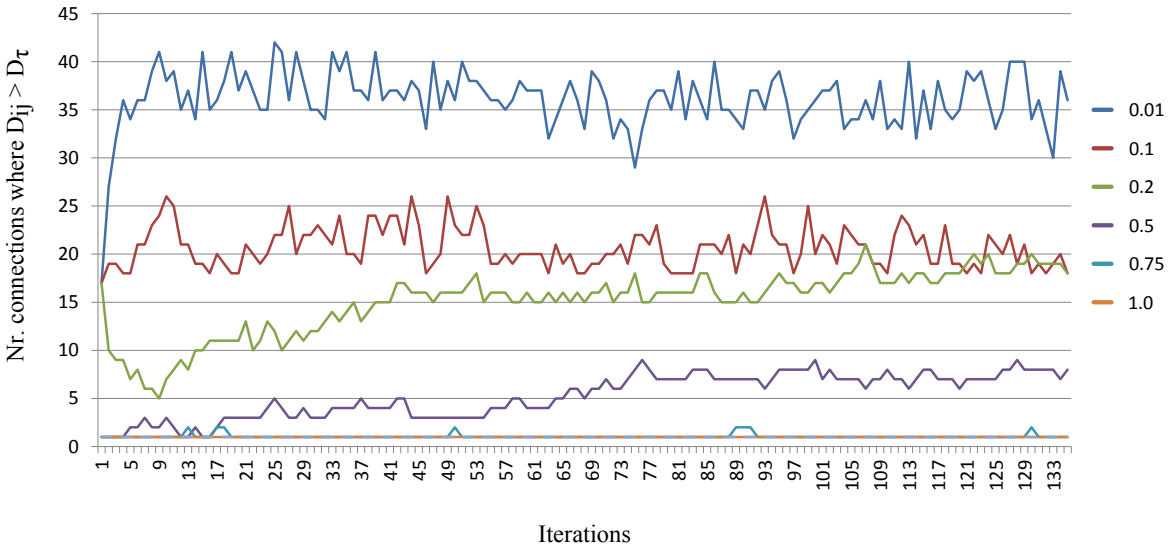


Figure 4.5: Development of number of connections higher than D_τ for network `A_n10_p3_a17_c3` with different values of λ.

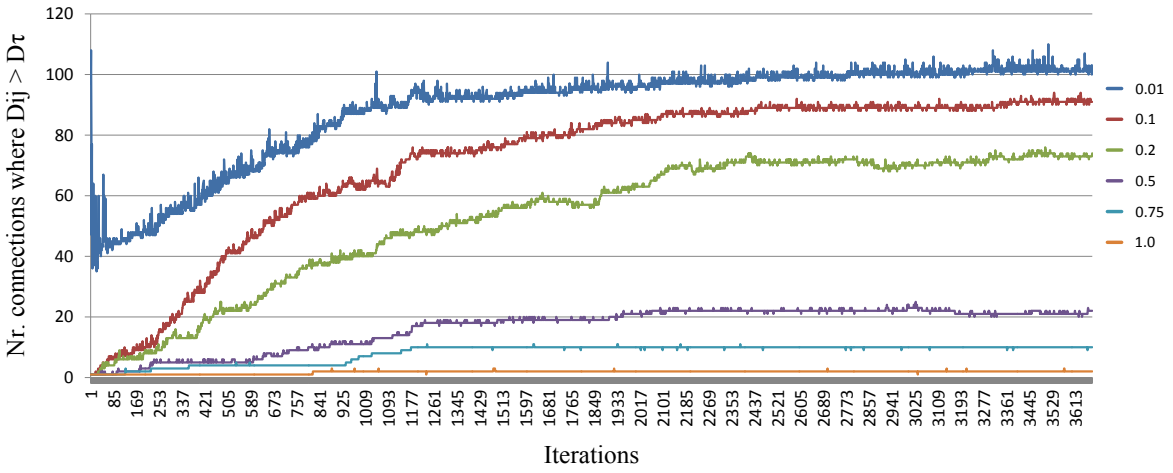


Figure 4.6: Development of number of connections higher than D_τ for network `A_n50_p3_a97_c3` with different values of λ.

if they have been pushed by score feedback or by selection within the *MFS-Physarum Solver*.

4.2.5 Investigating parameter w

Parameter w is supposed to weigh how strong the influence of conductivity changes caused by the *MFS-Physarum Solver* is, compared to the influence to conductivity by giving score feedback. Further, when using a very small value of λ, w does also configure the relation between flux and conductivity in adaptation equation of the *MFS-Physarum Solver*. Hence, w is assumed to have high impact on learning performance which is tested by investigating different values of w (0.1, 0.25, 0.5, 0.75 and 1.0) on the 90

benchmark datasets used in previous sections. For 83 networks, $w = 0.1$ has learned the highest scoring network regarding the Bayesian score. Hence, it can be assumed that the more influence is given to the score feedback part, the better learning performance is. Behaviour of learning under different configuration of w is studied deeper by analysing two networks that have been already used for investigation λ, namely `A_n10_p3_a17_c3` and `A_n50_p3_a97_c3`. For both networks, the number of connections exceeding threshold D_τ and score change over iterations is measured. Figure 4.7 shows score evolution for network `A_n10_p3_a17_c3` with different values of w and Figure 4.8 shows the number of connections exceeding D_τ (and are thus evaluated for score feedback) as a function of the number of iterations.

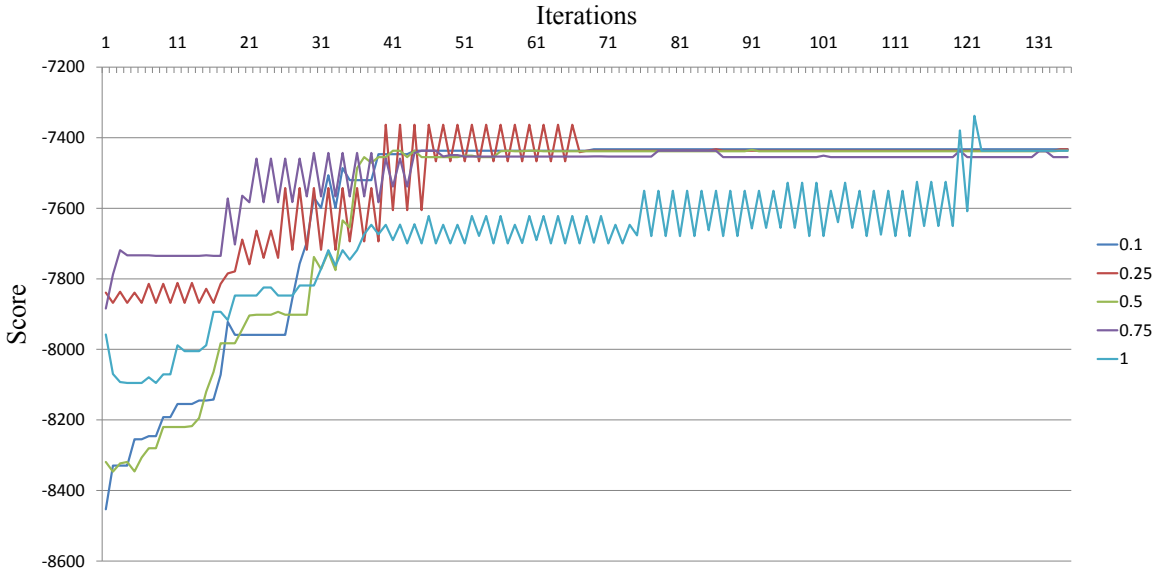


Figure 4.7: Score development over iterations for network `A_n10_p3_a17_c3` using different values of w.

Figures 4.9 and 4.10 show the same evaluations for network `A_n50_p3_a97_c3` respectively.

For the smaller network, the difference in number of connections with high conductivity between different values of w is not clear, but a trend can be seen that with increasing w less connections are overpassing D_τ. Investigating score evolution, all five configurations are evolving differently, but end at almost same results. On the other hand, when analysing benchmark network with 50 nodes, a clear difference is investigated. The smaller w is chosen, the more connections are learned with high variations. The real network includes 97 connection, where $w = 0.1$ is very close to.

This assumption can be verified by analysing the change in score over the iterations where configurations learned higher score with decreasing w. Please remind that a lower value of w results in more impact on conductivity by score feedback and less impact by *MFS-Physarum Solver*. Thus, this result is not surprising as conductivities are changed mostly based on calculated scores. But, giving more weight to the *MFS-Physarum Solver*, increases the flexibility of SO-PhyL as in each iteration, conductivities are changed more by the *MFS-Physarum Solver* and SO-PhyL is thus more likely to avoid getting stuck

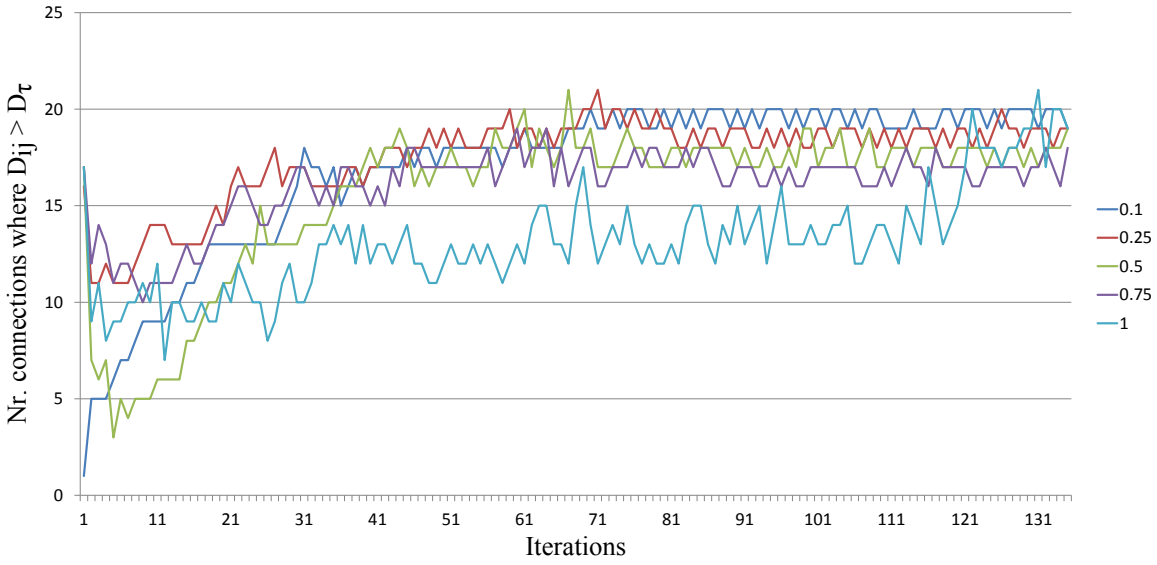


Figure 4.8: Development of number of connections higher than D_τ for network `A_n10_p3_a17_c3` with different values of w.

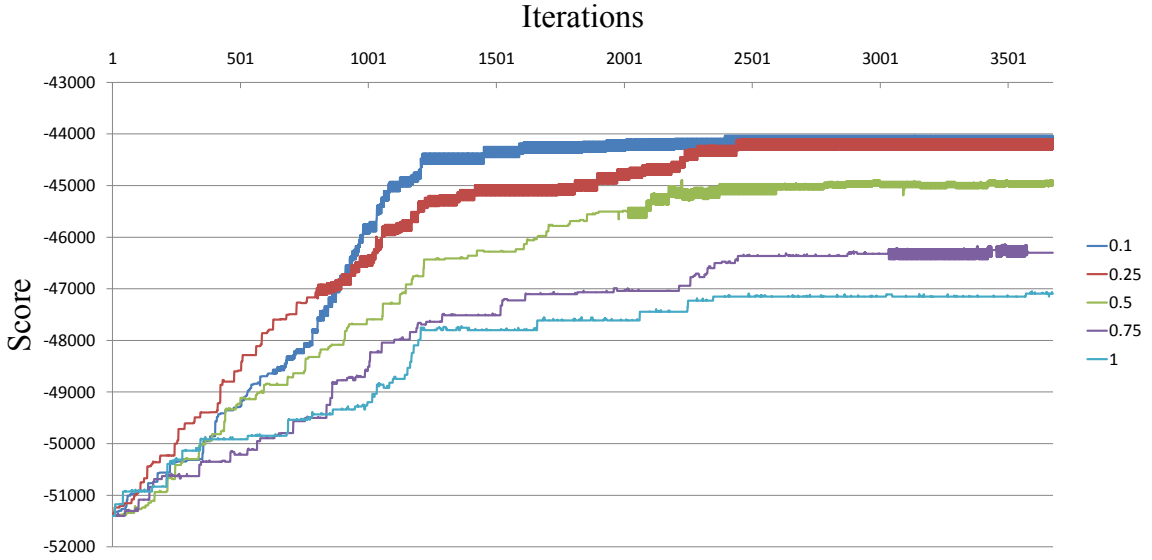


Figure 4.9: Score development over iterations for network `A_n50_p3_a97_c3` using different values of w.

in local maxima. Figure 4.11 prints the evolution of conductivities for all connections of network `A_n10_p3_a17_c3` for five different values of w. Note, that these graphics are not presented for network `A_n50_p3_a97_c3` as there are too many connections to create a clear graph.

In Figure 4.11 and its sub-figures, increasing shiver of conductivity values for connections can be observed with increasing w caused by adaptations of the *MFS-Physarum Solver*. These fluctuations enable connections to be lowered again instead of being stuck at the upper limit of D_{limit} once they reached it. At this point of parameter analysis, it is

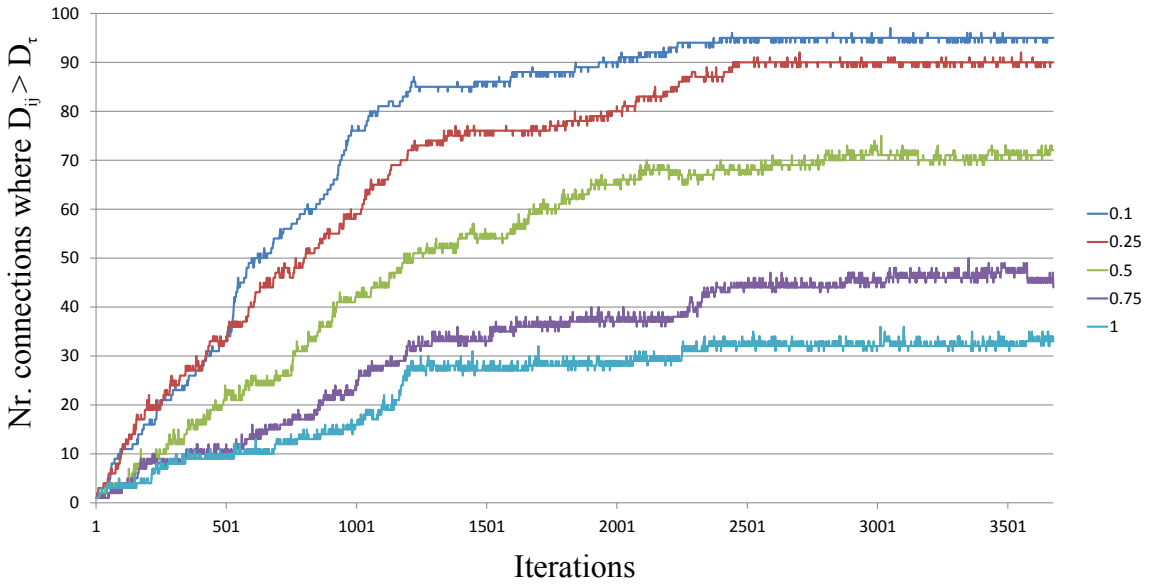
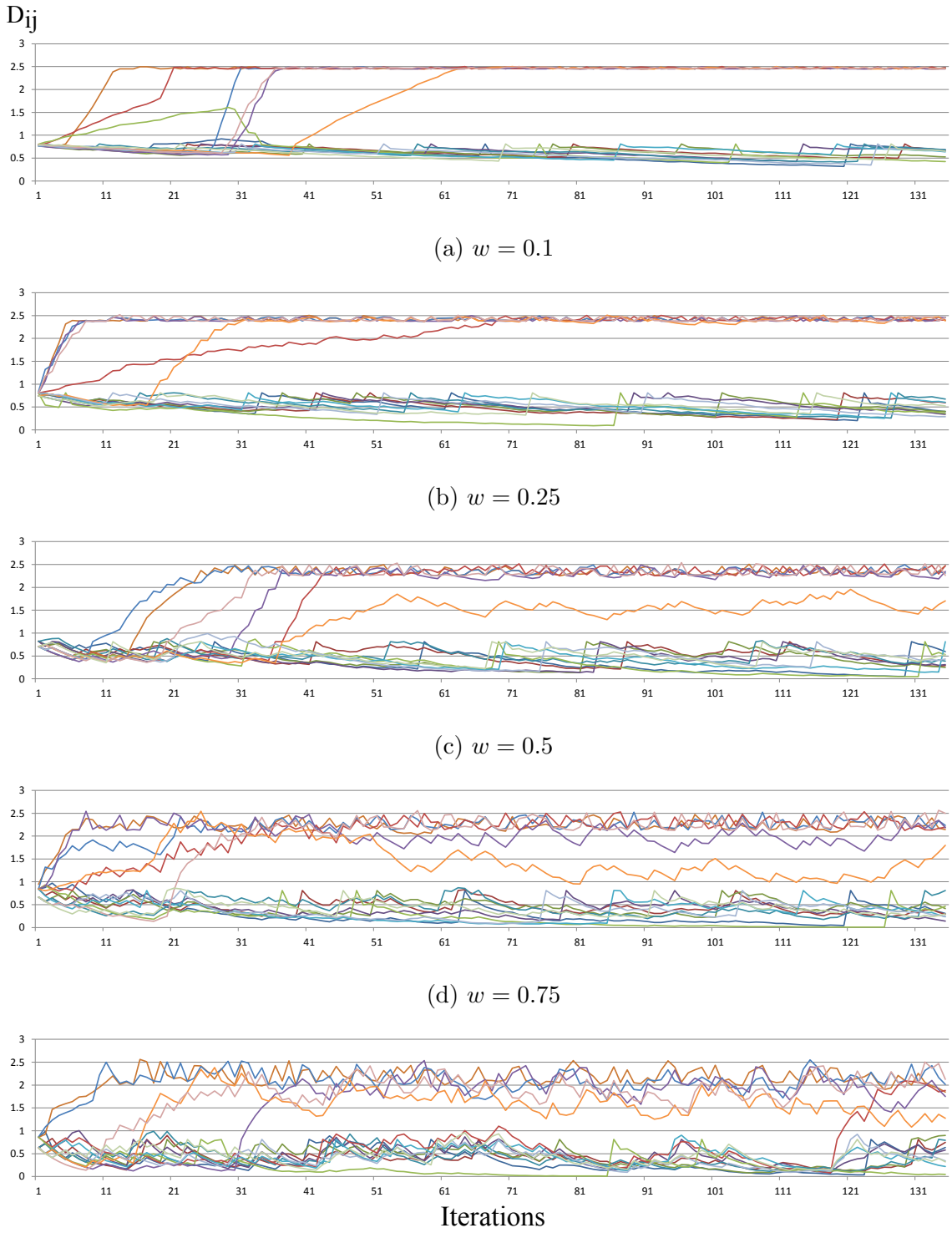


Figure 4.10: Development of number of connections higher than D_τ for network `A_n50_p3_a97_c3` with different values of w.

not clear if shivering is an advantage or a disadvantage as for example parameter I_0 has not yet been studied. When increasing I_0, more connections will survive and possibility of connections being lowered in conductivity again, might give an advantage in avoiding local maxima. On the other hand, this is also possible for small values of w when negative feedback is given to the connection, but with lower speed. Further, experiments showed better results with small values of w. Nevertheless, setting w very small is decreasing also influence of I_0 which is to be studies next. Hence, additional experiments in this section are performed with $w = 0.5$ to keep balance between influence on conductivity by score feedback and by *MFS-Physarum Solver*.



(a) $w = 0.1$

(b) $w = 0.25$

(c) $w = 0.5$

(d) $w = 0.75$

(e) $w = 1.0$

Figure 4.11: Conductivity change for all connections of network `A_n10_p3_a17_c3` using different values of w. Vertical axis shows the connections conductivity values where horizontal axis shows the iterations.

4.2.6 Amount of food I_0

The amount of food I_0 directly influences flux within the Physarum-Maze and has thus influence also on the conductivity values. Tero *et. al.* reported that settings for I_0 are crucial to the final result when using the *Physarum Solver* with multiple food sources [198]. For all their experiments, they used different values of I_0 where no value could be defined that works well for all maze configurations. In SO-PhyL, it is supposed that with increasing I_0, the number of connections with conductivity higher than threshold D_τ increases, too. It follows immediately, that this leads also to an increase in the number of learned connections, especially for networks of larger size. Seven different values 1, 2, 5, 10, 20, 35 and 50 are used as I_0 to learn 90 benchmark datasets and the number of times, SO-PhyL using a specific value of I_0 is performing best with respect to the Bayesian score is counted and presented in Table 4.7.

	$I_0 = 1$	$I_0 = 2$	$I_0 = 5$	$I_0 = 10$	$I_0 = 20$	$I_0 = 35$	$I_0 = 50$
All	19	26	33	40	49	53	**80**
n5	12	13	**15**	14	14	14	13
n10	4	8	12	12	12	12	**14**
n15	2	3	3	10	**13**	11	12
n20	1	2	3	4	7	10	**12**
n25	0	0	0	0	3	6	**14**
n50	0	0	0	0	0	0	**15**
p1	6	9	9	11	13	13	**18**
p2	5	7	9	11	13	14	**17**
p3	3	5	6	8	8	10	**15**
p4	2	2	4	6	6	6	**15**
p5	3	3	5	4	9	10	**15**
c2	5	6	8	9	14	16	**26**
c3	7	10	12	16	18	17	**27**
c4	7	10	13	15	17	20	**27**

Table 4.7: Counts how often SO-PhyL using different values of I_0 has learned the highest scoring network. Rows show results of networks fragmented by their number of nodes, parents and cardinality.

Results show an obvious benefit for higher values of I_0, as expected. With increasing food amount, more connections are pushed in their conductivity so that they are being evaluated regarding their score. But, with increasing connections of higher conductivity, score calculations increase too and execution time grows. Using a value of $I_0 = 50$ takes roughly two times as long as learning network structure using $I_0 = 5$. For networks with a small number of nodes, it can be seen from Table 4.7 that less food amount is already enough to learn a proper network structure. It can thus be defined that a good result can be obtained when using approximately two times the number of nodes as value of I_0.

Next, the number of learned arcs with respect to the number of arcs in the correct network (where the dataset has been sampled from) is examined for different values of I_0. The percentage of learned arcs is calculated by dividing the number of learned arcs for each of the 90 benchmark networks and seven configurations by the number of arcs present

in the corresponding true network. Then, networks of same node size are grouped and the percentage values are averaged among the group. For each value of I_0, these average percentage values are plotted as a function of the number of nodes and shown in Figure 4.12. For small networks, different values of I_0 are learning approximately the same

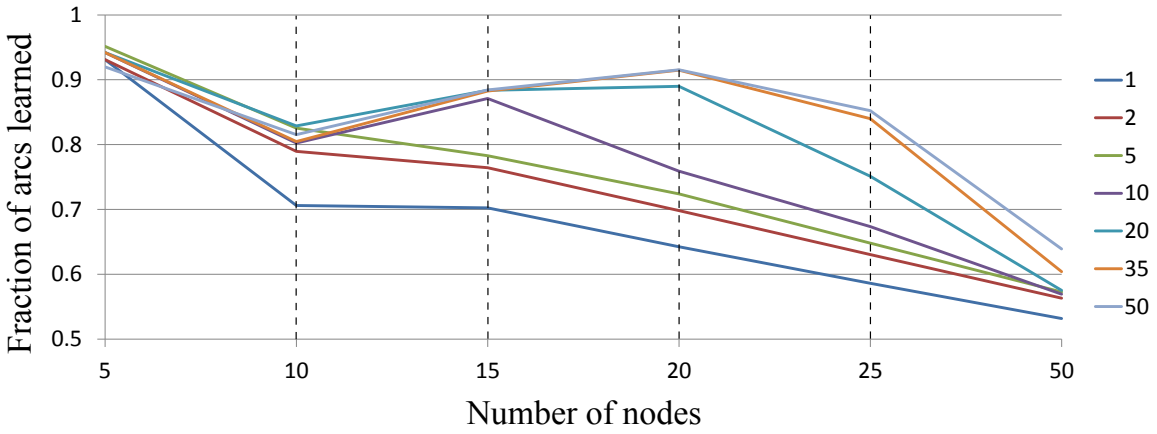


Figure 4.12: Average percentage of learned arcs with respect to the correct number of arcs for different values of I_0 over 90 benchmark networks grouped by the number of nodes.

number of nodes. But for network with a larger amount of nodes, higher values of I_0 are clearly more beneficial. The highest difference can be seen for the group of networks with 20 nodes, and $I_0 = 50$ could learn .92% of the number of arcs in the correct network, where $I_0 = 1$ could only learn .64% which is much too less. In addition, it is noted that no configuration learned a network structure containing more arcs than the original one.

Results in this and the previous section show that high influence can be given to SO-PhyL in the way connections are evolving during *MFS-Physarum Solver* iterations and that the interplay between λ, w and I_0 is very important for the number of learned connections, the quality of learned structures and the Bayesian score.

4.2.7 Conductivity settings

The SO-PhyL algorithm is parametrized by different values for the initial conductivity and conductivity threshold which are highly related to each other. Setting for example the initial conductivities in a range lower than the initial conductivity threshold, no connection is considered as Bayesian network connection at iteration 1. On the other hand, initializing conductivities higher than initial threshold D_τ, all connections are evaluated as Bayesian network arcs. Hence, conductivity parameters D_{min}, D_{max}, D_{τ_0} and $D_{\tau_{end}}$ are evaluated together. Please note that the maximum conductivity D_{limit} is analysed separately as this parameter in not directly related to the others. With respect to previous result, experiments are performed setting $r = 3$, ensemble size to 10, $\lambda = 0.2$, $w = 0.5$ and I_0 individually for networks with different node size. For networks with 5 nodes: $I_0 = 5$, 10 nodes: $I_0 = 20$, 15 nodes: $I_0 = 20$, 20 nodes: $I_0 = 35$, 25 nodes: $I_0 = 50$ and 50 nodes: $I_0 = 50$. 27 different configurations of conductivity parameters are used, see Table 4.8. In configuration 1-15, initially no connections exceed threshold D_τ. In con-

Config	D_{min}	D_{max}	D_{τ_0}	$D_{\tau_{end}}$
1	0.5	1.0	1.1	1.0
2	0.5	1.0	1.1	1.5
3	0.5	1.0	1.1	2.0
4	0.8	1.0	1.1	1.0
5	0.8	1.0	1.1	1.5
6	0.8	1.0	1.1	2.0
7	0.99	1.0	1.1	1.0
8	0.99	1.0	1.1	1.5
9	0.99	1.0	1.1	2.0
10	0.5	0.51	0.8	0.8
11	0.5	0.51	0.8	1.0
12	0.5	0.51	1.0	1.5
13	0.78	0.79	0.8	0.8
14	0.78	0.79	0.8	1.0
15	0.78	0.79	1.0	1.5
16	0.5	1.0	0.75	1.0
17	0.5	1.0	0.8	1.5
18	0.5	1.0	0.9	1.0
19	0.8	1.0	0.9	1.0
20	0.8	1.0	0.9	1.5
21	0.8	1.0	0.95	1.5
22	0.8	1.0	0.8	0.8
23	0.8	1.0	0.8	1.0
24	0.8	1.0	0.8	1.5
25	0.99	1.0	0.9	1.0
26	0.99	1.0	0.9	1.5
27	0.99	1.0	0.9	2.0

Table 4.8: Different configurations for conductivity parameters. In configuration 1-15, initially no connections exceed threshold D_τ. In configurations 16-21, some connections are initially higher than D_τ and for configurations 22-27, all connections are exceeding D_τ at initial state.

figurations 16-21, some connections are initially higher than D_τ and for configurations 22-27, all connections are exceeding D_τ at initial state. 75 benchmark networks of 5 to 25 nodes are learned using these 27 different conductivity configurations and evaluated regarding the Bayesian score. Please note, that networks with 50 nodes are not considered in these experiments as results have become clear already for smaller networks. For 53 of 75 networks, configuration 13 learned the highest scoring networks structure, but over all configurations, there is only minor difference in score. For small networks, all configurations have learned equal structures and for larger networks, score difference is very small and may also result from fluctuations caused by a small ensemble size of 10. Hence, it is observed that SO-PhyL converges to similar results independent of initial conductivity conditions.

4.2.8 Upper conductivity limit D_{limit}

Next, it is studied if varying the upper limit of conductivity D_{limit} (see Algorithm 7) has influence on learning performance of SO-PhyL. Therefore, SO-PhyL is learned with different values of D_{limit} (1.5, 2.5, 3.5, 4.5, 6.0, 10.0, 15.0) using the previously introduced 90 benchmark datasets again. Recalling that D_{limit} is stretching the range of conductivity values, it is assumed that higher values of D_{limit} are delimiting flexibility of SO-PhyL especially in releasing from local maxima as the way from D_{limit} to D_τ is more far. On the other hand, using a very small value of D_{limit} can result in blurring of high scoring connections with other connections. Other parameters are set as described in previous Section 4.2.7 and conductivity configuration 13 is used. Results presented in Table 4.9

	D_{limit}						
	1.5	2.5	3.5	4.5	6.0	10.0	15.0
All	49	55	51	**58**	**58**	55	50
n5	13	13	13	13	13	**15**	13
n10	8	8	10	11	**12**	11	11
n15	11	12	12	11	**15**	10	10
n20	8	**10**	9	**10**	8	9	8
n25	7	**8**	6	**8**	**8**	**8**	6
n50	2	4	1	**5**	2	2	2
p1	14	**15**	14	13	**15**	14	13
p2	14	12	12	14	13	**15**	12
p3	5	8	7	**9**	**9**	7	8
p4	7	11	9	**12**	9	9	7
p5	9	9	9	10	**12**	10	10
c2	14	14	15	**18**	14	17	14
c3	17	19	19	22	**24**	19	18
c4	18	**22**	17	18	20	19	18

Table 4.9: Counts of how often different values of D_{limit} learned the highest scoring network structure. Lines show results fragmented by the number of nodes, parents and cardinality.

show a beneficial configuration if setting D_{limit} to 4.5 or 6.0, but good performance is also

observed for other values. It can be concluded that the influence of D_{limit} is not crucial as most configurations showed good results, but to get best performance, D_{limit} would have to be optimised individually for each network. Especially for smaller networks, nearly all configurations resulted in same network structures. For larger networks, only specific settings learned the highest scoring network, but no value of D_{limit} could be pointed out which outperformed the others clearly. Moreover, achieved scores do not vary heavily within different configurations of D_{limit}.

4.2.9 Score feedback impact factor k

Last parameter to be studied is the score impact factor k of Equation 4.7, where five different values are tested, namely 1.0, 2.0, 3.0, 4.0 and 5.0. The well known 90 benchmark networks are learned under same parametrization as described in Section 4.2.8 and D_{limit} is set to 4.5. Results with different values of k are shown in Table 4.10 where the number of times the highest scoring network has been learned is counted with respect to the Bayesian score. A clear benefit can be seen when using $k = 5.0$ which has learned the

	$k = 1.0$	$k = 2.0$	$k = 3.0$	$k = 4.0$	$k = 5.0$
All	44	54	54	62	**71**
n5	13	14	14	**15**	14
n10	12	12	12	**13**	11
n15	10	10	11	**12**	**12**
n20	7	**10**	8	**10**	9
n25	2	8	9	9	**10**
n50	0	0	0	3	**15**
p1	13	15	14	16	**17**
p2	11	12	15	14	**16**
p3	7	8	7	11	**12**
p4	8	11	8	11	**13**
p5	5	8	10	10	**13**
c2	15	16	15	16	**21**
c3	16	20	18	21	**24**
c4	13	18	21	25	**26**

Table 4.10: Counts of how often different values of k learned the highest scoring network structure. Lines show results fragmented by the number of nodes, parents and cardinality.

highest scoring network for 71 of the 90 benchmark datasets. It can further be seen, that especially for networks with larger number of nodes or larger number of parents, higher k values are boosting learning performance where its more clear for networks with different number of nodes. Instead, no correlation to cardinality can be observed.

4.3 Experiments and analysis with benchmark networks

In the previous sections, a set of SO-PhyL parameters has been investigated in order to find a configuration that increases learning performance. In this section, a bunch of benchmark networks is used to compare SO-PhyL to state of the art learning algorithms. SO-PhyL is used with two different configurations, see Table 4.11. For networks with

	r	$E.S.$	μ	λ	w	D_{min}	D_{max}	D_{τ_0}	$D_{\tau_{end}}$	D_{limit}	k
SO-PhyL-1	3	10	1	0.2	0.5	0.78	0.79	0.8	0.8	4.5	5
SO-PhyL-2	3	10	1	0.01	0.1	0.78	0.79	0.8	0.8	4.5	5

Table 4.11: Different parameter configurations for SO-PhyL for artificial networks.

5 nodes: $I_0 = 5$, 10 nodes: $I_0 = 20$, 15 nodes: $I_0 = 20$, 20 nodes: $I_0 = 35$, 25 nodes: $I_0 = 50$ and 50 nodes: $I_0 = 50$ is used. The LAGD algorithm is again used with 5 good operations and a look ahead step size of 2. Tabu-Search is performed with tabu list size of 5 performing 500 runs. For Simulated Annealing, initial temperature t_0 is set to 10 and is reduced in each iteration by 0.999. Simulated Annealing is performed using 10,000 runs. All of these three algorithms have been learned with using arc reversal and not initializing network as a naive Bayes network. Further, as benchmark networks do not contain nodes with more than five parents, the maximum in-degree was set to 5 for all learning algorithms.

4.3.1 Artificial benchmark networks

Five learning algorithms are compared by using the same selection of 18 artificial benchmark networks including different numbers of nodes, parents and cardinalities to get sparse, normal and crowded networks that have been used for analysis of C-PhyL before. For all networks, datasets containing 1000 instances are sampled and used for structure learning. Results are analysed using same metrics introduced for C-PhyL in Section 3.3 and are presented in Tables 4.12, 4.13 and 4.14.

It can be immediately seen, that LAGD algorithm performs best on all benchmark networks except from two networks, `A_n25_p5_a67_c4` and `A_n50_p1_a49_c2`. Nevertheless, SO-PhyL shows comparable performance with only small difference in Bayesian score compared to LAGD. Further, second configuration SO-PhyL-2 learns better networks than SO-PhyL-1 which is in line with previous experiments. On the other hand, as more connections have to be evaluated for score feedback with SO-PhyL-2, execution time is longer than with SO-PhyL-1. In six of the 18 networks, SO-PhyL-2 outperformed Tabu search and for two more networks, SO-PhyL reached same results. Compared to Simulated Annealing, SO-PhyL learned a better network structure in 14 cases and same structure for one benchmark network. Investigating Tables 4.12, 4.13 and 4.14, it is observed that SO-PhyL performs worse for larger networks. Preliminary experiments for parameter studies showed that configuration of SO-PhyL becomes more difficult with growing number of nodes and that more iterations and a larger ensemble is needed to learn good network structures. But, increasing these parameters, execution time will increase, too. Especially for networks of bigger size, execution time is tremendously higher

Dataset	Learner	Bayes	A	T	R	M	E	t
	SO-PhyL-1	-3357.08	3	1	2	1	0	0.492
	SO-PhyL-2	-3357.08	3	1	2	1	0	0.515
A_n5_p1_a4_c2	LAGD	-3357.08	3	1	2	1	0	0.136
	Tabu	-3357.08	3	1	2	1	0	0.063
	SA	-3357.08	3	2	1	1	0	0.171
	SO-PhyL-1	-4613.24	7	3	4	0	0	0.133
	SO-PhyL-2	-4608.62	7	4	3	0	0	0.132
A_n5_p3_a7_c3	LAGD	**-4575.17**	7	7	0	0	0	0.031
	Tabu	-4613.24	7	3	4	0	0	0.072
	SA	-4614.23	7	2	5	0	0	0.215
	SO-PhyL-1	-5454.76	7	6	1	0	0	0.123
	SO-PhyL-2	-5454.76	7	6	1	0	0	0.121
A_n5_p5_a7_c4	LAGD	**-5426.88**	7	7	0	0	0	0.028
	Tabu	-5454.76	7	6	1	0	0	0.071
	SA	-5477.31	8	4	3	0	1	0.467
	SO-PhyL-1	-4162.01	7	3	2	4	2	0.732
	SO-PhyL-2	-4162.01	7	3	2	4	2	0.693
A_n10_p1_a9_c2	LAGD	**-4159.06**	9	4	2	3	3	0.035
	Tabu	-4161.09	9	3	2	4	4	0.094
	SA	-4161.61	11	5	2	2	4	0.212
	SO-PhyL-1	-7432.82	17	13	3	1	1	1.196
	SO-PhyL-2	-7321.45	17	16	1	0	0	1.017
A_n10_p3_a17_c3	LAGD	**-7318.79**	17	17	0	0	0	0.052
	Tabu	-7364.46	20	13	4	0	3	0.124
	SA	-7439.09	22	14	3	0	5	0.482
	SO-PhyL-1	-12154.17	13	8	4	10	1	0.891
	SO-PhyL-2	-12154.17	13	8	4	10	1	0.769
A_n10_p5_a22_c4	LAGD	**-12087.45**	14	11	3	8	0	0.066
	Tabu	-12154.45	13	7	5	10	1	0.119
	SA	-12247.75	16	6	8	8	2	0.463

Table 4.12: Comparison of five different structure learning algorithms for networks with 5 or 10 nodes. The highest score for each network is written in bold face.

Dataset	Learner	Bayes	A	T	R	M	E	t
	SO-PhyL-1	-8034.11	15	7	5	2	3	3.741
	SO-PhyL-2	-8034.11	15	7	5	2	3	4.719
A_n15_p1_a14_c2	LAGD	**-8022.11**	15	9	4	1	2	0.058
	Tabu	-8026.04	16	8	5	1	3	0.175
	SA	-8039.21	17	8	5	1	4	0.216
	SO-PhyL-1	-12529.00	26	21	3	3	2	4.298
	SO-PhyL-2	-12507.52	26	21	3	3	2	4.563
A_n15_p3_a27_c3	LAGD	**-12391.77**	27	25	2	0	0	0.097
	Tabu	-12414.65	29	24	3	0	2	0.180
	SA	-12592.11	32	18	7	2	7	0.503
	SO-PhyL-1	-18933.27	12	11	1	25	0	2.641
	SO-PhyL-2	-18933.27	12	11	1	25	0	2.650
A_n15_p5_a37_c4	LAGD	**-18927.73**	12	12	0	25	0	0.101
	Tabu	-19081.44	13	9	3	25	1	0.252
	SA	-19071.30	11	9	2	26	0	0.279
	SO-PhyL-1	-9827.77	23	11	6	2	6	15.241
	SO-PhyL-2	-9827.77	23	11	6	2	6	23.506
A_n20_p1_a19_c2	LAGD	**-9814.13**	24	18	0	1	6	0.116
	Tabu	-9819.85	24	11	6	2	7	0.321
	SA	-9867.50	34	5	11	3	18	0.258
	SO-PhyL-1	-18285.67	35	32	3	2	0	16.480
	SO-PhyL-2	-18266.23	37	33	2	2	2	17.828
A_n20_p3_a37_c3	LAGD	**-18012.18**	37	36	1	0	0	0.156
	Tabu	-18212.56	37	33	3	1	1	0.302
	SA	-18924.24	38	13	13	11	12	0.443
	SO-PhyL-1	-25700.10	21	8	9	35	4	10.229
	SO-PhyL-2	-25700.10	21	8	9	35	4	12.674
A_n20_p5_a52_c4	LAGD	**-25553.45**	19	17	2	33	0	0.193
	Tabu	-25705.61	22	11	7	34	4	0.343
	SA	-25705.49	21	10	7	35	4	0.413

Table 4.13: Comparison of five different structure learning algorithms for networks with 15 or 20 nodes. The highest score for each network is written in bold face.

Dataset	Learner	Bayes	A	T	R	M	E	t
	SO-PhyL-1	-12600.69	25	17	4	3	4	34.478
	SO-PhyL-2	-12600.63	26	17	4	3	5	68.441
A_n25_p1_a24_c2	LAGD	**-12595.36**	25	19	1	4	5	0.179
	Tabu	-12600.83	27	15	6	3	6	0.418
	SA	-12669.60	38	9	11	4	18	0.257
	SO-PhyL-1	-22592.73	49	30	11	6	8	60.765
	SO-PhyL-2	-22502.16	50	29	11	7	10	72.099
A_n25_p3_a47_c3	LAGD	**-21797.08**	47	44	3	0	0	0.441
	Tabu	-22440.99	52	31	11	5	10	0.651
	SA	-22354.81	57	28	16	3	13	8.351
	SO-PhyL-1	-32382.57	21	19	1	47	1	20.406
	SO-PhyL-2	-32382.57	21	19	1	47	1	29.328
A_n25_p5_a67_c4	LAGD	-32328.02	23	15	5	47	3	0.394
	Tabu	**-32101.45**	27	25	1	41	1	0.482
	SA	-32494.87	24	16	4	47	4	0.439
	SO-PhyL-1	-25188.25	37	23	9	17	5	222.666
	SO-PhyL-2	-25132.44	70	28	10	11	32	3031.382
A_n50_p1_a49_c2	LAGD	-25059.52	72	31	10	8	31	2.439
	Tabu	**-25058.40**	70	29	10	10	31	3.520
	SA	-25167.02	107	23	15	11	69	489.163
	SO-PhyL-1	-44028.47	88	69	13	15	6	751.566
	SO-PhyL-2	-43942.11	91	75	10	12	6	1785.637
A_n50_p3_a97_c3	LAGD	**-43520.83**	95	83	6	8	6	5.357
	Tabu	-43773.03	98	76	15	6	7	3.597
	SA	-44917.91	105	54	26	17	25	0.590
	SO-PhyL-1	-64241.13	29	21	6	115	2	171.988
	SO-PhyL-2	-63934.40	40	26	8	108	6	667.907
A_n50_p5_a142_c4	LAGD	**-63318.81**	44	41	3	98	0	1.737
	Tabu	-63605.19	46	32	8	102	6	2.950
	SA	-63939.13	41	28	8	106	5	0.539

Table 4.14: Comparison of five different structure learning algorithms for networks with 25 or 50 nodes. The highest score for each network is written in bold face.

for SO-PhyL compared to other learning methods. A reason is that SO-PhyL is performing an ensemble of learners where each one performs a number of iterations in which a hill climbing algorithm is performed on a subset of possible connections in addition with solving a linear equation system of quadratic dimension of the number of nodes in the network. Neither the *MFS-Physarum Solver* implementation nor the score feedback evaluation is yet optimized for speed or memory usage.

Studying results deeper, it can be seen that the relation between the number of true positive arcs and the number of true but reversed arcs is particular low compared to other learning algorithms, meaning that SO-PhyL tends to learn more reversed arcs than other methods. SO-PhyL determines directions within each iteration by calculating a score for both directions and choosing the more profitable one. Results in Tables 4.12, 4.13 and 4.14 indicate that this strategy does not work well compared to other methods and has to be optimized.

4.3.2 Real benchmark networks

The SO-PhyL algorithm has been studied in detail by analysing behaviour under different parameter configurations and has been compared to state of the art learning methods by using artificially generated benchmark networks. Next, SO-PhyL is applied to seven real world benchmark networks introduced in Section 2.5 and quality of learned network structures is compared to LAGD and Tabu Search. Please note that Simulated Annealing is not considered in these experiments because of Weka implementation issues. Datasets with 1000 instances sampled for experiments of C-PhyL are used again to learn the network structures of the benchmark networks. Three different parameter configurations of SO-PhyL are compared, described in Table 4.15. Again, for networks with 5 nodes:

	r	$E.S.$	μ	λ	w	D_{min}	D_{max}	D_{τ_0}	$D_{\tau_{end}}$	D_{limit}	k
SO-PhyL-1	3	10	1	0.2	0.5	0.78	0.79	0.8	0.8	4.5	5
SO-PhyL-2	3	10	1	0.01	0.1	0.78	0.79	0.8	0.8	4.5	5
SO-PhyL-3	5	15	1	0.2	0.5	0.78	0.79	0.8	0.8	4.5	5

Table 4.15: Different parameter configurations for SO-PhyL for real networks.

$I_0 = 5$, 10 nodes: $I_0 = 20$, 15 nodes: $I_0 = 20$, 20 nodes: $I_0 = 35$, 25 nodes: $I_0 = 50$ and 50 nodes: $I_0 = 50$ is used. Learned network structures are compared to the original structures and analysed using metrics described in Section 3.3 where execution time **t** is given in seconds. Results can be seen in Table 4.16, where the best score for each network is written in bold face.

For small networks, all test methods deliver equal network structures, see network *Cancer* and *Earthquake*. It can further be seen, that SO-PhyL performs little worse for network *Asia* where difference in score is minimal. But all three configurations of SO-PhyL learned one additional arc and found one reversed arc less than LAGD and Tabu Search did. For *Insurance* network, SO-PhyL-1 clearly outperforms LAGD and Tabu Search in Bayesian score, number of correctly learned arcs, and extra arcs. This benchmark network shows that SO-PhyL is able to beat state of the art learning methods for particular networks. The learned structure for the *Insurance* network by SO-PhyL-1 is illustrated in Figure 4.13.

Dataset	Nodes	Arcs	Learner	Bayes	A	T	R	M	E	t
Cancer	5	4	SO-PhyL-1	-2235.90	4	2	2	0	0	0.698
			SO-PhyL-2	-2235.90	4	2	2	0	0	0.491
			SO-PhyL-3	-2235.90	4	2	2	0	0	0.129
			LAGD	-2235.90	4	2	2	0	0	0.092
			Tabu	-2235.90	4	2	2	0	0	0.073
Earthquake	5	4	SO-PhyL-1	-521.43	4	4	0	0	0	0.194
			SO-PhyL-2	-521.43	4	4	0	0	0	0.117
			SO-PhyL-3	-521.43	4	4	0	0	0	0.113
			LAGD	-521.43	4	4	0	0	0	0.077
			Tabu	-521.43	4	4	0	0	0	0.077
Asia	8	8	SO-PhyL-1	-2319.06	9	4	2	2	3	0.474
			SO-PhyL-2	-2319.06	9	4	2	2	3	0.410
			SO-PhyL-3	-2319.06	9	4	2	2	3	0.756
			LAGD	**-2318.45**	8	4	3	1	1	0.030
			Tabu	**-2318.45**	8	4	3	1	1	0.102
Insurance	27	52	SO-PhyL-1	**-15429.57**	50	29	12	11	9	106.994
			SO-PhyL-2	-15637.49	53	29	12	11	12	144.044
			SO-PhyL-3	-15436.49	51	29	12	11	10	261.776
			LAGD	-15772.38	55	25	13	14	17	0.655
			Tabu	-15740.36	58	26	14	12	18	1.053
Alarm	37	46	SO-PhyL-1	-11350.81	50	33	8	5	9	191.712
			SO-PhyL-2	-11349.07	54	32	8	6	14	478.263
			SO-PhyL-3	-11349.40	52	32	8	6	12	486.408
			LAGD	**-11247.43**	54	38	4	4	12	0.867
			Tabu	-11299.76	55	34	8	4	13	1.340
Barley	48	84	SO-PhyL-1	-63576.16	65	15	27	42	23	569.444
			SO-PhyL-2	-63200.43	73	15	30	39	28	2048.472
			SO-PhyL-3	-63502.07	58	14	26	44	18	1440.081
			LAGD	**-61790.39**	81	23	28	33	30	15.067
			Tabu	-63021.69	80	15	33	36	32	10.189
Hailfinder	56	66	SO-PhyL-1	-51455.73	57	29	15	22	13	664.784
			SO-PhyL-2	-51376.04	72	33	15	18	24	2933.756
			SO-PhyL-3	-51437.35	59	28	16	22	15	1700.770
			LAGD	**-51322.70**	75	35	14	17	26	5.006
			Tabu	-51374.87	76	32	17	17	27	5.295

Table 4.16: Comparison of three different SO-PhyL configurations to LAGD and Tabu Search using seven real benchmark networks. First column shows the name of the networks. Column *Nodes* shows how many nodes the original network has and column *Arcs* indicated the number of arcs in the original network. Remaining columns show the metrics by which learning methods are evaluated, see Table 3.3. Highest score for each network is given in bold face.

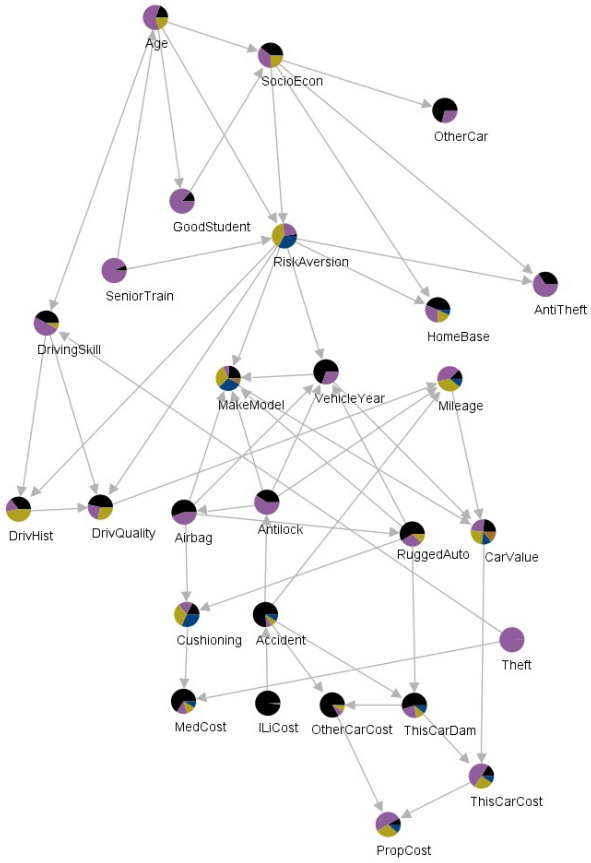


Figure 4.13: Structure of the *Insurance* network learned by SO-PhyL-1.

For the remaining three biggest networks, LAGD could learn the highest scoring network structure where score for SO-PhyL configurations is still good for *Alarm* and *Hailfinder* networks. Studying network *Barley*, score difference of Tabu Search and SO-PhyL to LAGD is relatively big compared to results of other networks which can be caused by the fact that *Barley* has many arcs per node and is thus more crowded. Over all networks, it can be observed that SO-PhyL-1 and SO-PhyL-3 suffer from learning less arcs compared to other algorithms and the true networks, but do thus learn also less extra arcs than other methods. SO-PhyL-2 instead, learns more arcs due to very small values of parameters λ and w and thus performs better for the three largest networks than for other SO-PhyL configurations. Analysing execution time shows a clear disadvantage for the SO-PhyL algorithm which has already been observed while studying artificial benchmark networks. Boosting SO-PhyL in speed and memory usage is probably the most important task for future work. Behaviour of execution time with increasing number of instances is not further studied for SO-PhyL because execution time is mostly dependent on the number of nodes for *MFS-Physarum Solver* and on score calculation methods which are equal for SO-PhyL, LAGD and Tabu Search. Increase in execution time is thus observed to be equally growing for all three methods with increasing instances of datasets, see for example Figure 4.14 showing logarithm of execution time for the *Asia* network.

The main principle of SO-PhyL is to consider only a subset of all possible connections as Bayesian network connections in each *MFS-Physarum Solver* iteration. Hence, defining

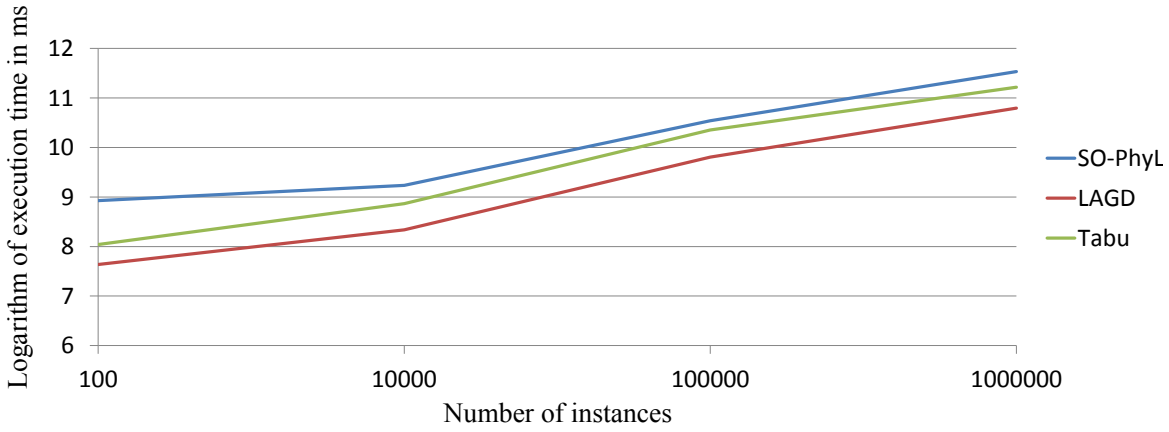


Figure 4.14: Logarithm of execution time in milliseconds as a function of the number of instances in the *Asia* dataset where learning algorithms are applied to.

this subset and therefore defining the conductivity parameters is the most crucial task when using SO-PhyL and is analysed more carefully by using the *Asia* network as an example network. Figure 4.15 shows conductivity evolution over r_{total} iterations with $r = 3$ for one ensemble learner using parameters defined for SO-PhyL-1.

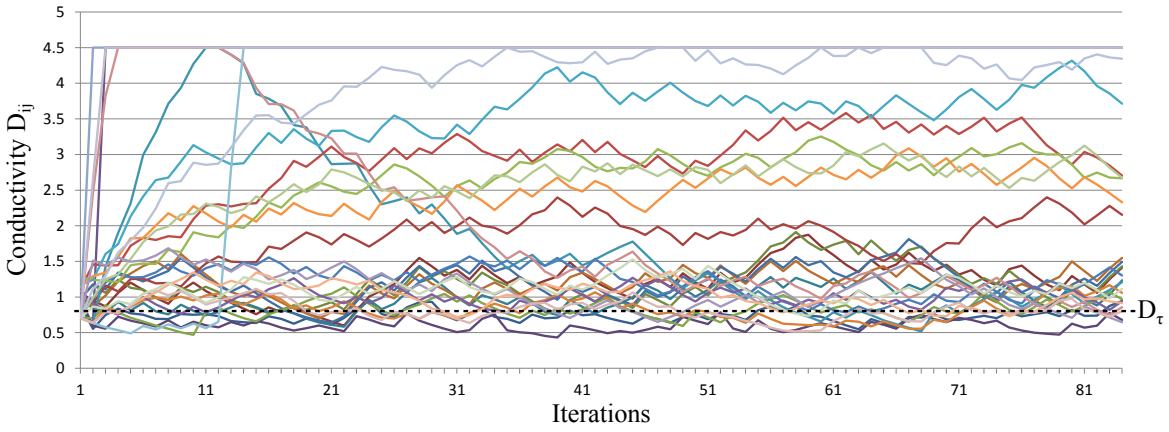


Figure 4.15: Evolution of conductivity values of *Asia* connections over $r_{total} = 84$ iterations using SO-PhyL-1. The threshold D_τ is shown by a dashed line.

It can be seen that immediately after the first iterations, there are some connections where conductivity is increased very fast while other connections are decreased in conductivity due to score feedback. It can be further observed, that there are roughly three groups of connections. The first group grows really fast and stays at the upper border D_{limit} while the second group has increased conductivity but is not reaching D_{limit}. Connections of this group are oscillating forth and back between 1.5 and D_{limit}. The last group includes connections that cannot be pushed far enough by positive score feedback to finally leave the area below 1.5. It can be assumed that connections of group one are increasing score the most and are the most valuable connections in the network. Connections of the intermediate group are increasing score, but may be in competition with other connections or do not have enough impact on score to be pushed to the upper limit.

Connections of the last group with low conductivity values are supposed to not increase score of the Bayesian network if they are added. On the other hand, as a result of low threshold D_τ, most of these connections are considered for score evaluation. Setting $D_{\tau_{end}}$ to 1.5 would lead to a better subset of connections for score ratings, but preliminary experiments showed that there are some important connections within this group that can lead to an increase in score. Hence, future experiments are needed to study more carefully why these connections are not pushed by score feedback and how the algorithm can be updated to end up with a configuration, where all connections are either near D_{limit} or below $D_{\tau_{end}}$ in the final iteration. This would result in a more stable and time efficient SO-PhyL algorithm that would probably also avoid the need of learning an ensemble. Interesting behaviour can be observed at iteration 11, where two connections that are at D_{limit} start immediately to fall down to the lowest group. These two connections are `XRay`↔`Tuberculosis` illustrated in a darker blue and `XRay`↔`Lung Cancer` printed in skin color. It can also be noted, that connection `XRay`↔`Tuberculosis or Lung Cancer` shots up at iteration 11, drawn in light blue. This is a great example on how SO-PhyL replaced two former score increasing connections by another connection while investigating that conditional probabilities of `XRay` on `Lung Cancer` and `Tuberculosis` is modelled by connection `XRay`↔`Tuberculosis or Lung Cancer` leading to a situation where the former two connections do not lead to a score increase any more, but to a decrease in score. Hence, these two connections are lowered in their conductivity and are replaced in the Bayesian network.

4.4 Conclusion and future work

In this chapter, a second novel algorithm based on *Physarum polycephalum* called SO-PhyL has been introduced that resolves drawbacks reported for C-PhyL by using the *Physarum Solver* for multiple food sources to search the space of possible network structures in order to optimize a score measuring the fit of the networks underlying distribution to the distribution observed in the dataset. The algorithm starts with a random network structure and uses the *MFS-Physarum Solver* to select a subset of connections to be evaluated if the connection increases or decreases the score when adding it to the current network and a positive or negative feedback is given to the conductivity value of the connection based on the relative score change. Based on score feedback and *MFS-Physarum Solver* equations, conductivity values change over iterations of the *MFS-Physarum Solver* and thus also the set of connections handled as Bayesian network connections. The algorithm runs over a predefined number of iterations and connections are ideally converging to have a very high conductivity or to disappear so that after all iterations have been performed, a subset of connections remains that is next inserted into the final Bayesian network structure.

Experiments with artificial and real benchmark networks showed, that not all connections are converging as expected and that SO-PhyL has to be parametrized carefully to retrieve a high scoring network. Further, it has been shown that the initial randomly set conductivity values are influencing learning performance. Hence, an ensemble of SO-PhyLs is run and the network structure with highest score over all evaluated structures is taken as final network. One of the main tasks for future work is to modify the algorithm

in a way such that all conductivities are converging to either the upper limit D_{limit} or to zero so that a clear subset of connections survives that builds the final Bayesian network structure. This would also imply the algorithm to become independent of the initial conductivity values and the order in which connections are evaluated to get a more stable structure learner. An implicit advantage is also the reduction in execution time as an ensemble of learners would not be needed any more if the algorithm can be stabilized to converge to a global maximum. Also, execution time further decreases as with increasing number of iterations, more and more connections are converging to zero and the number of connections to be evaluated for score are converging to the final number of Bayesian network connections. In line with that, the impact of the *MFS-Physarum Solver* on conductivity evolution and the interplay between *MFS-Physarum Solver* and score feedback has to be studied more carefully to get a deeper understanding in how connections can be forced to converge to the maxima and minima.

Parameters of the novel SO-PhyL algorithm have been studied individually and it could be seen that learning quality is strongly dependent on a valid parametrization of SO-PhyL which is in line with observations made by Tero *et. al.* who reported the use of different parameters for the *MFS-Physarum Solver* in his experiments. Especially parameter I_0 has been shown to be relevant for the number of connections learned by SO-PhyL where networks of larger size require higher values of I_0 in order to perform comparable to other state of the art structure learning methods. It has been further shown that with increasing ensemble size and increasing *MFS-Physarum Solver* iterations, the chance to find a higher scoring network increases. On the other hand, it can not be excluded that this may be a result of lucky punches. But for larger networks, experiments indicated that SO-PhyL needs more iterations to optimize the score and an ensemble of more learners can push the score, too. Both, increase of ensemble size and *MFS-Physarum Solver* iteration results in linear growing execution time and is probably not feasible for larger networks keeping in mind that benefit in score is not very strong.

Next, SO-PhyL has been applied to a set of artificially generated and real benchmark networks where SO-PhyL showed comparable learning performance to state of the art learning methods LAGD and Tabu Search and better performance than Simulated Annealing for most datasets. Especially for smaller datasets, SO-PhyL learned a score that is equal or very close to the best score retrieved by LAGD in most cases. For larger datasets, a disadvantage can be observed for SO-PhyL which is probably owed to less *MFS-Physarum Solver* iterations or a too small ensemble size. Nevertheless, SO-PhyL could not just show adequate results for nearly all networks, but also outperformed LAGD and Tabu Search clearly for the *Insurance* network. It can be observed in general, that configuration SO-PhyL-1 tends to learn less arcs than other methods which is one reason for excellent performance for the *Insurance* network. While other methods learned too many arcs limiting score, SO-PhyLs score benefit is supposed to be a result of learning less extra arcs than other learners.

To summarize this chapter, the novel SO-PhyL algorithm has been introduced, studies carefully and has been shown to be a competitive structure learning algorithm for Bayesian network that is able to outperform state of the art learning methods for some networks.

CHAPTER 5

Non-alcoholic Fatty Liver Disease (NAFLD) dataset

Finally, C-PhyL and SO-PhyL performances are evaluated on a real world novel medical dataset retrieved at the Medical University of Graz (MUG) from 29 patients with liver-biopsy confirmed non-alcoholic fatty liver disease (NAFLD) [26, 171, 21, 48, 162, 74]. This work is partially financed by the Austrian BioPersMed project (COMET K-project 825329) having the goal to find novel biomarker candidates for NAFLD. Thus, biomedical data has been retrieved from patients at the MUG and data is evaluated to get new insight into interconnections related to NAFLD and to derive possible biomarker candidates which are next verified by medical experts in the laboratories of the MUG. Please note that the NAFLD dataset is not public and can thus not be referenced or be presented within this thesis. From the dataset, the parameters present in less than 50% of the patients and parameters with a constant value over all patients have been removed. The missing values of the remaining 32 attributes have been inserted by modes and means using the built in `ReplaceMissingValues` filter of Weka. Further, the dataset has been extended from 29 to 2900 instances by bootstrapping. As the dataset contains a mixture of real, nominal and ordinal attributes, and Physarum learners require categorical attributes, all attributes have been grouped into categories by using Wekas `Discretize` filter with parameters `findNumBins` and `ignoreClass` set true.

First, the dataset has been searched for most discriminative parameters by combining it with another dataset of 108 healthy controls, also determined at the MUG. As measured clinical parameters differ between NAFLD dataset and controls, the parameter set reduces to 29 parameters that are shared in both datasets. A `CFSSubsetEvaluation` feature selection implemented in Weka has been performed in order to identify the five most important parameters to differ between NAFLD and controls instances. The algorithm delivered five parameters:

- Magnesium (Mg)
- Aspartate transaminase (AST)
- Alanine aminotranferease (ALT)

- Gamma-glutamyl transferase (GGT)
- Age

While AST, ALT and GGT are liver enzymes with known relation to NAFLD [165], parameter Mg has not been studied in detail as biomarker candidate for NAFLD until now. Please note that the age of patients suffering from NAFLD is also important, but it is not clear if selection of feature `Age` results from artefacts in data retrieval, normalisation steps or ideally is also a valid biomarker candidate. This question is forwarded to be answered by medical experts of the MUG. The quality of the five observed biomarker candidates with respect to their ability of differentiating between NAFLD patients and controls is validated by reducing the dataset to only these five parameters and measuring the classification accuracy. A `RandomForest` classification with 50 trees using 10-fold cross validation has been performed on a bootstrapped version of the dataset, where instances could be classified correctly with an accuracy of 98%. This results means that a previously unseen patient can be classified as NAFLD patient or healthy control by knowing only the five supposed parameters with an accuracy of 98%. As a result of this data analysis and the high classification accuracy, the novel investigation of promising biomarker candidate Magnesium is further studied by building a mouse model supervised by medical experts at the MUG in order to clinically verifying the novel observations. Unfortunately, performing a mouse model is a time consuming task and results are not expected to be available by the end of this dissertation.

5.1 Learning structure of NAFLD dataset

Next, C-PhyL is used with the same configuration as for learning benchmark networks to determine the network structure of the NAFLD dataset to indicate interesting relations between parameters. Please note, that healthy controls are not considered in this studies in order to investigate relations specific for NAFLD patients only. The learned network includes 60 arcs and is shown in Figure 5.1.

Also, configuration SO-PhyL-1 is used with $I_0 = 5.0$ to learn a more spare network increasing believe of learned connections and a maximum number of three parents per node to learn the network structure of the NAFLD dataset. The resulting network structure contains 79 arcs and is presented in Figure 5.2. It can be seen from Figures 5.1 and 5.2 that nodes `Total Cholesterol` and `Thrombocytes` have many connections in both graphs. It is further clear to see, that both networks follow the scale free topology which is typically for biological and medical datasets, where few nodes exist having many connections and the other nodes are connected by less arcs. Further, it is observed that obviously true connections as for example `HDL Cholesterol` $\leftrightarrow$ `Total Cholesterol` and `LDL Cholesterol` $\leftrightarrow$ `Total Cholesterol` are present in both networks. Nevertheless, a detailed analysis of connections is needed, performed by medical experts in order to validate the learned networks of the NAFLD dataset. The final evaluation of connections is performed by medical staff at the MUG and has not yet been started due to delays in project progress. For that reason, no further investigations and appraisees of learned network structures can be provided at this point.

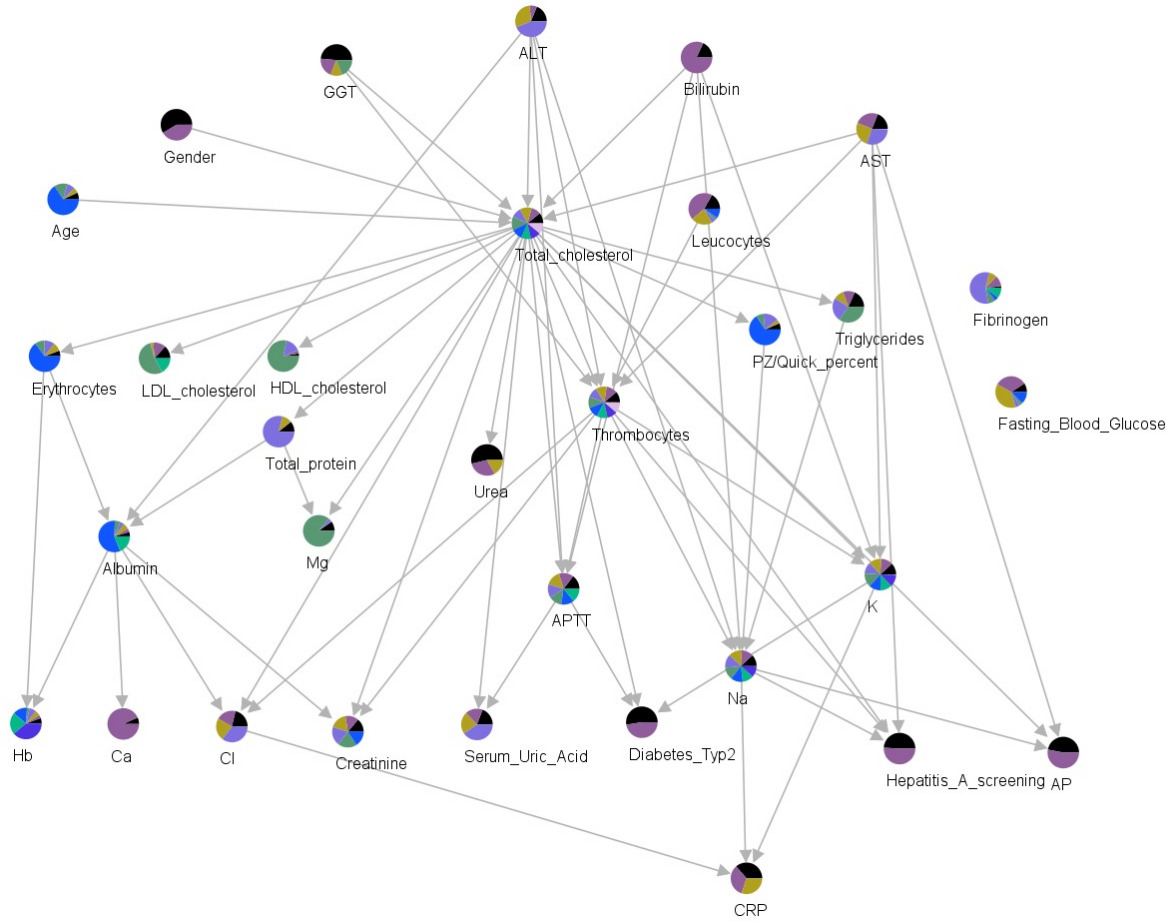


Figure 5.1: Learned network structure of NALFD dataset using C-PhyL.

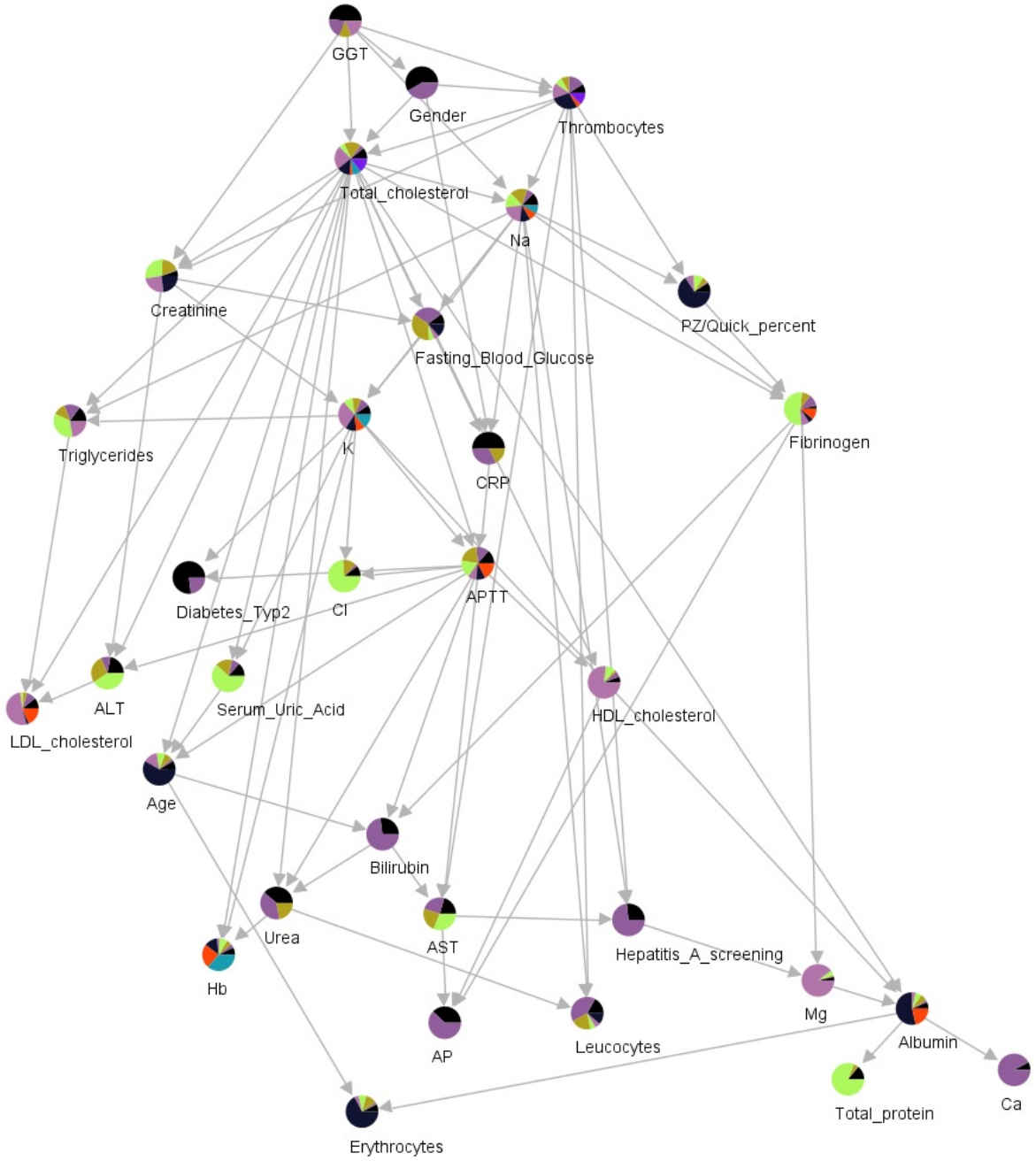


Figure 5.2: Learned network structure of NALFD dataset using SO-PhyL.

CHAPTER 6

Discussion

Experiments presented in this thesis showed that the concept of *Physarum polycephalum* can be successfully transferred from a path finding problem to the problem of learning Bayesian network structure from data in two different ways, although some modifications remain to be made. The first novel algorithm presented in this thesis called C-PhyL tackles the problem by building a Physarum-Maze where each parameter in the dataset becomes a node and by modelling length of connection between two nodes in the Physarum-Maze by calculating a correlation coefficient between the two parameters using the dataset. By calculation of pairwise correlation coefficients, higher order correlations can not be considered. For example if one parameter is only dependent on another parameter if a third parameter has a specific value, this cannot be modelled correctly by two dimensional correlation coefficients like the used Cramérs V coefficient. A Bayesian network on the other hand, models the distribution of a dataset by deriving marginal and conditional probabilities of higher orders. For example if a node has two parents, the node is dependent on values of both parents implying a higher order correlation between these three nodes. The C-PhyL algorithm, by definition is not able to calculate these kind of correlations, although it would be mathematically possible. But, how to define the length between two nodes in the Physarum-Maze if the correlation has been calculated using more than two nodes? It would be required to determine the impact of these two nodes to the higher correlation in order to define an appropriate distance. Hence, C-PhyL tries to model dependencies of a node on a group of other nodes by finding the shortest alternative path between two nodes assuming that the direct path is not available. In other words, a set of pairwise correlations is searched that explains the missing direct pairwise correlation best. Obviously, this is not very accurate as the interplay between a set of nodes can also not be considered in this way. Another problem occurs for nodes that would be connected only to one other node in the true network with respect to underlying distribution of the dataset. Assuming the two nodes of the correct connections are set as food sources and the connection between them is removed, *Physarum Solver* is forced to let an obviously wrong connection survive which is hence increased in its rank and becomes more likely to be part of the final Bayesian network. Knowing these issues, question

derives why C-PhyL is able to learn relatively good network structures. First, C-PhyL is observed to learn much less arcs compared to other learning methods which may be a result of the fact that strong connections are better represented by pairwise correlations and the more complex connections that are correlated by higher orders cannot be found by C-PhyL good enough. Further, from example network *Insurance* shown in Figure 3.5 can be seen that C-PhyL tends to learn also connections that are already modelled by a diverging connection or by a partial sequence of connections. This is also an indication that C-PhyL misses conditional distributions that are defined over more than one parent node. A possible way to clarify how well C-PhyL can model higher order correlations by finding paths over pairwise correlations that try to explain it, is to replace the ranked list of connections by a list where connections are ordered by their pairwise correlations. Performing these experiments, comparing the list generated by *Physarum Solver* and the correlation list and analysing results of learned network structures is referred to as future work.

Another crucial part of C-PhyL is how to transform the ranked list of connections to a valid Bayesian network. First, connections are undirected while a Bayesian network requires directed connections and second, adding a connection can hurt formal restrictions of Bayesian network by either causing a cycle or exceeding maximal number of parents for a node. Connections are thus only added if the resulting Bayesian network stays in an allowed state. Further, connections are also added only if they are increasing the score of the Bayesian network. It was initially not intended to incorporate score calculation at all into C-PhyL as the ranked list of connections was supposed to deliver connections in the right order. Adding a score based restriction is withdrawing responsibility from the list of ranks and expands execution time. Nevertheless, experiments showed that the list of ranked connections alone is not good enough to build competitive network structures. Also, without using score calculations, it is very hard to find an appropriate threshold on how to stop adding connections from the ranked list into the final Bayesian network.

Probably the most critical part of C-PhyL and hence also the part limiting score is how to determine directions of the Bayesian network as the *Physarum Solver* does not consider directions at all. In experiments presented in this thesis, a naive ordering based approach is used and has been shown to be improper. As already mentioned in Section 3.4, there are some other possibilities to determine directions, for example by using further developed ordering methods, calculating the score in both directions and picking the direction with higher score or by incorporating Kirchoff's second law into the *Physarum Solver*. On the other hand, C-PhyL as presented in this work has been shown to be a structure learner for undirected graphical models. It follows, that C-PhyL may also be considered for learning undirected graphical models like Markov networks instead of learning Bayesian networks or to learn only a skeleton of the network structure. As described in material chapter, a Bayesian network can be represented by a set of score equivalent graphs that have to share the same skeleton, meaning the same d-separation and colliders. Another indicator that this is a promising direction for future work is the above mentioned observation of clique like structures in the *Insurance* network learned by C-PhyL. One possible way to learn undirected graphical models is to update only Algorithm 3 of C-PhyL so that connections are added to a graph ordered by their rank instead of evaluating a score. It remains the problem of when to stop adding connections to the graph. A possible threshold is the average value of ranks, a maximum number of

cliques that should be present in the Markov model or a measurement of how crowded the final network is allowed to be.

Another idea of future directions is to use C-PhyL as a feature selection algorithm. This can either be done by counting how often a node is present in the top connections of the ranked list or by counting how often a node is part of a surviving path directly within the *Physarum Solver*. Result is a list of ranks for nodes and thus also for parameters in the dataset from which the highest ranked nodes can be selected.

Despite drawbacks and problems described so far, C-PhyL has been shown to learn adequate network structures and to also have essential advantages compared to other methods. Experiments with artificial and real benchmark networks showed that C-PhyL learns network structures with less extra arcs compared to other state of the art learning algorithms. Learning less arcs from which are most correct, indicates that C-PhyL reaches high specificity of arcs and that arcs learned by C-PhyL can be trusted more. Hence, C-PhyL is suitable for learning network structures where the goal is to find interesting relations between nodes. Generally speaking, there are two main objectives why a Bayesian network is learned from a dataset. The first is to model the underlying distribution of the dataset best in order to do for example classification or prediction. For this task, the number of arcs and if they exist in the correct network structure is less important. Everything needed is a structure that models the distribution best which is indicated by a high Bayesian score. The other main objective is to learn a Bayesian network structure in order to get correct arcs only, resulting in more sparse networks. Here, goal is to get insight into data, to identify which parameters are dependent on which other parameters in order to get a deeper understanding of the problem domain under study. As C-PhyL learns sparse networks with very less extra arcs, it is especially appropriate for the second task, namely modelling a network structure that offers insight into relations between parameters.

Another application area for C-PhyL can be found for datasets with less nodes but a huge number of instances. An example of these datasets can be streaming data, user behaviour data for example retrieved for mobile phone calls. As C-PhyL has to touch the dataset only once at initially preparing pairwise correlations, C-PhyL is at least theoretically much faster than search and score based structure learning algorithms which have to touch the dataset many times for calculating scores. Hence, once a time and memory efficient implementation of C-PhyL exists, C-PhyL can offer a big advantage in execution time for these family of datasets.

Also, score based algorithms have to be restricted in the number of maximal parents per node as calculation of scores requires to build all possible combinations of parent parameters which is a combinatorial problem that becomes infeasible rapidly for larger networks. C-PhyL does not need to consider these combinations as only pairwise correlations are calculated. Hence, C-PhyL does not require to be restricted in in-degree for learning the structure of a Bayesian network. Note, if a complete Bayesian network including also conditional probability distributions is to be learned, a maximal number of parents may also be needed in order to determine the CPTs, but this is done independently from C-PhyL. There have been also methods proposed that do not need to calculate CPTs, but model conditional probability distributions by Gaussian functions or by using classifiers for each node [114]. Using these methods, C-PhyL shows another big advantage compared to other methods described in this thesis. By changing Cramérs

V correlation coefficient to the common Pearson product-moment correlation coefficient, C-PhyL is also able to learn networks form continuous data. As ranges and meanings of correlation coefficients can be normalized to be in the same range, C-PhyL is even appropriate for learning structures form datasets with mixed parameter types. The only adaptation that has to be made to C-PhyL is to not use score calculations for building the network from the ranked list of connections, but adding connections simply in the order of their rank until rank falls below average rank value.

Moreover, the C-PhyL algorithm can be easily parallelised which is a crucial task for modern algorithms as more and more applications are implemented cloud based, where processes are distributed over a large number of processors. Even if the algorithm is run on a local machine, parallelisation can be beneficial as modern computers usually have several processor cores available. C-PhyL runs the *Physarum Solver* for each node pair independently of previous results and reports surviving connections to a shared list of ranks. Each of these *Physarum Solver*s can thus run on an extra thread in parallel and surviving connections are ranked thread safe. Especially for networks of larger node size, a parallelised C-PhyL algorithm can have enormous speed benefits compared to a single threaded version.

In contrast to the C-PhyL version published in resulting papers of this work [172, 173], C-PhyL presented in this thesis does not have any extra restrictions to the dataset where the structure should be learned from. The C-PhyL algorithm can thus be applied to any dataset that can be used with LAGD, Tabu Search and other related structure learning algorithms. This statement is also valid for the second novel algorithm called SO-PhyL, introduced in this thesis which also requires a categorical and complete dataset, only.

SO-PhyL is an advancement of the C-PhyL algorithm that tries to overcome the lack of considering correlations of higher orders in C-PhyL by incorporating score evaluations into the learning process. During the development process, it turned out that the concept of using pairwise correlations is not appropriate for integrating score calculations and hence, the SO-PhyL algorithm is based on a different concept than C-PhyL is. Length values of the Physarum-Maze are set constant and all *Physarum Solver* dynamics are modelled by changes in conductivities only. Also, the updated version of the *Physarum Solver* introduced by Tero *et. al.* and called *MFS-Physarum Solver* in this thesis is used. Thus, searching the shortest path between all possible node pairs is also not appropriate any longer. Instead, the *MFS-Physarum Solver* runs over a number of iterations until unimportant connections have disappeared. In each of these iterations, after conductivity changes caused by *MFS-Physarum Solver* dynamics have been applied, a Bayesian network is built out of connections with high conductivity values and these connections are reviewed by their contribution to the global score of the Bayesian network. If adding a connection increases the score, the connections' conductivity value is further pushed. If, on the other hand, an extra arc would cause score decrease, conductivity of the arc is lowered. In other words, feedback based on score calculations is given to the conductivity values of the connections. A strategy similar to greedy hill climbing is applied, where *MFS-Physarum Solver* is used to preselect only a subset of connections in each iteration that has to be evaluated for score instead of considering all available arcs.

Experiments showed that for most datasets, score increases with increasing size of subset of connections. It is thus unclear how strong the impact of selecting connections by *MFS-Physarum Solver* is on the learning result. The interplay between *Physarum*

polycephalum strategy to further push connections that are already in a winning position and the impact of score feedback has to be studied in more detail in future work as already discussed in the concluding section of Chapter 4. In an ideal scenario, all connections are converging to either the maximal or the minimal conductivity threshold. But, analysis of conductivity evolutions for connections of the *Asia* dataset showed that in practice, only some connections converge properly. Evolution of connections is also strongly dependent on parameters used to learn SO-PhyL. When changing parameters, it could be seen from preliminary tests, that it is possible to force all conductivities to finally converge but not do guide all connections to converge to the correct direction. This means that some connections which are important for the Bayesian network are converging to the lower limit and thus that the distribution of the dataset can not be modelled good enough ending in a worse score. For further development of SO-PhyL, probably the most important task is to better understand the interplay between *MFS-Physarum Solver* and score feedback to get good evolution of conductivity values. Ideally, connections are converging so that a global maximum is reached, which would also avoid the need of an ensemble of learners and diminish the impact of random initial conductivity values.

It has been mentioned already that there are several configuration parameters that strongly influence the behaviour of SO-PhyL. Three different parameter configurations have been tested for learning network structures of artificial and real benchmark networks. SO-PhyL-1 was shown to be faster than the other two settings as less connections have to be evaluated for score in each *MFS-Physarum Solver* iterations. But hence, also less connections are learned in the final Bayesian network. It can thus be concluded that if a more sparse network is desired for example to study relations between data parameters, it is recommended to use configuration SO-PhyL-1. It further turned out that SO-PhyL-1 outperformed all other methods for the *Insurance* network. On the other side, for almost all other methods, parameter settings two and three where SO-PhyL-2 considers more connections for score evaluation and SO-PhyL-3 used more iterations and a bigger ensemble size, could learn network structures with higher score than configuration SO-PhyL-1. But, obviously these two settings do need more execution time. It is a direction for future work to further analyse parameter configurations in order to possibly find a rule on what datasets which parameters are best to use. Until such rules are found, parameters may have to be optimised and fitted for each dataset individually to reach optimal learning performance. Doing so, there is a high risk of overfitting SO-PhyL to specific datasets. Overfitting can for example be avoided by restricting the number of iterations as can be seen in presented graphics showing evolution of score over iterations. It has been shown that learning is performed at most in the first few iterations and further improve in score can not be excluded to be caused by lucky random choices. It is hence recommended to use a small number for parameter r when overfitting has to be avoided.

Analysing results of SO-PhyL for both, artificial and real benchmark networks, it can be observed that SO-PhyL learns more reversed arcs in relation to the number of correctly directed arcs than other competitive methods. Direction of connections is determined while adding the connections from the Physarum-Maze to the intermediate Bayesian network in each direction by calculating the score impact of each connection in either direction. Hence, this procedure is rather promising a valid assignment of direction. Nevertheless, results show that the strategy is not working competitive and has to be further improved for future versions of SO-PhyL.

All experiments presented in this thesis have been investigated using the Bayesian score. As reported in the material chapter, there is a group of scoring functions for Bayesian networks including also the BDeu, MDL and AIC score. A possible direction for future work is to also compare learning quality of SO-PhyL to other state of the art learning methods regarding these other scoring metrics.

Despite reported issues, SO-PhyL has been shown to be a competitive structure learning method for Bayesian networks that is able to outperform state of the art algorithms in learned score, number of correctly learned arcs and the number of extra arcs for some networks. Experiments included in this thesis showed that SO-PhyL outperformed other methods for the *Insurance* benchmark dataset. This dataset does not have any specific characteristics that differs from other benchmark datasets. Hence, it can not be concluded that SO-PhyL might be especially good for specific network characteristics. However, it is observed that SO-PhyL performs excellent for networks with a small number of nodes and gets worse with increasing number of nodes. As already described, this is probably caused by using too few iterations to optimize the score or by using not enough learners within the ensemble.

It has been shown that setting parameters of SO-PhyL is a crucial and often hard task, but high influence on parameter can also be seen as a benefit. The algorithm can be used to learn different kinds of network structures, for example a more sparse network by using a small value of I_0 and a high value of $D_{\tau_{end}}$ if the desired network structure is to be used for determining interesting relations within variables of the dataset. On the other hand, if not the correct or true structure of the network is important, but that the underlying distribution of the dataset is represented correctly, a more crowded network structure can be learned by adjusting configuration parameters respectively. SO-PhyL is therefore more flexible to fit the different needs of learning Bayesian networks from data.

Compared to C-PhyL, the SO-PhyL algorithm itself can not be easily distributed over several processors without severe changes. But, as methods described in this thesis are using an ensemble of SO-PhyLs, parallelisation is possible by distributing the single SO-PhyLs of the ensemble. When using a cloud based environment where a theoretically unlimited amount of cores is available, speed and execution time of SO-PhyL can be improved by at most the factor of the ensemble size. For example, running an experiment with an ensemble size of 10 learners is theoretically ten times faster if all learners are executed in parallel than performing them sequentially. Thus, also SO-PhyL can benefit from modern architectures and can be used efficiently with them.

Comparing C-PhyL and SO-PhyL, it has to be noted first that both algorithms have to be optimized for speed and memory usage in order to be competitive in execution time with other state of the art learning methods. Further, a more detailed and theoretical analysis of complexity of both algorithms is needed and is a task for future work. With respect to the quality of the learned network structure, both algorithms have been shown to learn adequate networks, where structures learned by SO-PhyL are more competitive to other methods than these learned by C-PhyL. In contrast, C-PhyL learned structures with less extra arcs where only the most important arcs are present. Thus, the choice of using either C-PhyL or SO-PhyL depends on the desired goal. If goal is to get insight into a dataset, C-PhyL is probably the better choice. But if a network structure is to be learned that is representing the underlying probability distribution of the dataset best, SO-PhyL should be used.

For both algorithms, further experiments have to be made in order to test learning robustness for datasets including noise and missing values. In this thesis, no tests have been made in how Physarum inspired algorithms perform for datasets with a high noise level or with a big amount of missing values compared to other structure learning algorithms for Bayesian network.

Another possible improvement is to combine the benefits of both algorithms. This can be done for example by using pairwise correlations to replace the constant values of lengths in SO-PhyL. As a result, initial random settings of conductivity values will not have that much impact and the algorithm keeps some kind of ground truth by length values while changing conductivities. Although, this process comes with the problem that the pairwise correlations can not be adjusted within the algorithm and are thus not flexible to overcome local maxima or to revert initially made mistakes. A more promising way might be to use pairwise correlations to initialise conductivity values of the connections instead of picking them randomly. More time and experiments are needed to test if this leads to a benefit in learning performance and if the time that is needed to calculate correlations at the beginning is worth to be spent.

The initial attempt of the project financing this thesis was to find biomarker candidates for NAFLD by studying Bayesian network structures learned from established data. Because of delays in data retrieval from the medical project partner and a rather random meeting with the *Physarum Solver*, the topic of this thesis was redefined to investigate if the *Physarum Solver* can be used to learn Bayesian network structure from data. Although, a small dataset has been delivered containing common clinical parameters and additional parameters specific for patients suffering from liver diseases. The dataset delivered contains 29 liver biopsy confirmed NAFLD patients which is to the best of our knowledge the biggest NAFLD dataset available. In this set, data instances suffer from many missing values leading to a finally cleaned dataset of 32 clinical parameters. While combining the dataset with data from healthy controls, a feature selection algorithm has been applied and five parameters could be selected. With only these parameters, a Random Forest classification has been performed with 98% classification accuracy. In addition to the parameters with already known relation to NAFLD, Magnesium could be identified as possible novel biomarker candidate and is currently analysed by a mouse model at the MUG. Hence, the initial goal of supposing new biomarker candidates by analysing data has been achieved. In addition, the two novel structure learning algorithms have been used to learn network structures from the dataset in order to give new insight into relations important for NAFLD. It can be seen from Figures 5.1 and 5.2 that a lot of arcs have been learned compared to benchmark networks of similar node size. A reason for than might be the large number of missing values that had to be estimated and from heavy bootstrapping or from characteristics of scale free networks. But, structures show obviously correct connections and have been forwarded to the MUG with promising anticipations. Unfortunately, analysis of these network structures is time consuming and can thus not be included into this thesis.

The intention of this thesis was to investigate if the *Physarum Solver* previously mostly used for shortest path finding problems can be successfully adapted to the problem of learning a Bayesian network structure from data. It has been demonstrated that it is not only possible, but that the *Physarum Solver* based novel algorithm can also outperform commonly used state of the art structure learning methods for specific datasets. For most

of datasets analysed in this thesis, the LAGD structure learning algorithm has learned the highest scoring network structure and has thus been demonstrated to be an excellent learner. SO-PhyL has been shown to learn network structures of a score near to the one learned by LAGD for almost all datasets and has thus be demonstrated to be a strong competitor. Further, SO-PhyL outperformed Tabu Search in some datasets and learned better scores as Simulated Annealing for most networks.

Possible directions for future work have already been discussed in the previous paragraphs regarding improvements to both Physarum based algorithms. But beside algorithmic updates to resolve drawbacks of C-PhyL and SO-PhyL, there are many other possible fields of application for the novel algorithms. In this thesis for example, no experiments have been performed for studying the usage of Bayesian networks learned by C-PhyL and SO-PhyL for classification tasks. When using Bayesian networks for classification, the structure and hence the correctness of individual arcs of the structure is not that important. The important task for classification is to represent the distribution correctly no matter how the learned structure looks like. Hence, the C-PhyL algorithm which benefits from learning a more sparse network including only arcs with highest probability to be correct is not convenient to be used to learn networks in order to perform classification tasks. SO-PhyL instead has been shown to learn high scoring networks and to model the datasets marginal and conditional distributions appropriately. Therefore, a detailed study is needed in future experiments to analyse how SO-PhyL performs for learning Bayesian networks used for classification.

One more task for upcoming experiments with C-PhyL and SO-PhyL is to test the algorithms on a broader set of real benchmark networks and on datasets determined from different problem domains. SO-PhyL has been shown to perform especially well for the *Insurance* network belonging to the domain of financial or business data. Thus, it may be assumed that business data is of a structure that is more suitable to the benefits of SO-PhyL. A further study with more business data is needed to investigate why SO-PhyL performed best for precisely the *Insurance* network. Another problem domain of high interest in the last years is originated from the field of bioinformatics where a lot of gene expression data is generated and needed to be analysed. Benchmark networks used in this thesis did not include any gene expression dataset. These kind of data is characterized by a huge amount of parameters and a relatively small amount of instances, which is exactly the opposite of benefits observed for C-PhyL and SO-PhyL algorithms. It can thus be supposed, that applying the novel algorithms on these datasets is not to the best advantage. But, if applying a proper feature selection algorithm identifying the most important genes and reducing the dataset to them, Physarum algorithms are applicable and can be compared to other state of the art learning methods.

In the introduction if this thesis, methods of *Physarum polycephalum* are compared to techniques described by the ant algorithm and a publication is referenced where the ant optimization algorithm has been successfully used to learn network structures of Bayesian networks [64]. As both methods are related in the way they use bio-inspired approaches of finding a shortest path adapted to an optimization problem, it remains to briefly discuss the similarities and differences of both methods. Comparing the ant algorithm to C-PhyL, it is easy to observe that both methods are using totally different approaches. The ant algorithm is using score optimization in order to traverse the space of possible network structures while C-PhyL does not build a search space of network structure at all. On the

other hand, methods introduced with SO-PhyL do have similarities with the ones used by the ant algorithm. Both methods are using their individual bio-inspired techniques to search the space of possible network structure while optimizing a score measuring the fit of distribution modelled by the current structure to distribution observed in the dataset. But the way in which the search space is traversed is different. The ant algorithm uses individual ants that are travelling around in the search space and are changing the pheromone values initially applied to each connection. In each iteration, an ant can update the pheromone value of connections and connections are chosen with probability dependent on their pheromone value to be considered as next greedy hill climbing step. Comparing this method to SO-PhyL, attaching pheromone concentrations to connections is equivalent to applying conductivity values to connections and both are internally using greedy hill climbing. The main difference between these methods is that SO-PhyL uses the *MFS-Physarum Solver* to define a subset of connections that are considered for greedy hill climbing in each iteration in contrast to using probabilities of connections to be considered for greedy hill climbing as done by the ant algorithm. Both approaches have been demonstrated to show good learning performance but unfortunately, although both are evaluated on the *Alarm* and *Insurance* network, different evaluation metrics have been used so that both cannot be compared to each other directly. Comparing the ant algorithm to SO-PhyL by using the same benchmark metrics is an interesting task for further experiments and can give new insight into the difference between these two algorithms.

To be able to easily use C-PhyL and SO-PhyL in other problem domains and to offer effortless access to these algorithms, implementation of both methods have to be updated to implement the interface of Weka's Bayesian network framework. Building classes that can be easily integrated and used with Weka is planned for next steps to ensure accessibility of the novel algorithms to other researchers and get new insight into benefits and drawbacks reported in this thesis. Further, to the best of the authors knowledge, there is no freely available implementation of the *Physarum Solver* or the *MFS-Physarum Solver* in Java so far.

CHAPTER 7

Conclusion

Two novel structure learning algorithms for Bayesian networks inspired by the biological concept of *Physarum polycephalum* have been introduced and compared to several state of the art learning algorithms. Both algorithms showed adequate results where C-PhyL profits from learning less extra arcs than other methods and SO-PhyL is able to learn comparable network structures and was shown to be able to outperform state of the art learning algorithms for specific networks. Detailed conclusions for both algorithms introduced in this thesis are given in individual conclusion sections of corresponding chapters. In addition to the development of two structure learning algorithms, content of this thesis provides new insight into the *Physarum Solver* introduced by Tero *et. al.* and shows how this model can be used and transferred to other domains next to shortest path finding. More, a novel medical dataset retrieved from the Medical University of Graz containing data of patients suffering from Non-alcoholic Fatty Liver Disease has been analysed and Magnesium has been supposed as promising novel biomarker for NAFLD. Further, C-PhyL and SO-PhyL have been used to learn a Bayesian network from this dataset to provide new insights into relations between NALFD related parameters and to help understanding the processes of this disease.

CHAPTER 8

Acknowledgement

First, I would like to thank Karl Rößl, owner and founder of SustSol GmbH where I have been employed while working on my thesis, for giving me this great opportunity. Further, I especially thank Prof. Dr. Martin Stetter from the University of Applied Science Weihenstephan-Triesdorf for supervising me over the last few years and pushing my work forward with helpful comments and explanations. Also, I have to thank Martin for introducing me to Karl and helping me getting the position as researcher at SustSol GmbH.

I am further very thankful that my work has been supervised also by Prof. Dr. Elmar Lang from the University of Regensburg. I really enjoyed discussions with Elmar and my colleges which have not always been only technical. Many thinks I have learned over the last two years are based on the tremendous experience of Elmar.

At this point, I would also like to thank two special people not connected directly with my thesis, but who embossed by scientific career essentially. The first one is Dr. Guy Tsafnat from the University of New South Wales in Sydney who introduced me to scientific work while offering my the outstanding chance to work for a semester in his lab in Sydney. The other one is Dr. Alexey Tsymbal who supervised my diploma thesis at the Corporate Technology section of Siemens AG in Erlangen. Most of my previous knowledge in machine learning and data analysis relies on his experiences which he shared with me while spending time at Siemens.

Further, I am very grateful to got the chance to work with Dr. Karine Sargsyan and Dr. Erika Wichro form the Medical University of Graz. This work was supported by BioPersMed (COMET K-project 825329), which is funded by the Federal Ministry of Transport, Innovation and Technology (BMVIT) and the Federal Ministry of Economics and Labour/the Federal Ministry of Economy, Family and Youth (BMWA/BMWFJ) and the Styrian Business Promotion Agency (SFG).

Last but not least, I would like to thank my family and especially my wonderful wife Maria for supporting me, motivating me and helping me to keep up with my work. I am sure that this thesis would not have been the same without the encouragement of Maria.

To stay in line with the topic of this thesis, influences of different people on myself

and this thesis are also illustrated as a graph shown in Figure 8.1.

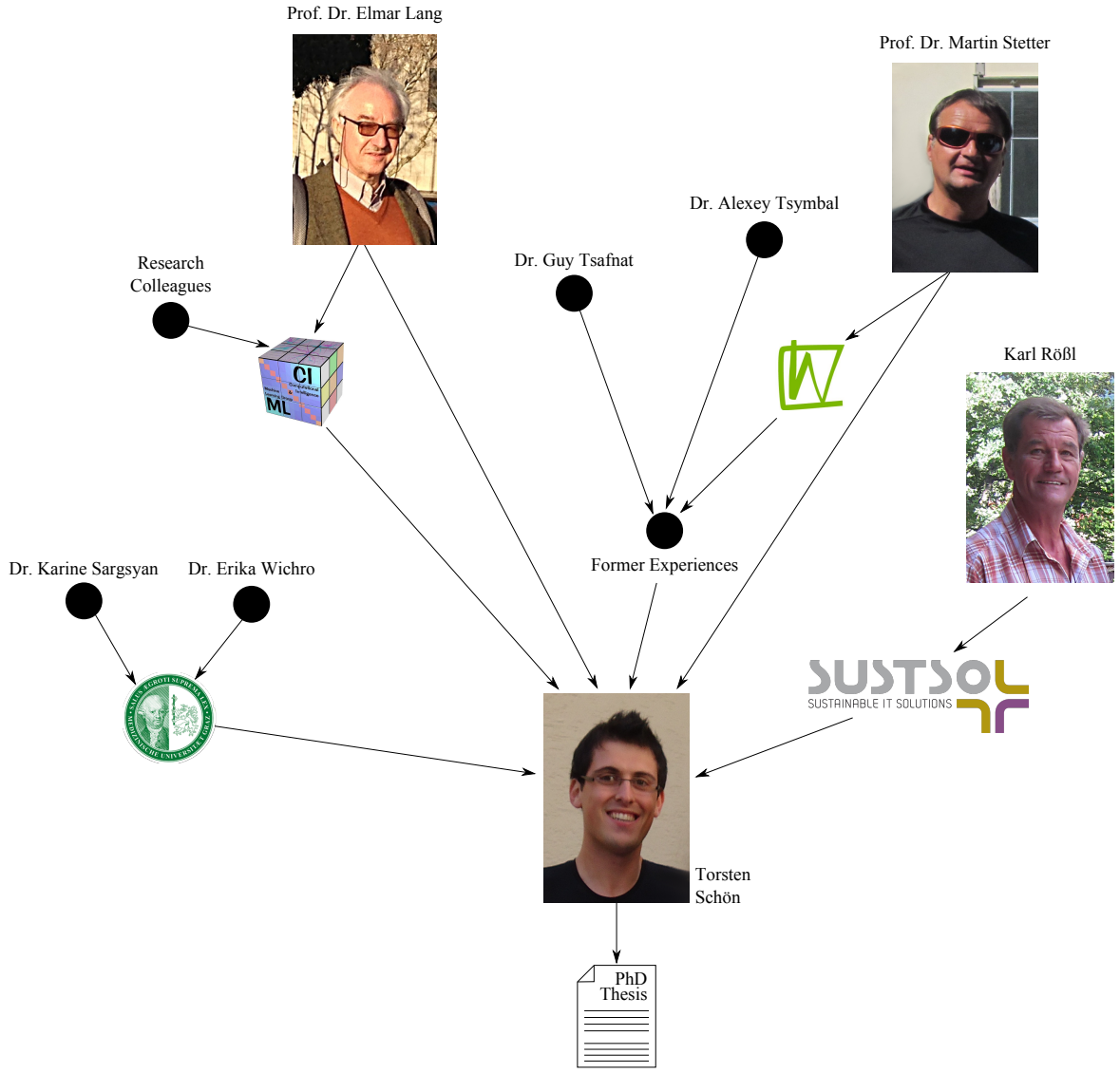


Figure 8.1: Influence of important people on this work

Appendices

APPENDIX A

Benchmark networks

Network structures shown in this chapter have been determined by taking screenshots of the SustSol InSilico Simulator (SISSi) software. SISSi is a Bayesian network viewer and editor that has been developed throughout this thesis. The work of this thesis was partially funded by SustSol GmbH as part of an Austrian K-Project called BioPersMed where one project goal was to develop a software component to be used by medical experts which is able to visualize Bayesian networks.

In SISSi, nodes are illustrated by a pie-chart showing marginal distribution of the nodes variables in different colors. For example, binary node *Smoker* in Figure A.1 can either be *True* (black) or *False* (purple) where the probability to be smoker is 30% and thus, the pie chart is 30% black.

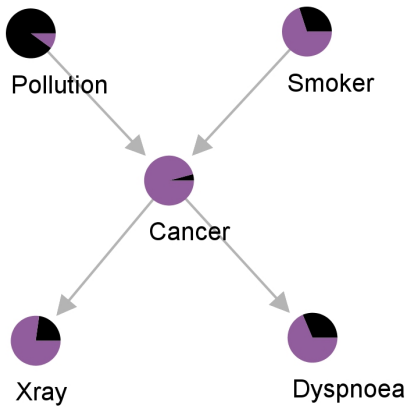


Figure A.1: Structure of the Cancer network printed with SISSi.

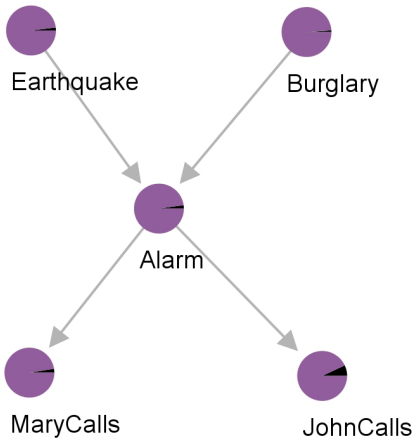


Figure A.2: Structure of the Earthquake network printed with SISSi.

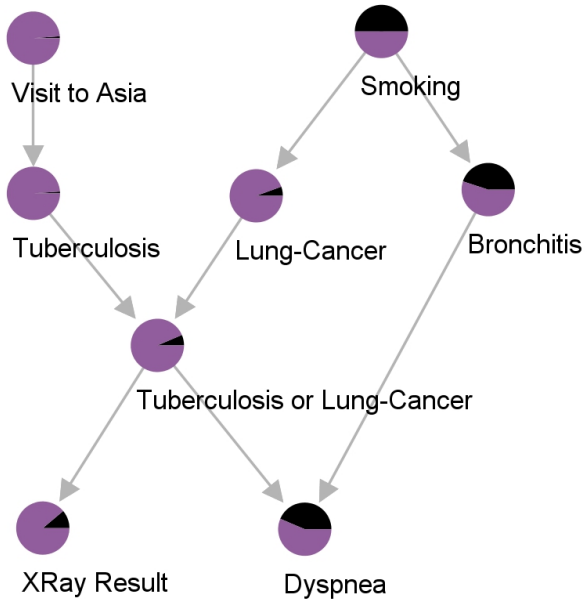


Figure A.3: Structure of the Asia network printed with SISSi.

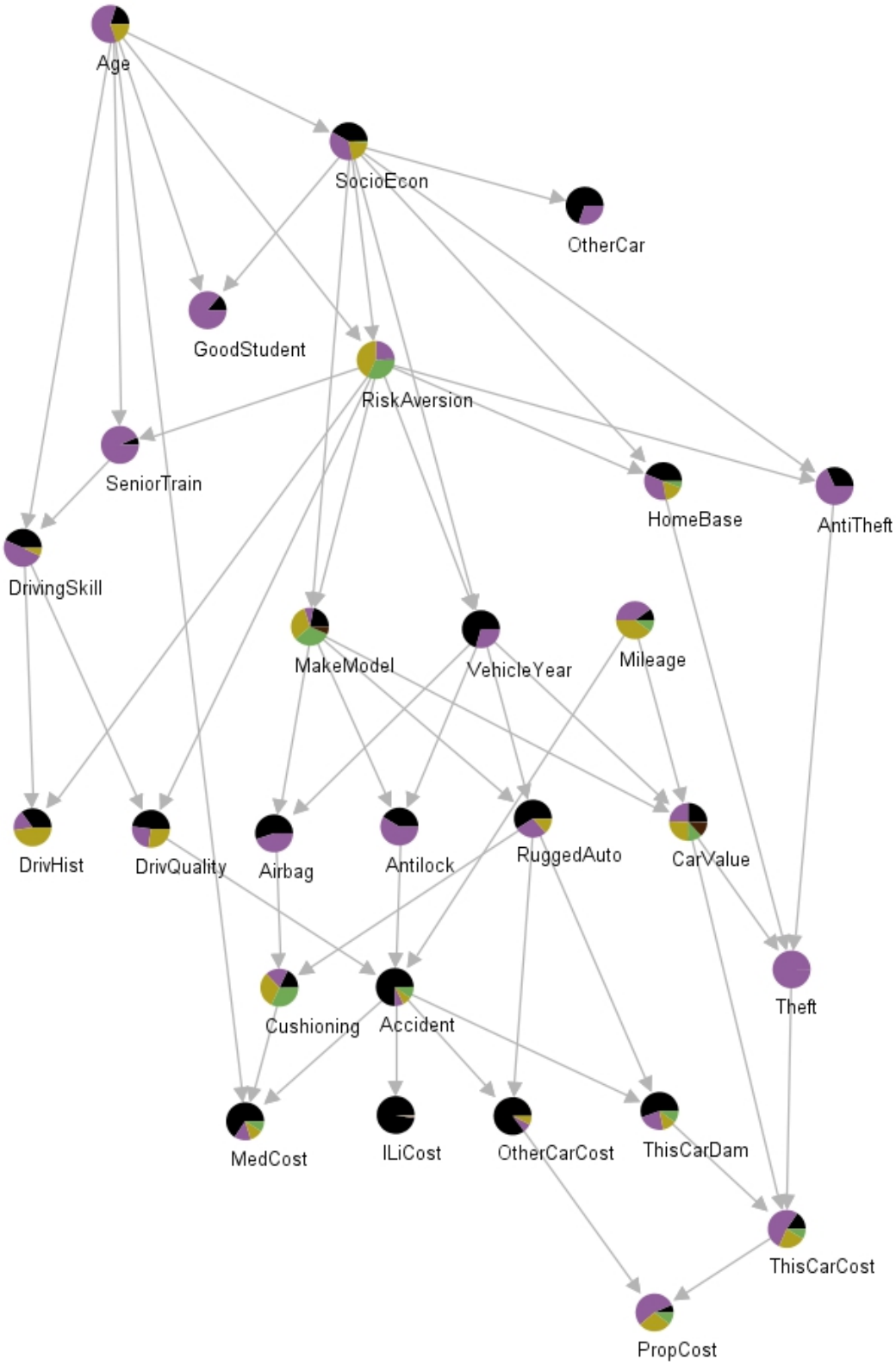


Figure A.4: Structure of the Insurance network printed with SISSi.

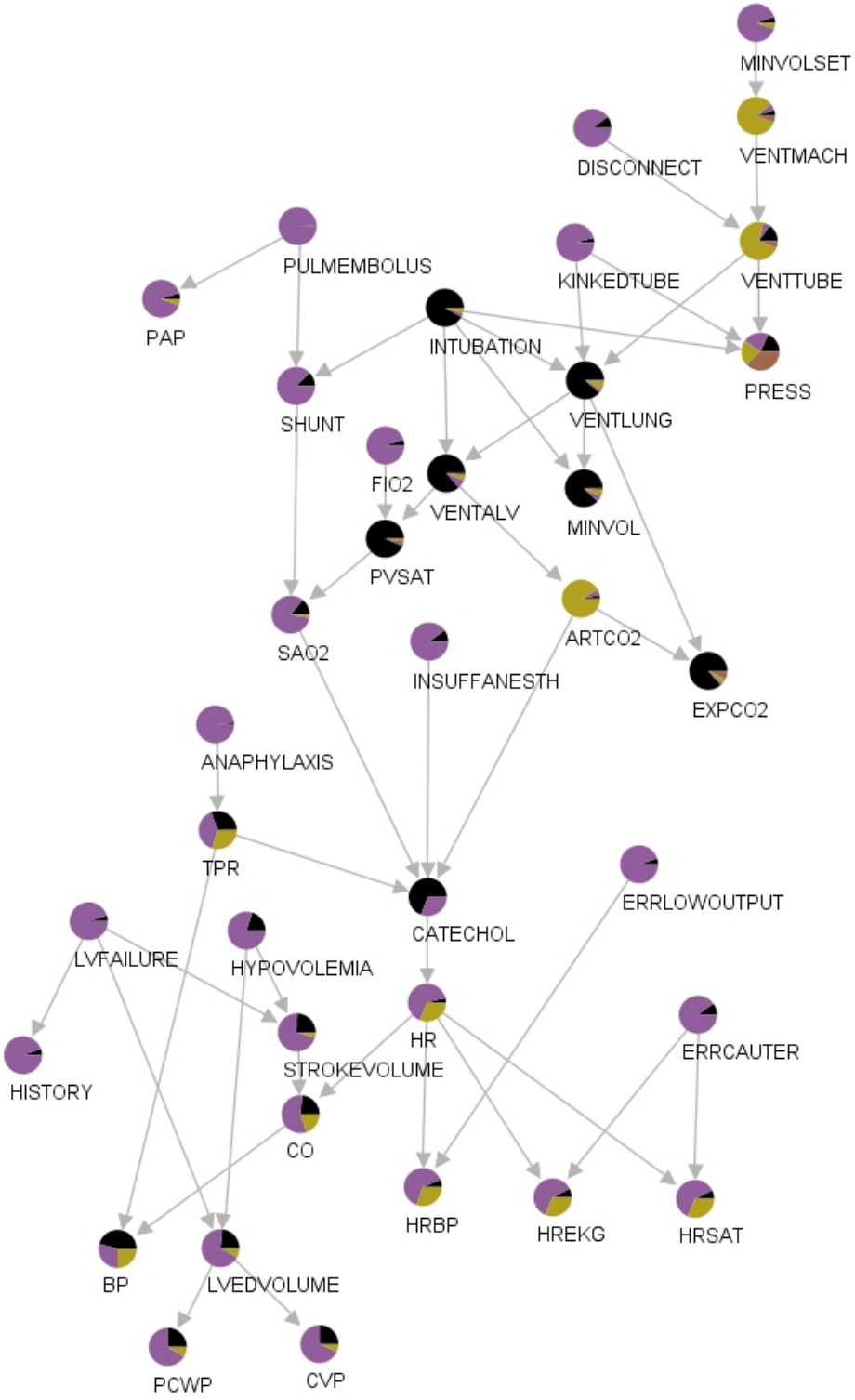


Figure A.5: Structure of the Alarm network printed with SISSi.

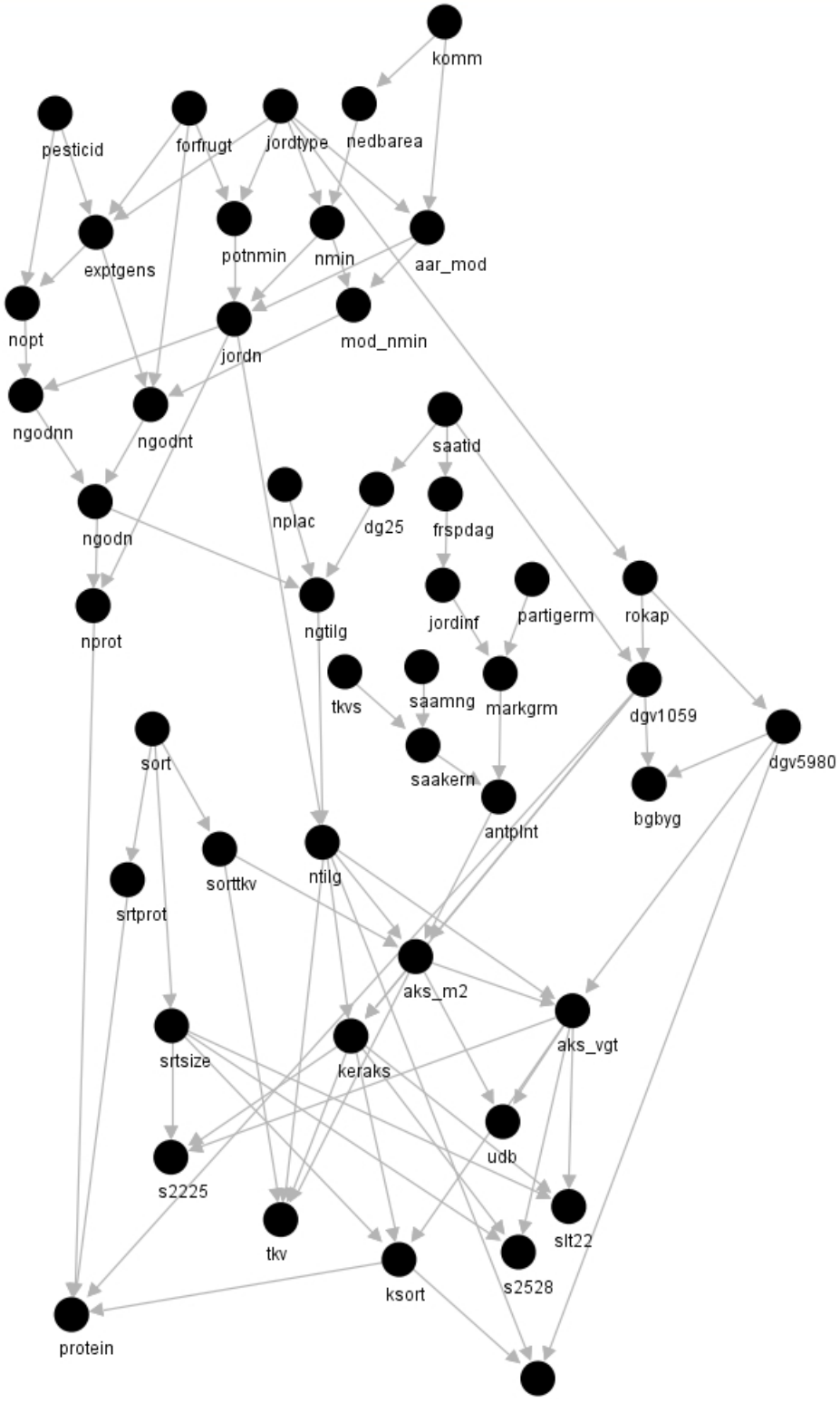


Figure A.6: Structure of the Barley network printed with SISSi. Note that nodes are drawn black as due to the large amount of parameter configurations, SISSi is unable to determine and illustrate marginal distributions.

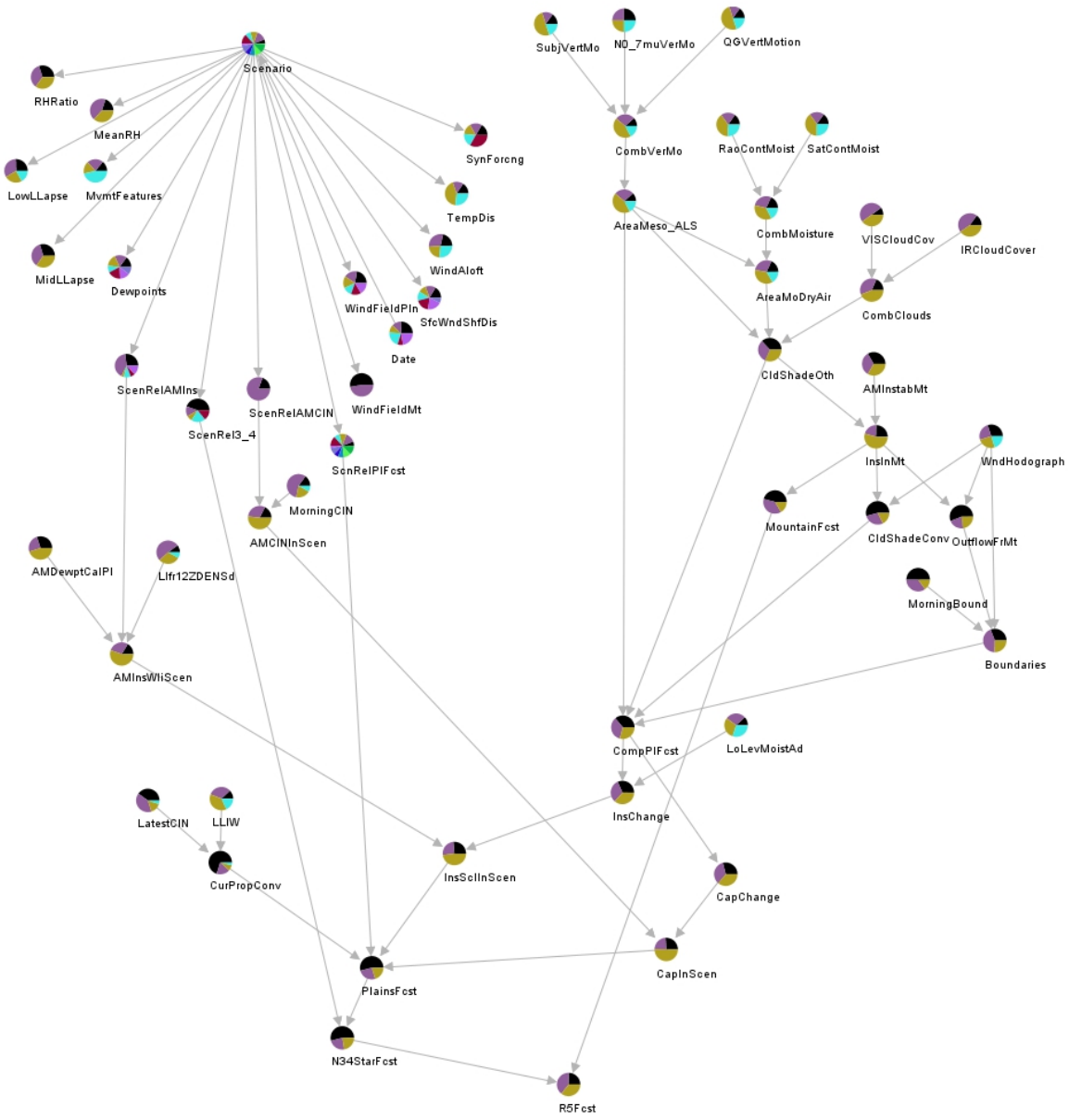


Figure A.7: Structure of the Hailfinder network printed with SISSi.

List of Algorithms

List of Figures

List of Tables

Bibliography

[1] Michael Abramovici, Manuel Neubach, Madjid Fathi, and Alexander Holland. Competing Fusion for Bayesian Applications. In *12th Information processing and management of uncertainty in knowledge-based systems*, pages 378–385, 2008.

[2] Bruce Abramson, John Brown, Ward Edwards, Allan Murphy, and Robert L. Winkler. Hailfinder: A bayesian system for forecasting severe weather. *International Journal of Forecasting*, 12(1):57 – 71, 1996.

[3] Andrew Adamatzky. Physarum machine: Implementation of a kolmogorov-uspensky machine on a biological substrate. *Parallel Processing Letters*, 17(04):455–467, 2007.

[4] Andrew Adamatzky. Physarum machines: encapsulating reaction-diffusion to compute spanning tree. *Naturwissenschaften*, 94:975–980, 2007.

[5] Andrew Adamatzky. *Physarum machines: Computers from slime mould*, volume 74. Series A. World Scientific Publishing, 2010.

[6] Andrew Adamatzky. Route 20, autobahn 7 and Physarum polycephalum: Approximating longest roads in USA and Germany with slime mould on 3D terrains. *ArXiv e-prints*, November 2012.

[7] Andrew Adamatzky. The world's colonization and trade routes formation as imitated by slime mould. *International Journal of Bifurcation and Chaos*, 22(08), 2012.

[8] Andrew Adamatzky and S. G. Akl. Trans-Canada Slimeways: Slime mould imitates the Canadian transport network. *ArXiv e-prints*, May 2011.

[9] Andrew Adamatzky, Selim Akl, Ramon Alonso-Sanz, Wesley van Dessel, Zuwairie Ibrahim, Andrew Ilachinski, Jeff Jones, Anne V.D.M. Kayem, Genaro J. MartÃnez, Pedro de Oliveira, Mikhail Prokopenko, Theresa Schubert, Peter Sloot, Emanuele Strano, and Xin-She Yang. Are motorways rational from slime mould's point of

view? *International Journal of Parallel, Emergent and Distributed Systems*, pages 1–19, 2012.

[10] Andrew Adamatzky and Ramon Alonso-Sanz. Rebuilding iberian motorways with slime mould. *Biosystems*, 105(1):89 – 100, 2011.

[11] Andrew Adamatzky, B. De Baets, and W. Van Dessel. Slime mould imitation of Belgian transport networks: redundancy, bio-essential motorways, and dissolution. *ArXiv e-prints*, December 2011.

[12] Andrew Adamatzky and Jeff Jones. Towards physarum robots: Computing and manipulating on water surface. *Journal of Bionic Engineering*, 5(4):348 – 357, 2008.

[13] Andrew Adamatzky and Jeff Jones. Programmable reconfiguration of physarum machines. *Natural Computing*, 9:219–237, 2010.

[14] Andrew Adamatzky and Jeff Jones. Road planning with slime mould: If physarum built motorways it would route m6/m74 through newcastle. *International Journal of Bifurcation and Chaos*, 20(10):3065–3084, 2010.

[15] Andrew Adamatzky, Michael Lees, and Peter Sloot. Bio-development of motorway network in the netherlands: A slime mould approach. *Advances in Complex Systems*, 2012.

[16] Andrew Adamatzky, Genaro J. MartÃnez, Sergio V. Chapa-Vergara, Rene Asomoza-Palacio, and Christopher R. Stephens. Approximating mexican highways with slime mould. *Natural Computing*, 10:1195–1214, 2011.

[17] Andrew Adamatzky and Pedro P.B. de Oliveira. Brazilian highways from slime mold's point of view. *Kybernetes: The International Journal of Systems; Cybernetics*, 40(9-10):1373–1394, 2011.

[18] Andrew Adamatzky and Mikhail Prokopenko. Slime mould evaluation of australian motorways. *International Journal of Parallel, Emergent and Distributed Systems*, 27(4):275–295, 2012.

[19] Andrew Adamatzky and T. Schubert. Schlauschleimer in Reichsautobahnen: Slime mould imitates motorway network in Germany. *ArXiv e-prints*, September 2012.

[20] H. Akaike. A new look at the statistical model identification. *Automatic Control, IEEE Transactions on*, 19(6):716–723, 1974.

[21] LM Alba and K Lindor. Non-alcoholic fatty liver disease. *Alimentary pharmacology & therapeutics*, 17(8):977–986, 2003.

[22] Reka Albert, Hawoong Jeong, and Albert-Laszlo Barabasi. Error and attack tolerance of complex networks. *Nature*, 406, August 2000.

[23] J. Aldrich. R. A. Fisher and the making of Maximum Likelihood 1912-22. *Statistical Science*, 12(3):162–176, 1997.

[24] S. Andreassen, F.V. Jensen, S.K. Andersen, B. Falck, U. Kjærulff, M. Woldbye, AR Sørensen, A. Rosenfalck, and F. Jensen. Munin-an expert emg assistant. *Computer-aided electromyography and expert systems*, 2:255–277, 1989.

[25] Mary Soon Lee Andrew Moore. Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research*, 8:67–91, 1997.

[26] Paul Angulo. Nonalcoholic fatty liver disease. *New England Journal of Medicine*, 346(16):1221–1231, 2002.

[27] E. Artin and M. Butler. *The gamma function*. Holt, Rinehart and Winston New York, 1964.

[28] Alex Aussem, Andre Tchernof, Sergio de Morais, and Sophie Rome. Analysis of lifestyle and metabolic predictors of visceral obesity with Bayesian Networks. *BMC Bioinformatics*, 11(1):487, 2010.

[29] Jayanth R. Banavar, Amos Maritan, and Andrea Rinaldo. Size and form in effcient transportation networks. *Nature*, 399:130–132, 1999.

[30] A.-L Barabasi. Scale-Free Networks: A Decade and Beyond. *Science*, 325(5939):412–413, 2009.

[31] A.-L Barabasi and E. Bonabeau. Scale-Free Networks. *Scientific American*, 288(5):50–59, 2003.

[32] M. Bayes and M. Price. An Essay towards Solving a Problem in the Doctrine of Chances. *Royal Society of London Philosophical Transactions Series I*, 53:370–418, 1763.

[33] I. A. Beinlich, H. J. Suermondt, R. M. Chavez, and G. F. Cooper. The ALARM Monitoring System: A Case Study with Two Probabilistic Inference Techniques for Belief Networks. In *Proceedings of the 2nd European Conference on Artificial Intelligence in Medicine*, pages 247–256. Springer-Verlag, 1989.

[34] John Binder, Daphne Koller, Stuart Russell, and Keiji Kanazawa. Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29(2):213–244, 1997.

[35] Mikael Bodén, Graham Dellaire, Kevin Burrage, and Timothy L. Bailey. A Bayesian Network Model of Proteins' Association with Promyelocytic Leukemia (PML) Nuclear Bodies. *Journal of Computational Biology*, 17(4):617–630, 2010.

[36] Vincenzo Bonifaci, Kurt Mehlhorn, and Girish Varma. Physarum can compute shortest paths. *Journal of Theoretical Biology*, 309:121 – 133, 2012.

[37] R. Bouckaert. Probabilistic network construction using the minimum description length principle. *Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 41–48, 1993.

[38] R.R. Bouckaert. *Bayesian Belief Networks: From Construction to Inference*. Universiteit Utrecht, Faculteit Wiskunde en Informatica, 1995.

[39] R.R. Bouckaert. Bayesian network classifiers in weka for version 3-5-6. *The University of Waikato*, 2007.

[40] Charlie Brummitt, Isabelle Laureyns, Tao Lin, David Martin, Dan Parry, Dennis Timmers, Alexander Volfson, Tianzhi Yang, Haley Yaple, and Mentor Lou Rossi. A Mathematical Study of Physarum polycephalum. In *GSMM Camp*, pages 1–24, 2010.

[41] W. Buntine. A guide to the literature on learning probabilistic networks from data. *IEEE Transactions on Knowledge and Data Engineering*, 8(2):195–210, 1996.

[42] Wray Buntine. Theory refinement on bayesian networks. In *Proceedings of the Seventh conference on Uncertainty in Artificial Intelligence*, UAI'91, pages 52–60, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[43] Wray Buntine. Learning classification trees. *Statistics and Computing*, 2:63–73, 1992.

[44] Wray Buntine. Operations for Learning with Graphical Models. *Journal of Artificial Intelligence Research*, 2:159–225, 1994.

[45] Neil A. Campbell and Jane B. Reece. *Biology*. Pearson Education, 6th edition, 2002.

[46] Cassio P. de Campos, Zhi Zeng, and Qiang Ji. Structure learning of Bayesian networks using constraints. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 113–120. ACM, 2009.

[47] Rui Chang and Martin Stetter. A knowledge-based dynamic bayesian framework towards molecular network modeling and quantitative prediction. In *BIOCOMP*, pages 37–43, 2007.

[48] Michael Charlton. Nonalcoholic fatty liver disease: a review of current understanding and future impact. *Clinical gastroenterology and hepatology*, 2(12):1048–1058, 2004.

[49] J. Cheng, D.A. Bell, and W. Liu. Learning belief networks from data: An information theory based approach. In *Proceedings of the sixth international conference on Information and knowledge management*, pages 325–331. ACM, 1997.

[50] J. Cheng and R. Greiner. Comparing bayesian network classifiers. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 101–108. Morgan Kaufmann Publishers Inc., 1999.

[51] D. M. Chickering. A Transformational Characterization of Equivalent Bayesian Network Structures. In *UAI '95: Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence*, pages 87–98. Morgan Kaufmann, 1995.

[52] David Chickering. Learning Bayesian Networks is NP-Complete. In *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130. Springer-Verlag, 1996.

[53] D.M. Chickering, D. Heckerman, and C. Meek. Large-sample learning of bayesian networks is np-hard. *The Journal of Machine Learning Research*, 5:1287–1330, 2004.

[54] Dietmar Cieslik and Johann Linhart. *Steiner minimal trees.* Kluwer Academic publishers, 1998.

[55] W.G. Cole. Understanding bayesian reasoning via graphical displays. In *ACM SIGCHI Bulletin*, volume 20, pages 381–386. ACM, 1989.

[56] Gregory F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2-3):393 – 405, 1990.

[57] Gregory F. Cooper and Edward Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, 1992.

[58] Harald Cramér. Mathematical methods of statistics. *Princeton University Press, Princeton*, 1946.

[59] Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks.* Cambridge University Press, 2009.

[60] S. Dasgupta. Learning polytrees. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 134–141. Morgan Kaufmann Publishers Inc., 1999.

[61] A. Philip Dawid. Conditional Independence in Statistical Theory. *Journal of the Royal Statistical Society. Series B (Methodological)*, 41(1):1–31, 1979.

[62] A. Philip Dawid. Conditional independence for statistical operations. *Annals of Statistics*, 8(3):598–617, 1980.

[63] L. de Campos and J. Puerta. Stochastic local algorithms for learning belief networks: Searching in the space of the orderings. *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, pages 228–239, 2001.

[64] Luis M. de Campos, Juan M. Fernandez-Luna, Jose A. Gamez, and Jose M. Puerta. Ant colony optimization for learning bayesian networks. *International Journal of Approximate Reasoning*, 31(3):291 – 311, 2002.

[65] M. H. DeGroot and Schervish M. J. *Probability and Statistics.* Pearson, 4th edition, 2011.

[66] M. Dejori, B. Schuermann, and M. Stetter. Hunting drug targets by systems-level modeling of gene expression profiles. *NanoBioscience, IEEE Transactions on*, 3(3):180–191, 2004.

[67] Mathäus Dejori, Anton Schwaighofer, Volker Tresp, and Martin Stetter. Mining Functional Modules in Genetic Networks with Decomposable Graphical Models. *OMICS: A Journal of Integrative Biology*, 8(2):176–188, 2004.

[68] B. N. Delaunay. Sur la Sphère Vide. *Bulletin of Academy of Sciences of the USSR*, pages 793–800, 1934.

[69] F.J. Diez. Parameter adjustment in bayes networks. the generalized noisy or-gate. In *Proceedings of the Ninth international conference on Uncertainty in artificial intelligence*, pages 99–105. Morgan Kaufmann Publishers Inc., 1993.

[70] A.R.T. Donders, G.J.M.G. van der Heijden, T. Stijnen, K.G.M. Moons, et al. Review: a gentle introduction to imputation of missing values. *Journal of clinical epidemiology*, 59(10):1087–1091, 2006.

[71] Marco Dorigo. Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano, Italy*, 1992.

[72] Eibe, Frank and Holmes, Geoffrey and Pfahringer, Bernhard and Reutemann, Peter and Witten, Ian H. The WEKA data mining software: An update. *SIGKDD Explorations, Volume 11, Issue 1*, 2009.

[73] G. Elidan, M. Ninio, N. Friedman, and D. Shuurmans. Data perturbation for escaping local maxima in learning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 132–139, 2002.

[74] Elisa Fabbrini, Shelby Sullivan, and Samuel Klein. Obesity and nonalcoholic fatty liver disease: biochemical, metabolic, and clinical implications. *Hepatology*, 51(2):679–689, 2010.

[75] U. Fayyad and K. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. *Thirteenth International Joint Conference on Articial Intelligence*, pages 1022–1027, 1993.

[76] William Feller. *An Introduction to Probability Theory and Its Applications*. Wiley, 2th edition, 1971.

[77] R. A. Fisher. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 222(594-604):309–368, 1922.

[78] J.E. Freund and G.A. Simon. *Modern elementary statistics*, volume 12. Prentice-Hall Englewood Cliffs, New Jersey, 1967.

[79] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2):131–163, 1997.

[80] N. Friedman and M. Goldszmidt. Learning bayesian networks with local structure. In *Proceedings of the twelfth international conference on uncertainty in artificial intelligence*, pages 252–262. Morgan Kaufmann Publishers Inc., 1996.

[81] N. Friedman, M. Goldszmidt, et al. Discretizing continuous attributes while learning bayesian networks. In *Machine Learning -International Workshop then Conference-*, pages 157–165. Morgan Kaufmann Publishers, Inc., 1996.

[82] Nir Friedman and Daphne Koller. Being Bayesian About Network Structure. A Bayesian Approach to Structure Discovery in Bayesian Networks. *Machine Learning*, 50(1):95–125, 2003.

[83] Nir Friedman, Michal Linial, Iftach Nachman, and Dana Pe'er. Using Bayesian networks to analyze expression data. In *RECOMB*, pages 127–135, 2000.

[84] D. Geiger and D. Heckerman. Learning gaussian networks. In *Proceedings of the Tenth international conference on Uncertainty in artificial intelligence*, pages 235–243. Morgan Kaufmann Publishers Inc., 1994.

[85] Dan Geiger and Judea Pearl. On the logic of causal models. In *Proceedings of the Fourth Annual Conference on Uncertainty in Artificial Intelligence*, UAI '88, pages 3–14, Amsterdam, The Netherlands, 1988. North-Holland Publishing Co.

[86] Dan Geiger, Thomas Verma, and Judea Pearl. d-separation: From theorems to algorithms. In *Proceedings of the Fifth Annual Conference on Uncertainty in Artificial Intelligence*, UAI '89, pages 139–148, Amsterdam, The Netherlands, 1989. North-Holland Publishing Co.

[87] Dan Geiger, Thomas Verma, and Judea Pearl. Identifying independence in bayesian networks. *Networks*, 20(5):507–534, 1990.

[88] O. Gevaert, F. D. Smet, D. Timmerman, Y. Moreau, and B. D. Moor. Predicting the prognosis of breast cancer by integrating clinical and microarray data with Bayesian networks. *Bioinformatics*, 22(14):e184–e190, 2006.

[89] P. Giudici and P.J. Green. Decomposable graphical gaussian model determination. *Biometrika*, 86(4):785–801, 1999.

[90] Fred Glover. Tabu search-part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[91] Fred Glover. Tabu search-part ii. *ORSA Journal on computing*, 2(1):4–32, 1990.

[92] Fred Glover and Claude McMillan. The general employee scheduling problem. an integration of ms and ai. *Computers & operations research*, 13(5):563–573, 1986.

[93] Eugene M. Goodman. Physarum polycephalum: A review of a model system using a structure-function approach. *Int. Rev. Cytol*, 63:1 – 58, 1980.

[94] D. Heckerman and D. Geiger. Learning bayesian networks: a unification for discrete and gaussian domains. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 274–284. Morgan Kaufmann Publishers Inc., 1995.

[95] David Heckerman. A tutorial on learning with bayesian networks. In *Innovations in Bayesian Networks*, volume 156 of *Studies in Computational Intelligence*, pages 33–82. Springer Berlin Heidelberg, 2008.

[96] David Heckerman, Dan Geiger, and David M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, 1995.

[97] David Heckerman, Abe Mamdani, and Michael P. Wellman. Real-world applications of bayesian networks. *Commun. ACM*, 38(3):24–26, March 1995.

[98] D.E. Heckerman, E.J. Horvitz, and B.N. Nathwani. *Toward normative expert systems: The Pathfinder project*. Knowledge Systems Laboratory, Medical Computer Science, Stanford University, 1992.

[99] D.E. Heckerman and B.N. Nathwani. An evaluation of the diagnostic accuracy of pathfinder. *Computers and Biomedical Research*, 25(1):56–74, 1992.

[100] D.E. Heckerman, B.N. Nathwani, et al. Toward normative expert systems: Part ii. probability-based representations for efficient knowledge acquisition and inference. *Methods of information in medicine*, 31(2):106–116, 1992.

[101] R.A. Howard, J.E. Matheson, and Strategic Decisions Group. *Readings on the principles and applications of decision analysis*. Number Bd. 1 in Readings on the Principles and Applications of Decision Analysis. Strategic Decisions Group, 1983.

[102] S. Isci, C. Ozturk, J. Jones, and H. H. Otu. Pathway analysis of high-throughput biological data within a Bayesian network framework. *Bioinformatics*, 27(12):1667–1674, 2011.

[103] Kentaro Ito, Anders Johansson, Toshiyuki Nakagaki, and Atsushi Tero. Convergence Properties for the Physarum Solver. *arXiv:1101.5249v1*, 2011.

[104] Schervish M. J. *Theory of Statistics*. Springer, 2th edition, 1997.

[105] Tommi Jaakkola, David Sontag, Amir Globerson, and Marina Meila, editors. *Learning Bayesian Network Structure using LP Relaxations*, 2010.

[106] Anders Johannson and James Zou. A slime mold solver for linear programming problems. In S.Barry Cooper, Anuj Dawar, and Benedikt Löwe, editors, *How the World Computes*, volume 7318 of *Lecture Notes in Computer Science*, pages 344–354. Springer Berlin Heidelberg, 2012.

[107] N. Kamiya. Protoplasmic streaming. *Protoplasmatologia*, 8:1–199, 1959.

[108] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[109] Thomas H. Kent. FloraFinder.com - Physarum Polycephalum (Slime Mold). `http://www.florafinder.com/Species/Physarum_polycephalum.php`, January 2013.

[110] J.H. Kim and J. Pearl. A computational model for causal and diagnostic reasoning in inference systems. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 190–193, 1983.

[111] Scott Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics*, 34:975–986, 1984.

[112] Uffe B. Kjaerulff and Anders L. Madsen. *Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis*. Springer Science, 2008.

[113] Mikko Koivisto and Kismat Sood. Exact Bayesian Structure Discovery in Bayesian Networks. *The Journal of Machine Learning Research*, 5:549–573, 2004.

[114] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. The MIT Press, 2009.

[115] P. Komarek and A. Moore. A dynamic adaptation of ad-trees for efficient machine learning on large data sets. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 495–502, 2000.

[116] K. Korb and A. Nicholson. *Bayesian Artificial Intelligence*. Chapman and Hall, 2nd edition, 2010.

[117] S.C. Kramer and H.W. Sorenson. Bayesian parameter estimation. *Automatic Control, IEEE Transactions on*, 33(2):217 –222, feb 1988.

[118] Kristian Kristensen and Ilse A. Rasmussen. The use of a bayesian network in the design of a decision support system for growing malting barley without use of pesticides. *Computers and Electronics in Agriculture*, 33(3):197 – 217, 2002.

[119] Joseph B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.

[120] Y. Kuramoto. *Chemical Oscillations, Waves, and Turbulence*. Springer–Verlag, New York, 1984.

[121] Wai Lam and Fahiem Bacchus. Learning Bayesian belief networks: An approach based on the MDL principle. *Computational Intelligence*, 10(3):269–293, 1994.

[122] S. L. Lauritzen and D. J. Spiegelhalter. Local Computation with Probabilities on Graphical Structures and their Application to Expert Systems. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 50(2):157–224, 1988.

[123] E.L. Lehmann and J.P. Romano. *Testing statistical hypotheses*. Springer, 2005.

[124] D. Madigan, S.A. Andersson, M.D. Perlman, and C.T. Volinsky. Bayesian model averaging and model selection for markov equivalence classes of acyclic digraphs. *Communications in Statistics–Theory and Methods*, 25(11):2493–2519, 1996.

[125] D. Madigan, J. York, and D. Allard. Bayesian graphical models for discrete data. *International Statistical Review/Revue Internationale de Statistique*, pages 215–232, 1995.

[126] Marc Teyssier and Daphne Koller, editors. *Ordering-Based Search: A Simple and Effective Algorithm for Learning Bayesian Networks*, 2005.

[127] Luca Masi and Massimiliano Vasile. Optimal multi-objective discrete decision making using a multidirectional modified Physarum Solver. In *Proceedings of the EVOLVE 2012 International Conference*, Mexico City, 2012.

[128] Kenji Matsumoto, Tetsuo Ueda, and Yonosuke Kobatake. Propagation of phase wave in relation to tactic responses by the plasmodium of physarum polycephalum. *Journal of Theoretical Biology*, 122(3):339 – 345, 1986.

[129] D.G. Mayer and D.G. Butler. Statistical validation. *Ecological Modelling*, 68(1-2):21 – 32, 1993.

[130] Richard E. Mayer. Multimedia learning. *Psychology of Learning and Motivation*, 41:85 – 139, 2002.

[131] C. Meek. Causal inference and causal explanation with background knowledge. In *Uncertainty in Artificial Intelligence*, volume 11, pages 403–410, 1995.

[132] C. Meek. Finding a path is harder than finding a tree. *Journal of Artificial Intelligence Research*, 15:383–389, 2001.

[133] Kishan Mehrotra, Chilukur K Mohan, and Sanjay Ranka. *Artificial Neural Networks*. the MIT Press, 1997.

[134] B. Middleton, M. Shwe, D. Heckerman, M. Henrion, E. Horvitz, H. Lehmann, and G. Cooper. Probabilistic diagnosis using a reformulation of the internist-1/qmr knowledge base, part ii. *Methods of Information in Medicine*, 30:256–267, 1991.

[135] Haruki Miura and Masafumi Yano. A model of organization of size invariant positional information in taxis of physarum plasmodium. *Progress of Theoretical Physics*, 100(2):235–251, 1998.

[136] Tomoyuki Miyaji and Isamu Ohnishi. Mathematical analysis to an adaptive network of the plasmodium system. *Hokkaido Mathematical Journal*, 36(2):445–465, 2007.

[137] Tomoyuki Miyaji and Isamu Ohnishi. Physarum can solve the shortest path problem on riemannian surface mathematically rigorously. *International Journal of Pure and Applied Mathematics*, 47(3):353–369, 2008.

[138] Tomoyuki Miyaji, Isamu Ohnishi, Atsushi Tero, and Toshiyuki Nakagaki. Failure to the shortest path decision of an adaptive transport network with double edges in plasmodium system. *International Journal of Dynamical Systems and Differential Equations*, 1(3):210–219, 2008.

[139] Yoshihiro Miyake, Sunao Tabata, Hirofumi Murakami, Masafumi Yano, and Hiroshi Shimizu. Environment-dependent self-organization of positional information field in chemotaxis of physarum plasmodium. *Journal of Theoretical Biology*, 178(4):341 – 353, 1996.

[140] Yoshihiro Miyake, Hideki Tada, Masafumi Yano, and Hiroshi Shimizu. Relationship between intracellular period modulation and external environment change in physarum plasmodium. *Cell Struc. Funct.*, 19(6):363–370, dec 1994.

[141] Kevin Patrick Murphy. *Dynamic bayesian networks: representation, inference and learning*. PhD thesis, University of California, Berkeley, 2002.

[142] R. Nagai, R.N. Yoshimoto, and N. Kamiya. Cyclic production of tension force in the plasmodial strand of physarum polycephalum and its relation to microfilament morphology. *Journal of Cell Science*, 33(1):205–225, 1978.

[143] A. Nägele, M. Dejori, and M. Stetter. Robust learning of high-dimensional biological networks with bayesian networks. *Robust Intelligent Systems*, page 139, 2008.

[144] T. Nakagaki, R. Kobayashi, Y. Nishiura, and T. Ueda. Obtaining multiple separate food sources: behavioural intelligence in the physarum plasmodium. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 271(1554):2305–2310, 2004.

[145] T. Nakagaki, T. Saigusa, A. Tero, and R. Kobayashi. *Topological Aspects of Critical Systems and Networks. Effects of amount of food on path selection in the transport network of an amoeboid organism*, chapter 12, pages 94–100. World Scientific Publishing Co. Pte. Ltd., 2007.

[146] Toshiyuki Nakagaki. Smart behavior of true slime mold in a labyrinth. *Research in Microbiology*, 152(9):767–770, 2001.

[147] Toshiyuki Nakagaki and Robert D. Guy. Intelligent behaviors of amoeboid movement based on complex dynamics of soft matter. *Soft Matter*, 4:57–67, 2008.

[148] Toshiyuki Nakagaki, Makoto Iima, Tetsuo Ueda, Yasumasa Nishiura, Tetsu Saigusa, Atsushi Tero, Ryo Kobayashi, and Kenneth Showalter. Minimum-risk path finding by an adaptive amoebal network. *Phys. Rev. Lett.*, 99, Aug 2007.

[149] Toshiyuki Nakagaki, Atsushi Tero, Ryo Kobayashi, Isamu Onishi, and Tomoyuki Miyaji. Computational ability of cells based on cell dynamics and adaptability. *New Generation Computing*, 27:57–81, 2008.

[150] Toshiyuki Nakagaki, Hiroyasu Yamada, and Masahiko Hara. Smart network solutions in an amoeboid organism. *Biophysical Chemistry*, 107(1):1–5, 2004.

[151] Toshiyuki Nakagaki, Hiroyasu Yamada, and Masami Ito. Reaction-diffusion-advection model for pattern formation of rhythmic contraction in a giant amoeboid cell of the physarum plasmodium. *Journal of Theoretical Biology*, 197(4):497–506, 1999.

[152] Toshiyuki Nakagaki, Hiroyasu Yamada, and Agota Toth. Intelligence: Maze-solving by an amoeboid organism. *Nature*, 407(6803):470, 2000.

[153] Toshiyuki Nakagaki, Hiroyasu Yamada, and Ágota Tóth. Path finding by tube morphogenesis in an amoeboid organism. *Biophysical Chemistry*, 92(1–2):47–52, 2001.

[154] Toshiyuki Nakagaki, Hiroyasu Yamada, and Tetsuo Ueda. Interaction between cell shape and contraction pattern in the physarum plasmodium. *Biophysical Chemistry*, 84(3):195–204, 2000.

[155] R.B. Nelsen. Pearson product-moment correlation coefficient, encyclopedia of mathematics. `http://www.encyclopediaofmath.org/index.php?title=Pearson_product-moment_correlation_coefficient&oldid=18562`, February 2013.

[156] Saulius Pacekajus. Ordering estimation for bayesian network structure learning. *Aalborg University, Denmark*, 2009.

[157] Pekka Parviainen and Mikko Koivisto. Exact structure discovery in Bayesian networks with less space. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 436–443, 2009.

[158] Judea Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29(3):241 – 288, 1986.

[159] Judea Pearl. *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Morgan Kaufmann, San Francisco and Calif, 2 edition, 1988.

[160] Judea Pearl and T.S. Verma. A theory of inferred causation. In *Conference on Knowledge Representation and Reasoning (KR)*, pages 441–452, 1991.

[161] Karl Pearson. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(302):157–175, 1900.

[162] David Preiss and Naveed Sattar. Non-alcoholic fatty liver disease: an overview of prevalence, diagnosis, pathogenesis and treatment considerations. *Clinical Science*, 115:141–150, 2008.

[163] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technology Journal*, 36:1389–1401, 1957.

[164] Qingfeng Chen and Y.-P P. Chen. Mining Protein Kinases Regulation Using Graphical Models. *IEEE Transactions on NanoBioscience*, 10(1):1–8, 2011.

[165] R Scott Rector, John P Thyfault, Yongzhong Wei, and Jamal A Ibdah. Non-alcoholic fatty liver disease and the metabolic syndrome: an update. *World journal of gastroenterology: WJG*, 14(2):185, 2008.

[166] Chris R. Reid and Madeleine Beekman. Solving the towers of hanoi - how an amoeboid organism efficiently constructs transport networks. *The Journal of Experimental Biology*, 2013.

[167] E B Ridgway and A C Durham. Oscillations of calcium ion concentrations in physarum polycephalum. *The Journal of Cell Biology*, 69(1):223–226, 1976.

[168] C. Riggelsen. *Approximation methods for efficient learning of Bayesian networks*, volume 168. Ios PressInc, 2006.

[169] Sheldon Ross. *First Course in Probability*. Prentice Hall, 7th edition, 2005.

[170] Elham Salehi and Robin Gras, editors. *An empirical comparison of the efficiency of several local search heuristics algorithms for Bayesian network structure learning*, 2009.

[171] Arun J Sanyal. Aga technical review on nonalcoholic fatty liver disease. *Gastroenterology*, 123(5):1705–1725, 2002.

[172] Torsten Schön, Martin Stetter, and Elmar W Lang. Structure learning for Bayesian Networks using the Physarum Solver. In *Proceedings of the 11th International Conference on Machine Learning and Applications ICMLA 2012*, pages 488–493. IEEE, December 2012.

[173] Torsten Schön, Martin Stetter, Ana Maria Tomé, and Elmar W Lang. A new Physarum Learner for network structure learning from biomedical data. In *Proceedings of the 6th International Conference in Bio-inspired Systems and Signal Processing BIOSIGNALS 2013*, pages 151–156, February 2012.

[174] G. Schwarz. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[175] Marco Scutari. bnlearn - an R package for Bayesian network learning and inference. `http://www.bnlearn.com/bnrepository`, March 2013.

[176] P. Sedlmeier and G. Gigerenzer. Teaching bayesian reasoning in less than two hours. *Journal of Experimental Psychology: General*, 130(3):380, 2001.

[177] J. A. Sepulchre and A. Babloyantz. Motions of spiral waves in oscillatory media and in the presence of obstacles. *Phys. Rev. E*, 48:187–195, Jul 1993.

[178] J.A. Sepulchre, A. Babloyantz, and L. Steels. Path finding with nonlinear waves. In *Proceedings of the International Conference on Artificial Neural Networks*, ICANN-91, pages 1265–1268, 1991.

[179] M.A. Shwe, B. Middleton, DE Heckerman, M. Henrion, EJ Horvitz, HP Lehmann, and GF Cooper. Probabilistic diagnosis using a reformulation of the internist-1/qmr knowledge base, part i. *Methods of Information in Medicine*, 30:241–255, 1991.

[180] T. Silander and P. Myllymaki. A simple approach for finding the globally optimal bayesian network structure. *arXiv preprint arXiv:1206.6875*, 2012.

[181] Tomi Silander and Petri Myllymäki, editors. *A simple approach for finding the globally optimal Bayesian network structure*, 2006.

[182] J. Siriwardana and S.K. Halgamuge. Fast shortest path optimization inspired by shuttle streaming of physarum polycephalum. In *Evolutionary Computation (CEC), 2012 IEEE Congress*, pages 1–8, June 2012.

[183] J. Q. Smith. Influence diagrams for statistical modelling. *Annals of Statistics*, 17(2):654–672, 1989.

[184] Yuning Song, Liang Liu, and Huadong Ma. A physarum-inspired algorithm for minimal exposure problem in wireless sensor networks. In *Wireless Communications and Networking Conference (WCNC), 2012 IEEE*, pages 2151–2156, April 2012.

[185] Yuning Song, Liang Liu, Huadong Ma, and Athanasios V. Vasilakos. Physarum optimization: a new heuristic algorithm to minimal exposure problem. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, Mobicom '12, pages 419–422, New York, NY, USA, 2012. ACM.

[186] David J. Spiegelhalter, A. Philip Dawid, Steffen L. Lauritzen, and Robert G. Cowell. Bayesian analysis in expert systems. *Statistical Science*, 8(3):219–283, 1993.

[187] David J. Spiegelhalter and Steffen L. Lauritzen. Sequential updating of conditional probabilities on directed graphical structures. *Networks*, 20(5):579–605, 1990.

[188] P. Spirtes, C. Glymour, and R. Scheines. *Causation, prediction, and search*, volume 81. MIT press, 2001.

[189] Emanuele Strano, Andrew Adamatzky, and Jeff Jones. Physarum itinerae: Evolution of roman roads with slime mould. *International Journal of Nanotechnology and Molecular Computation (IJNMC)*, 3(2), 2013.

[190] Steven H. Strogatz. Exploring complex networks. *Nature*, 401:268–276, 2001.

[191] S.P. Sutera and R. Skalak. The history of poiseuille's law. *Annual Review of Fluid Mechanics*, 25:1–19, 1993.

[192] J. Suzuki. A construction of bayesian networks from databases based on an mdl principle. In *Proceedings of the Ninth international conference on Uncertainty in artificial intelligence*, pages 266–273. Morgan Kaufmann Publishers Inc., 1993.

[193] Atsuko Takamatsu, Kengo Takahashi, Makoto Nagao, and Yoshimi Tsuchiya. Frequency coupling model for dynamics of responses to stimuli in plasmodium of physarum polycephalum. *Journal of the Physical Society of Japan*, 66(6):1638–1646, 1997.

[194] Atsushi Tero, Ryo Kobayashi, and Toshiyuki Nakagaki. Physarum solver: A biologically inspired method of road-network navigation. *Physica A: Statistical Mechanics and its Applications*, 363(1):115–119, 2006.

[195] Atsushi Tero, Ryo Kobayashi, and Toshiyuki Nakagaki. A mathematical model for adaptive transport network in path finding by true slime mold. *Journal of Theoretical Biology*, 244(4):553–564, 2007.

[196] Atsushi Tero, Toshiyuki Nakagaki, Kazutaka Toyabe, Kenji Yumiki, and Ryo Kobayashi. A method inspired by physarum for solving the steiner problem. *IJUC*, 6(2):109–123, 2010.

[197] Atsushi Tero, Seiji Takagi, Tetsu Saigusa, Kentaro Ito, Dan P. Bebber, Mark D. Fricker, Kenji Yumiki, Ryo Kobayashi, and Toshiyuki Nakagaki. Rules for biologically inspired adaptive network design. *Science*, 327(5964):439–442, 2010.

[198] Atsushi Tero, Kenji Yumiki, Ryo Kobayashi, Tetsu Saigusa, and Toshiyuki Nakagaki. Flow-network adaptation in Physarum amoebae. *Theory in Biosciences*, 127(2):89–94, 2008.

[199] The Apache Software Foundation. Commons Math: The Apache Commons Mathematics Library. `http://commons.apache.org/math`, February 2013.

[200] O. Troyanskaya, D. Botstein, and R. Altman. Missing value estimation. *A practical approach to microarray data analysis*, pages 65–75, 2003.

[201] Soichiro Tsuda, Masashi Aono, and Yukio-Pegio Gunji. Robust and emergent physarum logical-computing. *BioSystem*, 73:45–55, January 2004.

[202] P.J. van Laarhoven and E.H. Aarts. *Simulated annealing: theory and applications*, volume 37. Springer, 1987.

[203] Thomas Verma and Judea Pearl. Equivalence and synthesis of causal models. In *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*, UAI '90, pages 255–270, New York, NY, USA, 1991. Elsevier Science Inc.

[204] Thomas Verma and Judea Pearl. An algorithm for deciding if a set of observed independencies has a causal explanation. In *Proceedings of the Eighth international conference on Uncertainty in artificial intelligence*, UAI'92, pages 323–330, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[205] Tom Verma and Judea Pearl. Causal networks: Semantics and expressiveness. In *Proceedings of the Fourth Conference on Uncertainty in Artificial Intelligence (UAI-88)*, pages 59–72, 1988.

[206] Shin Watanabe, Atsushi Tero, Atsuko Takamatsu, and Toshiyuki Nakagaki. Traffic optimization in railroad networks using an algorithm mimicking an amoeba-like organism, physarum plasmodium. *Biosystems*, 105(3):225 – 232, 2011.

[207] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

[208] I.H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[209] David H Wolpert and William G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.

[210] Hiroyasu Yamada, Toshiyuki Nakagaki, and Masami Ito. Pattern formation of a reaction-diffusion system with self-consistent flow in the amoeboid organism physarum plasmodium. *Phys. Rev. E*, 59:1009–1014, Jan 1999.