Learning Bayesian Networks Using Fast Heuristics

Liangyi Luo

M.Sc. Thesis
UNIVERSITY OF HELSINKI
Department of Computer Science

Helsinki, May 26, 2017

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Liangyi Luo

Työn nimi — Arbetets titel — Title
Learning Bayesian Networks Using Fast Heuristics

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
M.Sc. Thesis	May 26, 2017	46

Tiivistelmä — Referat — Abstract

This thesis addresses score-based learning of Bayesian networks from data using a few fast heuristics. The algorithmic implementation of the heuristics is able to learn size 30-40 networks in seconds and size 1000-2000 networks in hours. Two algorithms, which are devised by Scanagatta *et al.* and dubbed *Independence Selection* and *Acyclic Selection OBS* have the capacity of learning very large Bayesian networks without the liabilities of the traditional heuristics that require maximum in-degree or ordering constraints. The two algorithms are respectively called *Insightful Searching* and *Acyclic Selection Obeying Boolean-matrix Sanctioning* (acronym ASOBS) in this thesis. This thesis also serves as an expansion of the work of Scanagatta *et al.* by revealing a computationally simple ordering strategy called *Randomised Pairing Greedy Weight* (acronym RPGw) that works well as an adjunct along with ASOBS with corresponding experiment results, which show that ASOBS was able to score higher and faster with the help of RPGw. Insightful Searching, ASOBS, and RPGw together form a system that learns Bayesian networks from data very fast.

Avainsanat — Nyckelord — Keywords
Bayesian network structure learning, score-based learning, heuristic, artificial intelligence

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Contents

1 Introduction

This thesis is about score-based learning of Bayesian networks using heuristics, therefore the learning is performed by heuristic algorithms, and the result of learning is measured by *scores*, which are real numbers computed by *score functions*. The learning methods in this thesis weigh computational simplicity over scores, but this does not mean improving scores is out of consideration. The result should be "good enough" meaning the scores of the learnt Bayesian networks should be high enough. How high is "enough" depends on the learning requirements. Given different requirements, different methods may apply. In general, there are exact methods, which achieve optimal scores at all costs; approximation methods, which guarantee certain non-optimal scores; or heuristic methods, which achieve scores as high as they get but provide no specific guarantee[1]. Choices are at the discretion of users. Results are limited by the reality. This thesis focuses on heuristics.

This thesis covers a set of heuristic algorithms that is capable of, starting from raw data, learning networks of about 30-40 nodes in a few seconds and about 1000-2000 nodes in a few hours using mid-to-high-end personal computers available in 2017. (A few assumptions apply.) These algorithms do not guarantee Bayesian networks that meet specific quality requirements such as "the scores have to be optimal" or "the scores have to exceed a certain value".

Achieving the same level of learning speed and quality might be difficult using merely traditional heuristics. Traditional heuristic algorithms often require capping the maximum in-degrees of the network structures, which does not help avoid examining some evidently low-potential structures under the caps. Traditional heuristic algorithms might also use topological orderings as constraints in order to guarantee directed acyclic graphs as network structures, but this comes with the price of excluding all structures that have different topological orderings, some of which may be high-potential structures.

Independence Selection and Acyclic Selection OBS [39] (OBS probably stands for "Ordering-Based Search", which was hinted but not confirmed in the original paper) debuted in 2015 need no maximum in-degree and do not work with ordering constraints, in spite of which they have the capacity of learning Bayesian networks with a larger number of variables in reasonable time for good scores. In this thesis, Independence Selection is called *Insightful Searching* and Acyclic Selection OBS is called *Acyclic Selection Obeying Boolean-matrix Sanctioning* (acronym ASOBS). Insightful Searching uses a constant-time computable approximation to the score function of *Bayesian Information Criterion* [40]. ASOBS does not work with ordering constraints

[1] Approximation algorithms and heuristics may have intersections but I would like to distinguish them by aims: guaranteed results or guaranteed resource budgets.

and was shown outperforming Ordering-Based Search [44]—a peer algorithm that does in several studies [39, 45, 26]. That being said, ASOBS also requires an ordering, which is used as a queue, to work, and the supply of orderings determines the quality of the result. This thesis includes an ordering strategy—*Randomised Pairing Greedy Weight* (acronym RPGw) that works better than supplying random orderings and computationally simpler than another alternative strategy known as BestFirst-Based ordering generation [45].

The key idea of this thesis is what I call *stairs of approximation*. Because both Insightful Searching and RPGw benefit from the idea of using an additional degree of approximation to an already existing "chain" of approximations. I would rather call the "chain" of approximations "stairs" of approximation. Ascending the stairs, step by step, one may approximate the truth but every step upwards will become more difficult. Therefore, one may start from a lower stair that is easy enough to step on, the truth might not be clear from a greater distance but it might be visible enough to guide an easier walk of ascending.

1.1 Thesis Scope and Outline

Discussing a topic such as learning Bayesian networks is difficult without specifying a few scopes to gain better focus and making many assumptions to ease the discussion. The following two paragraphs include most scopes and assumptions, but a few subject specific assumptions will appear in the appropriate places in later sections.

This thesis focuses on heuristics for score-based learning. Methods that are not heuristics for score-based learning will not be elaborated. I will assume heuristics are necessary when a NP-hard problem has to be handled under the condition that there is no applicable assumption on how big the problem instance might be.

I will only discuss the cases of learning where the data are generated by a constant set of variables: variables and their parameters do not change midway when generating a data set. I will also assume that each instance in any data set is independently sampled: the sampling of every instance does not affect the sampling of every else. Moreover, the discussion and presentation concern only discrete variables for the sake of convenience and focus. I assume all data sets used for learning to be *complete*, which means there is no missing datum or hidden variable. With this assumption, I need not cover applicable methods for handling incomplete data sets, but this is by no means suggesting those methods are not important. On the contrary, handling incomplete data sets is very important since raw data sets are often incomplete. The assumption also means the set of algorithms in this thesis cannot yet handle incomplete data sets, thus it needs the corresponding expansion to gain such capacity.

The structure of this thesis is as follows. Section 2 is about the essential concepts necessarily for comprehending the rest of the thesis, they are *Bayesian network*, *score-based structure learning*, and *decomposed learning*. Section 3 is about the score function required by the heuristics in this thesis. Section 4 includes the heuristics with the corresponding algorithmic implementation—Insightful Searching and ASOBS. Section 5 is about ordering strategies for ASOBS including RPGw. RPGw was tested. The experiment results appear in Section 5.3. Finally in Section 6, I conclude this thesis, discuss possible usages of the heuristics presented, and take a few outlooks.

1.2 Bayesian Networks are for Knowledge Representation

Before I decided to take on my thesis topic, I hadn't had much of an impression on Bayesian networks, if any, it was bad. I had taken one or two courses about Bayesian statistics and learnt some basics, but the most indelible memory had come from a project, during which my partner had tried compressing data using Bayesian networks,"what an awful and slow tool" sums up my initial impression. Despite all of these, I took the topic with delight and wrote this thesis.

My view of Bayesian networks shifted when I understood their purpose, which is, in the fewest words possible, "knowledge representation". Learning Bayesian networks is about computers that acquire and present knowledge. But, what is the knowledge?

For those who have ruminated, "What is knowledge?" they perhaps would not be able to recollect a satisfying definition right away. I could think nothing but a few examples. So I opened my digital version of *Merriam-Webster's Collegiate Dictionary 11th Edition* and entered "knowledge" - as they appeared in the definition were all familiar words: *information*, *understanding*, *truth*, and so on, each of which leads to another whole set of ideas. *Knowledge* is a slippery concept that encapsulates many things.

Then, how to acquire knowledge? Children may be told that they should study, read books, learn from others, and the children nowadays have also the Internet. But think about it, where did the knowledge in books, in other people's mind, or on the Internet come from?

Human brains are able to turn what they see and think into knowledge, but brains are only the start. With the help of symbols and writing, knowledge is able to survive and propagate without a specific individual. Armed with logic, mathematics, and science, humans have become an eccentric species that creates knowledge for the sake of creating knowledge. About half a century ago, the era of digital computers has arrived. People have started digitalising their tools and transferring some "mental labours" incurred by seeking knowledge to machines, then knowledge exploded. For the first time in history, knowledge has become more than what people could handle. As a

result, mankind gave birth to its first child—artificial intelligence and has since been urging it to acquire knowledge in the name of "machine learning".

There are two types of knowledge machine learning can handle: the properties of things, such as "A is a red cube", "B is a red ball", and "C is a yellow ball"; and, the dependencies among things, such as "a red cube is red because it was made of red material or painted red" and "a red cube being red has (probably) nothing to do with being cut into a cube". A machine learning system that learns properties can do classification. When there comes a yellow cube, the machine can tell that it is in the same class of A by shape and C by colour. A machine learning system that learns dependencies can do inference. When there comes a yellow cube, the machine can infer that it was probably made of yellow material or painted yellow, and it would still be yellow even if it had been cut into a ball because its shape was independent of its colour. Such knowledge of dependencies is what Bayesian networks are capable of. (To be specific, Bayesian networks model dependencies with Bayesian statistics.)

Knowledge of dependencies is not merely training material for artificial intelligence. People want to know the answers to questions such as "Does climate change have anything to do with food prices?" "Does my favourite food have anything to do with my stomach problem?" "Was the increasing salary the result of my hard working or just the inflation?" When people have gotten enough observations on weather, health, economy, or anything interesting, they may acquire some knowledge of dependencies useful for inferring causes, analysing situations, making decisions, and so on. Nowadays, observations represented as data are abundant, only if there are tools for refining these data, turning them into useful knowledge of dependencies. The computational tools that are able to learn Bayesian networks from data are one genre of such tools that no one could afford to pass.

1.3 Bayesian Networks are Indispensable to Artificial Intelligence

A machine that learns properties can do classification. A machine that learns dependencies can do inference. What can a machine that is capable of both do? It can solve puzzles, problems, manipulate things to fit your (or its own) need. When there comes a yellow cube, you tell the machine: by colour or by shape, whichever the rule is, put the yellow cube in the class of red cubes. If the machine has a certain level of intelligence as we have hoped, the machine will paint the yellow cube red to ensure that the request is fulfilled.

It would be impossible to build any forms of artificial general intelligence without Bayesian networks, because an AI that has no knowledge of dependencies can hardly be "general" in solving problems. Moreover, tools for learning Bayesian networks are vital in pursuing AGI because dependencies are so many, so complex, so disparate that handcraft and hard-coding hardly

apply.

Learning Bayesian networks ought to be done efficiently in case of artificial general intelligence. Because one cannot assume unlimited time or other resources for solving a problem or making a decision. A robot is not supposed to take actions only after it finishes its computation. A intelligent machine should not take forever to give an answer, even if it has been asked of the question of life, the Universe, and everything. The pioneers of artificial intelligence have established since the birth of the field that heuristics will always be an essential part of human-like general intelligence [30, 32, 31, 28, 29]. These are all good reasons for studying learning Bayesian networks using fast heuristics.

2 Introducing the Key Concepts

This section is about a few key concepts necessary for understanding what Bayesian networks, score functions, and score-based learning are. Score-based learning is often split into two tasks, one of which is solving a combinatorial optimisation problem. Section 2.3 is about the reasons behind approaching score-based learning in such a way.

2.1 The Essentials of Bayesian Network

There are some real world entities that can be abstracted as discrete random variables[2]. For example, someone may abstract the weather of a place as a discrete variable *weather*, which may have states *shiny*, *cloudy*, *rainy*, and so on, and the occurrence of each state is modelled by a probability distribution. How comprehensive or detailed a variable representation of a real world entity is may vary from case to case. The results of such abstraction—discrete random variables are building blocks of Bayesian networks. Between every pair of variables, there might exist a dependency. For example, *humidity* might depend on *weather*.

A Bayesian network is for representing knowledge of dependencies as a computational model [35, 36]. Given a set of discrete variables $\{x_1, x_2, \ldots, x_n\}$, let $\mathcal{P}$ denote the set of probabilities that measure the strength of the dependencies and directed acyclic graph G represents variables as nodes and dependencies as arcs, then the corresponding Bayesian network is the double $(G, \mathcal{P})$ where G is called a *structure*.

If there exists a *direct* dependency between node x_i and x_p and x_i depends on x_p; then, x_p is called a *parent* of x_i and x_i a *child* of x_p, and the arc that represents the dependency goes from node x_p to x_i; in addition, the dependency is quantified by conditional probabilities $P(x_i|x_p)$.

[2] Although some real world entities are represented as random variables in a Bayesian network, they are not necessarily random.

The set of all nodes that x_i depends on is called the *parent set* of x_i. In this thesis, a parent set is denoted as Π with possible subscripts, for example, the parent set of x_i is Π_i. If x_i has no parent, then x_i has an empty parent set denoted as $\emptyset$. Structure G that contains every node and arc is a *global structure*. A child and all of its parents together with all arcs in-between is a *local structure*. Every node has a local structure.

A node may be both a parent and a child. A node's parents' parents and so on are the *ancestors* of that node. In this thesis, an ancestor of a node is not a parent of that node. A node's children and the children's children and so on are the *descendants* of that node meaning a child is also a descendant.

All Bayesian network structures must be *acyclic*. If a cyclic relationship indeed belongs to a set of variables, then Bayesian networks might be the wrong type of model for them. For example, Bayesian networks are not for modelling a feedback loop such as the one in Figure 1. In this thesis, I assume no underlying cyclic relationship is present in the data sets used for learning Bayesian networks.

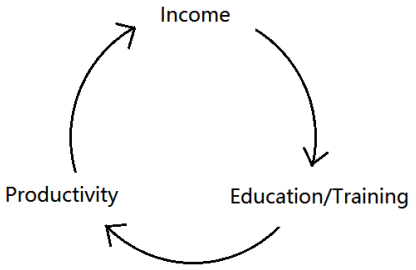

Figure 1: If a cyclic relationship such as the example exists among variables, then learning Bayesian networks might be the wrong approach.

If in a Bayesian network of n variables, every variable x_i, assumed to be discrete, has o_i number of states, then the Bayesian network has $o_1 \cdot o_2, \ldots, o_n$ possible states, each of which is called a *global configuration*. Similarly, a local structure has a number of *local configurations* and this number is the product of the numbers of states of variables in that local structure. For example, a local structure has a child x_1 and two parents x_2 and x_3, thus it has $o_1 \cdot o_2 \cdot o_3$ local configurations. A parent set has a number of *parent set configurations* which is product of the numbers of states of variables in that parent set. This number is denoted as q with various subscripts. For example, parent set $\Pi_i = \{x_2, x_3\}$ has $q_i = o_2 \cdot o_3$ parent set configurations. The number of local configurations of node x_i is often expressed as $o_i \cdot q_i$.

A conditional probability of a variable x_i is the probability of one of the possible states of the variable conditioned on a possible parent set configuration. For example, $P(0|1,2) = 0.34$ is a conditional probability of the child being in state 0 given that its two parents are in state 1 and 2

respectively, in which case $(1, 2)$ is a possible parent set configuration and $(0, 1, 2)$ a possible local configuration. Nodes with no parent have probabilities instead of conditional probabilities, for example, $P(0) = 0.3$.

Each node x_i has a number of parameters presenting its probabilities or conditional probabilities. A *parameter* is a probability or conditional probability, but the required number of parameters at the minimum is $(o_i - 1)$ or $(o_i - 1) \cdot q_i$. A global configuration also has a probability and it is the product of all probabilities and conditional probabilities of the nodes in the states that constitute the global configuration.

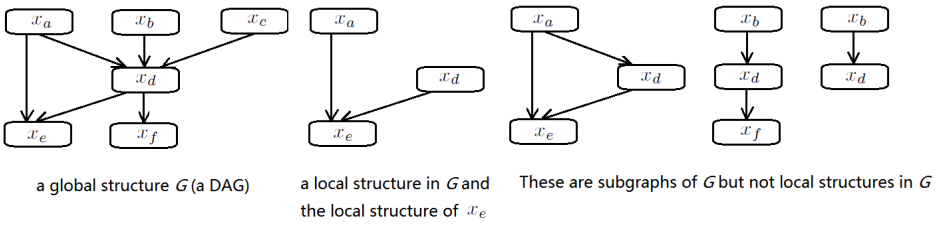

Figure 2: This is an example of Bayesian network global structure, a local structure, and three counter-examples of local structure.

Figure 2 serves as the example of this section. The Bayesian network structure G has 6 nodes representing 6 variables. In the example, node x_a is a parent of x_d and x_e, but an ancestor of x_f. Node x_d, x_f, and x_e are all descendants of x_a, but x_d and x_e are also the children of x_a. The local structure of x_e is illustrated in Figure 2. It has $o_a \cdot o_d \cdot o_e$ local configurations. The parent set of x_e is $\{x_a, x_d\}$ and it has $q_e = o_a \cdot o_d$ configurations. In Bayesian network $(G, \mathcal{P})$,

$$\begin{aligned}\mathcal{P} = \{&P(x_a), P(x_b), P(x_c),\\ &P(x_d|x_a), P(x_d|x_b), P(x_d|x_c), P(x_e|x_a), P(x_e|x_d), P(x_f|x_d)\}\end{aligned} \quad (1)$$

where each conditional P corresponds to an arc. Simplifying $\mathcal{P}$ gives

$$\mathcal{P} = \{P(x_a), P(x_b), P(x_c), P(x_d|x_a, x_b, x_c), P(x_e|x_a, x_d), P(x_f|x_d)\} \quad (2)$$

where each P corresponds to a variable or a local structure. Variable x_e has $o_a \cdot o_d \cdot o_e$ conditional probabilities (the number of $P(x_e|x_a, x_d)$ is $o_a \cdot o_d \cdot o_e$) and the corresponding node x_e has $o_a \cdot o_d \cdot (o_e - 1)$ parameters at minimum. Node x_a, x_b, or x_c has no parent, so each has a local structure of one node—the node itself and zero arc. The three subgraphs of G shown are not local structures because a local structure does not include ancestors and the arcs between parents but all parents and all arcs between the parents and the child.

2.2 Scored-based Structure Learning

Dependencies are learnt from data—observations of variables. Given a set of variables, a number of observations of them constitute a data set. Each complete observation, where every relevant variable is present with a possible state, is an *instance* and an instance can be one of the possible global configurations.

There might exist a specific tangible Bayesian network. Its set of variables is given and the dependencies among variables make the objective of learning. If there is a tangible network, then recovering this network by recovering all dependencies may be the goal. Such scenarios, for example, exist in the pertinent researches, where researchers test learning methods using synthetic data generated from existent Bayesian networks. However, one cannot take tangible Bayesian networks for granted. If a tangible Bayesian network does not exist, then, at what should we be aiming?

Bayesian networks are models of dependencies so dependencies are the objective of learning. Dependencies are part of the reality, so one can imagine an "ideal" or "perfect" model that is the closest representation of that reality. Given a set of variables, if there existed a data set D_∞ with infinite properly sampled instances, the closest thing to the reality might be an ideal Bayesian network[3] B_{ideal} and this B_{ideal} might be the Bayesian network that had optimised $P(B_{ideal}|D_\infty)$. Certainly, looking for B_{ideal} is unrealistic. But it is possible to make approximations with what is available.

With an already sampled and available data set, denoted as D, the closest approximations come from learning a network that has the highest posterior probability conditioned on D. This network can be called an *optimal network*. Let B denote a Bayesian network, then $P(B|D)$ is evaluated as follows

$$P(B|D) = \frac{P(D|B)P(B)}{P(D)} \tag{3}$$

according to Bayes' theorem. If B maximises $P(B|D)$, then B is an optimal network. There might be more than one B that have the highest posterior probability, and an optimal network is not necessarily a good model because one can deliberately manipulate prior probability $P(B)$ to achieve the highest posterior probability. I will assume prior probabilities are all reasonable—there are proper reasons backing the evaluation of prior probabilities when

[3] Phrases like "ideal Bayesian network" are not common. Publish literature may call it "true" or "underlying" network/model, which are both unsatisfactory terms because there might not be a "true" or "underlying" network. Making a model of the reality does not mean the model already exists. Saying "optimal network" is also inappropriate because an optimal network learnt with a specific optimisation procedure using a specific data set is not necessarily a faithful representation of the reality and it is possible to have multiple optimal networks. "Generating network" does not work either because it is a confusing term and it also suggests that a network already exists. Some papers use the term "distribution" rather than "network" but one cannot assume there are actual "distributions" that are affecting the reality. Rather, we use distributions to describe the reality.

such evaluation is necessary, such that finding an optimal network can be a meaningful goal.

A Bayesian network consists of a structure and a set of parameters. Structure learning precedes parameter learning because knowing the parameters depends on knowing the structure. A parameter is a probability or conditional probability. Knowing a parameter of a node implies knowing the local structure of the node. For example, knowing $P(x_a|x_b, x_c, x_d)$ implies knowing nodes x_b, x_c, and x_d are the parents of x_a hence knowing the local structure of x_a. Knowing all parameters implies knowing all local structures hence the global structure. Therefore, it is impossible to know parameters without knowing the structure in the first place.

Let G denote the structure of B and θ the parameters. Since the parameters depends on the structure, the prior probability of a Bayesian network can be express as $P(B) = P(G, \theta) = P(\theta|G)P(G)$, which can be plugged into equation (3). In addition, the probability of the data set $P(D)$ is a constant when a data set D is given. Therefore, $P(D)$ can be omitted. Also, $P(D, B) = P(D|B)P(B)$, so there come the following equations:

$$P(D, B) = P(D|B)P(B) = P(D|\theta, G)P(\theta|G)P(G) \tag{4}$$

and $P(D, B)$ is what to be maximised.

Since parameter learning depends on structure learning, learning Bayesian networks, at its core, wraps the problem of *Bayesian Network Structure Learning* or just *structure learning*. Learning the structure is the first interest, so it is necessary to find the structure G that maximise $P(D, G)$ without knowing θ. $P(D, G)$ is evaluated using the following integration:

$$P(D, G) = \int_{\theta} P(D|\theta, G)P(\theta|G)P(G)d\theta \tag{5}$$

where parameters θ are unknown; therefore, the integration is unsolvable at this stage. Other than parameters, there is also the problem of how to evaluate $P(G)$.

When the lack of knowledge of the parameters and the structure prior probability hinders evaluating $P(D, G)$, one may compute an alternative function that indicates how high $P(D, G)$ is. This alternative function is called a *score function* or a *metric*[4] (I will use both terms depending on the context) and all approaches based on score functions are *score-based learning* or *score-based structure learning*. Besides score-based learning there is constraint-based learning. Constraint-based learning is not the concern of this thesis.

A metric—a score function is denoted as M with possible subscripts in this thesis. Given a set of variables V, let $\mathbb{DAG}$ denote the space of all

[4] Since a "score" can mean both a score function and a numeric value score. In order to avoid confusion, I shall use the term *metric* for score functions or the straightforward *score function*. The term *score* is only for numeric scores.

possible directed acyclic graphs over V and $\mathbb{D}$ the space of all possible data sets from V, then a score function is a mapping

$$M : \mathbb{DAG} \times \mathbb{D} \mapsto \mathbb{R}$$

Let D denote a data set originated from V and G a network structure (a directed acyclic graph) where each node in G has to have corresponding data present in D, M takes D and G as the input and returns a real number value $M(D, G)$, which is the score of G given D, as the output.

Metrics are different in how they approach an issue. The main issue is evaluating structure prior probabilities. The differences in approaching this issue divide metrics into two classes: Bayesian metrics and non-Bayesian metrics. Bayesian metrics typically adopt Dirichlet distribution with certain assumptions in order to estimate structure prior probabilities while non-Bayesian metrics circumvent the issue, typically by taking out prior probabilities and performing maximum likelihood estimation with a term that keeps structure complexities in check. Notable members of the former are known as Bayesian Dirichlet metrics (BD metrics) which include: BDeu [7] BD, and BDe [22]; and, the latter information theoretical metrics (information criteria) which include: Akaike Information Criterion [1], Schwarz Information Criterion [40], Minimum Description Length [37], Mutual Information Tests [10], and factorized Normalized Maximum Likelihood [41]. This thesis focus on Schwarz Information Criterion, which is more often called *Bayesian Information Criterion*. Some general reasons that bolster this focus appear in Section 3 and a heuristic specific reason appears in Section 4.

Two merits make an apt metric. A *consistent* metric helps learning because it favours a structure that more accurately captures the dependencies, or the simpler structure of two possible choices that both accurately capture the dependencies, when a large and reliable data set is used [12]. A metric is *asymptotic* if it approximates $P(D|G)$ or, more usually, its logarithm with errors bounded by $O(1)$ when data set size goes to infinity [6, 20]. An asymptotic metric helps learning because the difference between $P(D, G)$ and the score of G can be made insignificant using enough data as long as $P(G)$ is reasonable and non-zero.

With an apt metric, structure learning can be approached through *scored-based structure learning*. It has two major components: a metric and a search method. The search method finds structures for the metric to evaluate then use the corresponding scores as feedback in order to improve new findings. Score-based structure learning algorithms often consist of at least two corresponding procedures.

In a way, score-based learning can be likened to standard tests for students. In college or graduate school, the staff in charge of admission do not know how well the applicants will be doing before they are admitted, the staff hence ask for standard test scores that can serve as indicators of academic potential.

In learning Bayesian networks, when accurate or precise estimations of structure prior or joint probabilities are difficult or not available, scores given by learning metrics are used as indicators.

2.3 The Two Tasks of Decomposed Learning

If a score function is composed in a way such that the *global score*—the score of a global structure is the product or sum (if the score function is logarithmic) of all of the *local scores*—the scores of the local structures in the global structure, then this score function is *decomposable*. All score functions discussed in this thesis are logarithmic (score functions being logarithmic is indeed the mainstream), hence I will only say that the global score of a structure is the *sum* of the local scores. Given data set D, global structure G and local structures $S_1 \ldots, S_n$ where $G = \bigcup_{i=1}^{n} S_i$, if M is a decomposable score function, then the following equation:

$$M(D, G) = \sum_{i=1}^{n} M(D, S_i) \tag{6}$$

holds.

With a decomposable score function, score-based structure learning can be approached in a way called *decomposed learning*[5] in this thesis. From this point, every "decomposed learning" refers to "decomposed score-based structure learning" or "score-based structure learning using decomposable score functions".

Decomposed learning is about learning local structures first then composing a global structure using the local structures learnt. There are two incentives behind decomposed learning. For one, optimal or near-optimal local structures are much more attainable than optimal global structures. For two, decomposed learning transforms a part of score-based structure learning into combinatorial optimisation problems—a well-studied class of problems with numerous available approaches. How local structures are more attainable and why the problem is only transformed but not resolved?

There can be a tremendous number of candidate global structures even with a small number of nodes [38]. For example, 10 variables lead to 4175098976430598143 possible global structures[6]. Find a few optimal global structures among such astronomical number of possibilities using merely score functions would take eons.

Local structures are way more tractable. Given n variables, for every variable, there exist 2^{n-1} possible local structures, thus the total number

[5] Phrases such as "local (structure) learning" or "learning (with) local structure" are perhaps more common in literature, but I will call score-based structure learning using decomposable metrics "decomposed learning" in this thesis because global structures rather than local structures are the objective of learning.

[6] https://oeis.org/A003024

of local structures is $n \cdot 2^{n-1}$. With decomposed learning, 10 variable lead to "merely" 5120 local scores: a cake walk. However, the number of local structures still grows exponentially. Exploring all local structures out of a greater number of nodes, say 100, still ends up taking eons. Nevertheless, decomposed learning is the lesser of two evils.

The problem is that finding local structures constitutes only half of the work. Splitting score-based structure learning into two parts is what decomposed learning is. The first part—the task of finding local structures, is called *cache construction* in this thesis. A *cache*, denoted by C with possible subscripts, is a list of doubles in the form of (local score, local structure). A cache is very often associated with a node. For example C_i denotes the cache of node x_i. When a cache is associated with a node, the doubles in the cache can also be (local score, parent set) because knowing the parent set of a node is knowing the corresponding local structure of the node.

Cache Construction

Input: A set of n nodes $V = \{x_1, \ldots, x_n\}$, a data set D consists of independently sampled instances of V, an integer $i \in \{1, \ldots, n\}$.

Output: caches $C_1, \ldots, C_n$ where each C_i is the cache of x_i and it contains an indefinite number of doubles in the form of (local score, parent set).

Although local structures are directed acyclic graphs, encoding arcs is usually unnecessary because arcs always go from the parents to the child in a local structure meaning distinguishing the child node form the parent nodes preserves complete information on arcs.

The reason why cache construction rather than finding optimal local structures becomes the first task is that the optimal local structures over a set of nodes might not form an directed acyclic graph due to possible imperfectness of the data sets (I did not assume arbitrarily large, noiseless, or error-free data sets, and no one should). Moreover, the number of local structures is exponential to the number of nodes meaning at least checking all possible local structures one by one for the optimal will be impractical in cases of a large number of variables. Cache construction is usually done under a constraint to avoid checking all possible local structures and the optimal local structures under the constraint might not form a valid structure.

After completing cache construction, the second task of decomposed learning becomes a combinatorial optimisation problem called Optimal-Structure in this thesis.

OptimalStructure

Input: Caches $C_1, \ldots, C_n$, where each cache contains a indefinite number of doubles in the form of (local score, local structure).

Output: A directed graph $G = \bigcup_{i=1}^{n} S_i$, such that G is acyclic and the global score $\sum_{i=1}^{n} s_i$ is maximised, where S_i is one of the candidate local structures in C_i and s_i the local score of S_i.

Decomposed learning does nothing to diminish the hardness of score-based structure learning, it converts a part of score-based structure learning into a task more tractable and another more familiar. Decomposed learning is the focus of this thesis. Non-decomposed score-based structure learning does exist [11], but it is relatively new and outside the scope.

3 Stairs of Approximation

This section includes an overview of two metrics: *Minimum Description Length* [37] and *Bayesian Information Criterion* [40], which are identical in almost all aspects including linear asymptotic time complexity, not needing prior probabilities, preventing unnecessarily complex structures, and so on. Their properties are advantages in decomposed learning, especially when taking heuristic approaches. Sometimes MDL and BIC are regard as one metric because their score functions are the exact negative of each other if details (parameter dimensionality, the logarithm, et cetera) are set to be identical; however, I would like to introduce them separately for their different perspectives in learning Bayesian networks.

3.1 Encoding Networks with Minimum Description Length

Minimum Description Length [37], when used in learning Bayesian networks, is a metric that balances network fitness—how close does a network fit the data sample and usefulness—how complex and hence hard to use the learnt network is [25]. It also obviate the need for considering prior probabilities [43, 25]. Understand Minimum Description Length help understand its position in learning Bayesian networks. Moreover, the knowledge is essential to encoding the network.

In the earliest pertinent work [43] in 1993, Joe Suzuki adumbrated the essentials of applying Minimum Description Length Principle to structure learning and showed that considering prior probabilities of Bayesian networks is not necessary if MDL serves as the metric. At the same time, Wai Lam and Fahiem Bacchus not only did similar work [25], but also argued convincingly that, on the one hand, metrics that "balance accuracy and complexity" provide practical value because they allocate due weight to simpler networks, which are easier to understand and compute (as supported by the pertinent

computation complexity result [14]) and hence more useful; on the other hand, an iota of hope there is for a raw data sample to retain the information adequate for recovering the "true underlying distribution" but much greater chance for an approximation; therefore, learning a Bayesian network that is faithful to the data sample may be less useful yet taking excessive tolls. In addition to all above, Lam and Bacchus also regarded evading prior probabilities as a benefit of using metric MDL.

Applying Minimum Description Length Principle to learning Bayesian networks sets aside maximising joint probabilities. The goal has become finding a network that minimises the description length of both the network itself and the data.

Given a Bayesian network structure G of n nodes a data set D of N instances and let x_i be a node from a set of n nodes $\{x_1, \cdots, x_n\}$, the description length of the node's local structure is

$$\frac{\log_2 N}{2} (o_i - 1) (q_i) \tag{7}$$

where $(o_i - 1)(q_i)$ is the number of parameters of x_i and $\frac{\log_2 N}{2}$ the number of bits required to encode a parameter [43, 19]. The number of bits used for encoding nodes is a constant given that the number of nodes is a constant, hence the description length of nodes can be omitted. The description length of the part of the data that corresponds to the local structure of x_i is

$$-\sum_j \sum_k N_{ijk} \log_2 \frac{N_{ijk}}{N_{ij}} \tag{8}$$

where N_{ijk} is the occurrence of the kth local configuration, which includes the jth parent set configuration, and N_{ij} the occurrence of the jth parent set configuration, thus $\frac{N_{ijk}}{N_{ij}}$ is the conditional probability of the kth state of x_i conditioned on jth parent set configuration (the kth state of x_i makes the kth local configuration). For example, if the local structure of x_i has three nodes including x_i, if (0,1,2) is the kth local configuration and (1,2) the jth parent set configuration, (0,1,2) appeared 250 times and (1,2) appeared 500 times in the data set, then $\frac{N_{ijk}}{N_{ij}} = P(0|1,2) = 0.5$. The number of bits $\log_2 \frac{N_{ijk}}{N_{ij}}$ is used for encoding a local configuration. The minimum description length of the whole thing—structure plus data, which is denoted as $M_{MDL}(D, S)$, can be evaluated as the following equation:

$$M_{MDL}(D, S) = \sum_{i=1}^{n} \left(\frac{\log_2 N}{2} (o_i - 1) (q_i) - \sum_j \sum_k N_{ijk} \log_2 \frac{N_{ijk}}{N_{ij}} \right) \tag{9}$$

, and since knowing N_{ijk} and N_{ij} requires going through all instances in D to count their frequencies and n is a constant, the evaluation takes $O(N)$ time .

How does the MDL score function help balance fitness and complexity? A more complex structure that fits the data better helps reduce the description length of the data but the resulting greater number of parameters will increase the description length of the structure. Therefore, the minimum description length shall come from the best balance that minimises the description lengths of the network and data together.

This thesis is not about minimising the description length, so the most important ideas about metric MDL are it is a computationally simple metric with no prior probability required and it keeps structure complexity in check. Since metric MDL is the negative of metric BIC, metric BIC has all of the properties metric MDL has.

3.2 BIC Score Function: an Approximation to Approximations

Bayesian Information Criterion[7]—firstly derived by Schwarz in 1978 [40] yet another model selection criterion from information theory adapted to serving as a metric in learning Bayesian networks, is the exact negative of metric MDL in spite of details. They share the same properties yet judge Bayesian networks from two different perspectives—minimum description length and maximum joint probability. By presenting the essential rationale behind metric BIC, the reasons are rendered lucid why it is indispensable to the heuristics in this thesis.

Metric BIC is decomposable so I will present it in a "decomposed" fashion. Since this section and the rest of the thesis are not about encoding (in bits), I will use the natural logarithm instead of the logarithm to base 2 for convenience.

Given a data set D with N instances and a structure G that consists of n local structures $S_i, \ldots, S_n$, global score $M_{BIC}(D, S)$ is evaluated using the following equation:

$$M_{BIC}(D, S) = \sum_{i=1}^{n} M_l(D, S_i) \tag{10}$$

where each local score $M_l(D, S_i)$ is evaluated by

$$M_l(D, S_i) = \sum_j \sum_k N_{ijk} \ln \frac{N_{ijk}}{N_{ij}} - \frac{\ln N}{2}(o_i - 1)(q_i) \tag{11}$$

[7] Should readers wonder why Schwarz's information criterion [40] is called BIC, checking the paper [1] about Akaike Information Criterion may lead to the source of the name. The acronym BIC agrees with Akaike's suggestion: "IC stands for information criterion and A is added so that similar statistics, BIC, DIC etc., may follow". So it is possible that, as Akaike had suggested "B Information Criterion", later literature started calling Schwarz Information Criterion BIC.

where o_i, q_i are known from S_i and n, N, N_{ijk}, N_{ij} are known from D. The evaluation takes $O(N)$ time.

Metric BIC differs from metric MDL in the perspective. Metric BIC is about maximising structure-data joint probabilities. David Heckerman [21] presented a derivation for the case in 1995. The derivation also appeared in a paper [20] co-authored by Heckerman that serves as expansion of other earlier works about structure probabilities or metric BIC [23, 6].

Since the goal is to find the structure G that maximise $P(G|D)$ and $P(D)$ is a constant, maximising $P(D, G)$ fulfills the same purpose. $P(D, G)$ is expressed as

$$P(D, G) = \int_\theta P(D|\theta, G)P(\theta|G)P(G)d\theta \tag{12}$$

where parameters θ and structure prior probability $P(G)$ might not be available. In order to proceed without considering the structure prior probability, assuming G is not impossible and hence $P(G)$ is non-zero, one can take it away through division, thus acquire

$$P(D|G) = \int_\theta P(D|\theta, G)P(\theta|G)d\theta \tag{13}$$

where parameters θ are to be handled with Gaussian approximation with a few **assumptions**: 1. the data distribution is in an exponential family, 2. $P(\theta|G)$ and $P(\theta)$ are in the same family of probability distributions, 3. parameters are mutually independent, and 4. the data set is complete. Besides, the data set has to be sufficiently large. Then, there is the following approximation:

$$P(D|\theta, G)P(\theta|G) \approx P(D|\hat{\theta}, G)P(\hat{\theta}|S)e^{-\frac{1}{2}(\theta-\hat{\theta}\mathbf{H}(\theta-\hat{\theta})^T} \tag{14}$$

where symbol $\hat{\theta}$ denotes estimators of parameters and $\mathbf{H}$ denotes the negative of the Hessian matrix of function f and $f(\theta) = \ln P(D|\theta, G)$.

Plug approximation (14) into integration (13) gives

$$P(D|G) \approx P(D|\hat{\theta}, G)P(\hat{\theta}|G) \int_\theta e^{-\frac{1}{2}(\theta-\hat{\theta})\mathbf{H}(\theta-\hat{\theta})^T} d\theta \tag{15}$$

which can be approached using Laplace's method for the multivariate case to acquire the following approximation (*Laplace approximation*)

$$P(D|G) \approx P(D|\hat{\theta}, G)P(\hat{\theta}|G)(\frac{2\pi^d}{det|\mathbf{H}|})^{\frac{1}{2}} \tag{16}$$

where, d is the dimensionality of $\hat{\theta}$ and $det|\mathbf{H}|$ is the determinant of $\mathbf{H}$. Assuming $P(\hat{\theta}|G)$ is not zero, approximation (16) can be put into the logarithmic form as the following

$$\ln P(D|G) \approx \ln P(D|\hat{\theta}, G) + \ln P(\hat{\theta}|G) + \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln det|\mathbf{H}| \tag{17}$$

which is not simple enough.

Replacing $\frac{1}{2}\ln det|\mathbf{H}|$ with $\frac{d}{2}\ln N$ renders computing $det|\mathbf{H}|$ unnecessarily (both increase with N). The term $\ln P(\hat{\theta}|G)$ and $\frac{d}{2}\ln 2\pi$ can be omitted because they can be made insignificant using enough data (they do not increase with N). Parameter dimensionality is $d = \sum_{i=1}^{n}(o_i - 1)(q_i)$ in learning Bayesian networks. Making changes accordingly to approximation (17) yields the following approximation:

$$\ln P(D|G) \approx \ln P(D|\hat{\theta}, G) - \frac{\ln N}{2}\sum_{i=1}^{n}(o_i - 1)(q_i) \tag{18}$$

which can be make more accurate with more data.

The derivation shown by David Heckerman stops here, but there are just few steps left to reach the fully detailed BIC score function. At the right hand side of approximation (18), the logarithmic posterior probability of the data set can be expressed as follows

$$\ln P(D|\hat{\theta}, G) = \sum_{t}(N \cdot \hat{g}_t)\ln \hat{g}_t \tag{19}$$

where $\hat{g}_t$ are estimated probabilities of global configurations and $N \cdot \hat{g}_t$ the occurrence of global configurations in the data set. It is unnecessary to actually estimate $\hat{g}_t$ because the right hand side is going to be decomposed as follows

$$\sum_{t}(N \cdot \hat{g}_t)\ln \hat{g}_t = \sum_{i}\sum_{j}(N_{ij}\sum_{k}\hat{c}_k \ln \hat{c}_k) \tag{20}$$

where $\hat{c}_k$ are estimated probabilities of local configurations and also the estimators of parameters. Estimators $\hat{c}_k$ can be computed using the following equation:

$$\hat{c}_k = \frac{N_{ijk}}{N_{ij}} \tag{21}$$

therefore,

$$\sum_{i}\sum_{j}(N_{ij}\sum_{k}\hat{c}_k ln\hat{c}_k) = \sum_{i}\sum_{j}\sum_{k}N_{ijk}\ln\frac{N_{ijk}}{N_{ij}} \tag{22}$$

perform a cascade of plug-ins on equations (22), (20), (19), and approximation (18) then it is clear that metric BIC is an approximation to $\ln P(D|G)$

$$\ln P(D|G) \approx \sum_{i=1}^{n}\left(\sum_{j}\sum_{k}N_{ijk}\ln\frac{N_{ijk}}{N_{ij}} - \frac{\ln N}{2}(o_i - 1)(q_i)\right) \tag{23}$$

However, the ultimate purpose of metric BIC is to be an indicator for a structure's potential in maximising $P(G|D)$. As the following approximation shows,

$$\ln P(G|D) \approx M_{BIC}(D, S) + \ln P(G) - \ln P(D) \tag{24}$$

as long as $P(G)$ and $P(D)$ are not some extreme values and the assumptions behind the approximations hold, using enough data, metric BIC can be a trustworthy indicator.

One of the biggest advantages of metric BIC (which is also MDL but negative) over Bayesian metrics is that it obviates the concern for prior probabilities, which is bolstered by both the developers [37, 40] in information theory and the earliest adopters [43, 25, 22] in learning Bayesian networks.

The scores given by metric BIC should not be taken for granted even if data are in abundance. One of the assumptions is particularly intriguing—the assumption that the data distribution is in an exponential family. One cannot assume that all data collected from the real-world scenarios agree with the assumption; for examples, weather data—observations of chaotic behaviours or ICU data—a hodgepodge of human elements, coincidences, randomness, and so on.

Nevertheless, the pith of score-based learning is to avoid the hustle of estimating structure probabilities. As long as the assumptions hold, metric BIC can be very useful because it is not only a good indicator of structure posterior or joint probability but also fast to compute. Moreover, it keeps structure complexity in check.

From the reality, to the data sample, to the structure posterior probability, to the BIC score one has to descend stairs after stairs of approximations. But metric BIC is still not simple enough for the heuristics in this thesis, yet the score function of metric BIC has already reached the simplest form it might get without losing its properties. Therefore, there is one more stair of approximation to descend. Another approximation to metric BIC shall be used to build faster heuristic algorithms. But before that, I am going to wrap up theoretical bases.

3.3 Recapitulation of the Theoretical Bases

The goal of learning Bayesian networks from data is to recover the knowledge of dependencies among random variables abstracted from real world entities and present them in the form of a Bayesian network. Learning Bayesian networks includes parameter learning and structure learning but parameter learning depends on structure learning. The goal of structure learning is to recover a structure with the highest structure-data joint probability; however, estimating probabilities might be difficult, so instead of which scores are used as indicators of structures' potential. Scores are computed by score functions

that take a structure and a data set as the input. Learning structures using score functions is called score-based structure learning.

Decomposed learning is a more tractable approach to score-based structure learning. It consists of two tasks: cache construction, which is about learning local structures, and solving OptimalStructure, which is about assembling the local structures in the optimal way. In the following sections, I will show how to approach these two tasks using a few fast heuristics.

4 Decomposed Learning Heuristic

In Section 4, I am going to discuss a few heuristics for decomposed learning. Section 4.1 is about the purposes and general aspects of heuristic. Section 4.2 is about a few essential heuristics used for decomposed learning. In Section 4.3 and Section 4.4, I will introduce the heuristics focused with the corresponding algorithmic implementations.

4.1 The Purposes of Heuristics

The adoption of heuristics is supported by established computational complexity results [12]; and, in cases of decomposed learning, OptimalStructure—a typical combinatorial optimisation problem is NP-hard [34]. However, hardness is just one of the many reasons because heuristics have their value even if the aforementioned hardness results were not true. To show their value, the purposes of heuristics have to be made clear.

Given a problem and a method of checking the solutions given to that problem, if a procedure is able to provide a correct solution in some but not all problem instances, then that procedure can be called a *heuristic* [30]. For example, A, H, W are three procedures that are able to provide solutions to a problem, say OptimalStructure (non-optimal structures are incorrect solutions), and A always finds optimal structures, H may or may not find optimal structures, while W guarantees non-optimal structures; therefore, H is a heuristic, A is an exact procedure, and W hardly exists in real-life. By the definition, approximation algorithms are also heuristics. This thesis, however, mainly discusses heuristics that are not required to provide performance guarantees.

It is hard to discuss heuristics without introducing resource limits—time, memory, steps, trials, electricity, and whatnot; because, once assumed unlimited resources, there is no need to care about heuristics. Therefore, in this thesis, I assume that all heuristics are working under tacit or stated resource limits. The heuristics discussed in this thesis subject to time limits.

Heuristics differ from exact procedures that, for some instances of the problem, it might not provide a correct solution even if resources were infinite, while exact procedures are bound to find a correct solution. However, by bringing resource limits into the picture, such difference is trimmed off. Exact

procedures may also provide a (guaranteed) correct solution within a certain resource limit. Therefore, there have to be a few other expectations for heuristics.

Given the same problem and settings, we expect a heuristic to provide a solution faster than exact procedures, and on top of which a heuristic should be able to provide a solution faster than any procedure that provides better solutions. As for the measure of "fastness", asymptotic time complexity should suffice. For example, a procedure with $O(n^2)$ complexity is considered faster than one with $O(n^3)$. In addition, incorrect solutions from a heuristic, albeit being incorrect, should have some value. What counts as "value" depends on the motivations, purposes, or requirements behind solving a problem. In cases of decomposed learning, what heuristics meet the expectations? A heuristic that helps achieve the same level of scores faster than counterparts may be considered meeting the expectations.

In order to fulfill the expectations, a heuristic has to reduce the workload required for finding a solution. The space of all possible solutions to a problem is the *search space* or *problem space* of the problem. According to Herbert Simon and Allen Newell, a heuristic avoids traversing the entire space but instead allocates a small part of the space for searching for a solution [29]. According to Marvin Minsky, a heuristic may need some *structure knowledge* of the search space in order to locate the area that is worthy of searching [28][8]. Available structure knowledge varies from problem to problem. In learning Bayesian networks, for example, valid structures in the search space are all directed acyclic graphs and every directed acyclic graph has at least one topological ordering are two pieces of available structure knowledge. Minsky also suggested that a heuristic might need to learn *patterns* from the solutions it has already tested in order to improve its future solutions. What kind of patterns is available differs from case to case. In decomposed score-based structure learning, for example, certain local structures that have high scores may show patterns.

To sum up, the purpose of a heuristic is to give a solution fast, be it correct or not, optimal or not, but the solution should have certain value. A heuristic does not check all possible solutions but only a few of them. A heuristic works towards good solutions using the knowledge that is available from the problem's search space. A heuristic may learn from its previous solutions in order to improve future solutions.

4.2 Traditional Heuristics for Decomposed Learning

This section includes an overview of the traditional heuristics frequently used in decomposed learning, some of which are for cache construction and some are for OptimalStructure.

[8] This paper has also one of the earliest appearances of the terms "Bayes net" and "Bayes network model".

A procedure for cache construction is applied to all variables. Given n variables, there are $n \cdot 2^{n-1}$ possible local structures in total. A heuristic traverses only a small part of all possibilities. A good heuristic should have a high chance in hitting optimal or at least high-score structures fast.

A very common "heuristic" is assuming a maximum in-degree k then adding all possible local structures with k or less parents to caches. This strategy indeed helps reduce the search space. However, not all local structures with k or less parents are worth keeping. Some local structures can be pruned if their scores are worse than simpler local structures [9].

In order to ensure the completion of cache construction within a given time limit, one can start the search from adding the local structures with empty parent sets to caches, then proceed to size-1 parent sets, size-2 parent sets, and so on, until the time runs out. This "heuristic" has the same drawback as using a maximum in-degree k. Not all local structures found within the time limit are worth keeping. Pruning applies too.

The above two are clearly brute-force approaches with a stop condition, which are too crude to be used in heuristics. In the spirit of heuristic, evidently low-potential local structures should be evaded without the trouble of checking them. That being said, "maximum in-degree k" and "stop at a time" work with most decomposable metrics. If cache construction is done with a clear idea of how complex local structures are supposed to be, the two approaches could provide good chances of covering high-potential or even optimal local structures.

If adopting certain information theoretical metrics, such as BIC, then it is possible to use pre-pruning strategies from the work of Cassio P. de Campos and Ji Qiang [8] to avoid checking evidently low-potential local structures. Excluding low-potential local structures reduces the work of not only cache construction but OptimalStructure as well. An algorithm that handles OptimalStructure takes caches as the input. Smaller caches lead to smaller problem instances of OptimalStructure.

Even with smaller caches, the search spaces of OptimalStructure are still monstrous. Not all directed graphs made of local structures are acyclic. So a topological ordering may be introduced as a constraint that allocates an acyclic-only search space. If cycles are out of concern, then finding a global structure may become much easier [23, 15]. Most ordering-based approaches need ordering constraints, such as Ordering-Based Search [44], order MCMC [18], partial order MCMC [33], and so on.

4.3 Guided Search using a Guiding Metric

This section is about managing caches construction using BIC^*—an approximation to metric BIC derived by Scanagatta *et al.* in their recent work [39].

In this thesis, I call BIC^* a *guiding metric* and the corresponding output

guiding scores. I would like to describe a guiding metric as the following. A guiding metric of a metric is a function that provides a real number value as a guiding score in constant time. Given the same structure and data set, the difference between their score and guiding score should be bounded.

BIC^* is a guiding metric of BIC. Let M_l denote the local score function of metric BIC, and D denote a data set. Given a node x_i, non-empty parent sets Π_i, $\Pi_i^{'}$, and $\Pi_i^{''}$ such that $\Pi_i = \Pi_i^{'} \cup \Pi_i^{''}$ and $\Pi_i^{'} \cap \Pi_i^{''} = \emptyset$, and three locals scores $s_i^{\emptyset} = M_l(D, \{x_i\})$, $s_i^{'} = M_l(D, S_i^{'})$ and $s_i^{''} = M_l(D, S_i^{''})$ where parent set $\emptyset$ belongs to local structure $\{x_i\}$, $\Pi_i^{'}$ belongs to $S_i^{'}$, and $\Pi_i^{''}$ belongs to $S_i^{''}$, then the guiding score of Π_i, which is $BIC^*(x_i, \Pi_i^{'}, \Pi_i^{''})$, is evaluated using the equation:

$$BIC^*(x_i, \Pi_i^{'}, \Pi_i^{''}) = s_i^{'} + s_i^{''} + \frac{\ln N}{2}(o_i - 1)(q_i^{'} + q_i^{''} - q_i^{'} \cdot q_i^{''} - 1) - s_i^{\emptyset} \quad (25)$$

and the evaluation takes constant time. Moreover, the difference between $BIC^*(x_i, \Pi_i)$ and $M_l(D, S_i)$ is bounded by $N \cdot ii(\Pi_i^{'}; \Pi_i^{''}; x_i)$ where $ii(\Pi_i^{'}; \Pi_i^{''}; x_i)$ is the so-called *Interaction Information* [27].

The main purpose of a guiding metric is to help discover high-potential local structures faster, which is fulfilled by giving high-potential structures priorities that are measured by guiding scores. A guiding metric helps with cache construction especially when there are time limits. However, a guiding metric is not necessarily a heuristic by itself because it does not tell whether certain structures should be discarded meaning all local structures will be covered eventually if there is no time limit.

The value of BIC^* comes from the fact that one can use BIC^* to gain insight into the potential of local structures, thus avoid traversing local structures indiscriminately. BIC^* is the most vital component of Insightful Searching [39].

Insightful Searching is the epitome of heuristic algorithms for cache construction. Multiple heuristics are implemented in Insightful Searching. First, Insightful Searching is an anytime algorithm working under a time limit. It uses BIC^* for guidance and hence discovers better local structures within the time limit. Second, it has both the pre-pruning [8, 39] and the post-pruning procedures [9]. The pre-pruning procedure is customised to work with BIC^* and it averts cache construction from checking evidently low-potential local structures. The post-pruning procedure deletes low-potential structures overlooked by the pre-pruning procedure from caches. Last but not least, Insightful Searching does not have to work with a maximum in-degree k.

The input of Insightful Searching consists of a data set D of N instances, a set of n nodes $\{x_1, \ldots, x_n\}$; an positive real number t as the running time limit. The output consists of n caches $C_1, \ldots, C_n$ corresponding to the nodes where each cache contains an indefinite number of triples in the form of (local

score, parent set, entropy) which are sorted by the local scores in descending order.

The Insightful Searching algorithm uses both BIC score function and BIC^* thus has two corresponding subroutines. The first subroutine is denoted by LocalScore(data set, parent set), and its output—local scores are denoted by s with possible subscripts. LocalScore also *saves* the entropies of parent sets or nodes for pre-pruning decisions (Algorithm 1 line 18). Entropies are denoted by η with possible subscripts, for example, η_{Π_i} is the entropy of Π_i and η_{x_i} of x_i. All output of LocalScore are added to a *closed list*, which is a cache that saves entropies. The closed list of x_i is denoted by C_i. Generally speaking, LocalScore(data set, parent set) is just a procedure that computes BIC scores with the additional instructions on saving entropies. The second subroutine is **BIC***(closed list, parent set, parent set) that computes guiding scores, which may be denoted as s^* with possible subscripts. The output of **BIC*** are doubles in the form of (parent set, guiding score).

Insightful Searching executes its procedure for each node x_i. The procedure has three stages as follows.

1. Initialisation and search preparation. (pseudo-code line 2-11)
2. Guided search. (line 12-25)
3. Prepare the result as the cache of x_i. (line 26-31)

There are three major steps in Stage 1: initialisation and search preparation, which are as follows.

1. Initialise two lists: closed list C_i and open list O_i, and a variable *time* of positive real number that records the time usage.
2. Compute the score of the empty and all size-1 parent sets of x_i with entropies saved, then add the parent sets together with their scores and entropies to C_i.
3. Compute guiding scores using function **BIC*** and the data saved in C_i for all size-2 parent sets, then add the size-2 parent sets and their guiding scores to O_i

Up to this point, the algorithm has C_i with BIC local scores of empty and all size-1 parent sets and O_i with the guiding scores of all size-2 parent sets. From now on, the algorithm will proceed to Stage 2 without going back.

There are six major steps in Stage 2: guided search, which are listed as the following. The algorithm will repeat these steps until O_i is exhausted or the time limit is reached.

Algorithm 1 Insightful Searching

INPUT: a data set D, a set of nodes $\{x_1, \ldots, x_n\}$, a positive real number t
OUTPUT: lists $C_1, \ldots, C_n$ where each C_i is the cache of x_i and it contains an indefinite number of triples in a structure such as (local score, parent set, entropy)

for every node x_i in $\{x_1, \ldots, x_n\}$ **do**
  $time$ is the time passed since starting the procedure for x_i
  initialise lists C_i and O_i
  LocalScore$(D, \emptyset) \rightarrow (s_\emptyset, \emptyset, \eta_{x_i}), N, o_i$ where N is #instances in D, o_i is #states of x_i, and η_{x_i} is the entropy of node x_i
  $C_i \leftarrow (s_\emptyset, \emptyset, \eta_{x_i})$
  for every node x_j in $\{x_1, \ldots, x_n\}$ that is not x_i **do**
    $C_i \leftarrow (s, \{x_j\}, \eta_{\{x_j\}}) \leftarrow$ LocalScore$(D, \{x_j\})$
  end for
  for every pair of nodes x_j, x_k in $\{x_1, \ldots, x_n\}$ that are not x_i **do**
    $O_i \leftarrow (\Pi, s^*) \leftarrow$ **BIC***$(C_i, \{x_j\}, \{x_k\})$ where $\Pi = \{x_j\} \cup \{x_k\}$
  end for ▷ size-2 parent sets' guiding scores computed
  while O_i is not exhausted **and** $time < t$ **do**
    $O_i \rightarrow (\Pi, s^*)$ where s^* is the highest in O_i where $\Pi = \Pi' \cup \{x'\}$
    $C_i \leftarrow (s, \Pi, \eta) \leftarrow$ LocalScore(D, Π)
    for every node x_k in $\{x_1, \ldots, x_n\}$ that is not x_i **and** $x_k \notin \Pi$ **do**
      $C_i \rightarrow \eta_{x_i}, \eta_{\Pi'}, \eta_{\{x'\}}$ ▷ extract entropies
      make parent set $\Pi_+ = \Pi \cup \{x_k\}$ where Π_+ has q configurations
      if $s^* + \frac{\ln N}{2}(o_i - 1) \cdot q \leqslant N \cdot \min\{\eta_{x_i}, \eta_{\Pi'}, \eta_{\{x'\}}\}$ **then**
        if Π_+ is not in O_i **and** C_i **then**
          $O_i \leftarrow (\Pi_+, s^*_+) \leftarrow$**BIC***$(C_i, \Pi, \{x_k\})$
        end if
      end if
    end for
    delete (Π, s^*) from O_i
  end while
  for every pair of triples (s_a, Π_a, η_a) **and** (s_b, Π_b, η_b) in C_i **do**
    if $s_a \leqslant s_b$ **and** $\Pi_b \subset \Pi_a$ **then**
      delete (s_a, Π_a, η_a) from C_i
    end if
  end for
  sort C_i by scores in descending order
end for
return $C_1, \ldots, C_n$

1. Extract the parent set Π with the highest guiding score, denoted as s^* from O_i, Π was constructed from the union $\Pi^{'} \cup \{x^{'}\}$.

2. Compute the BIC local score of Π and save the entropy of Π during the computation, then add Π with its local score and entropy to C_i.

3. Expand Π by adding an extra node x_k. Let Π_+ denote expansion so $\Pi_+ = \Pi \cup \{x_k\}$.

4. Determine whether Π_+ should be saved for consideration or discarded using the following inequality:

$$s^* + \frac{\ln N}{2}(o_i - 1) \cdot q \leqslant N \cdot \min\{\eta_{x_i}, \eta_{\Pi'}, \eta_{\{x'\}}\} \tag{26}$$

   where q is the number of configurations of Π_+ and entropies $\eta_{x_i}, \eta_{\Pi'}$, and $\eta_{\{x'\}}$ corresponding to node x_i, parent set $\Pi^{'}$ and $\{x^{'}\}$ are extracted from C_i. If the inequality is satisfied, the algorithm proceeds to checking whether Π_+ has been added to the two lists already. If Π_+ satisfies the inequality and is not present in the two lists, the algorithm computes the guiding score of Π_+ and adds the result to O_i; otherwise, Π_+ will be discarded.

5. Go to step 2 if there is still x_k left for expanding Π, otherwise delete the record of Π from O_i then proceed to the next step.

6. Go to step 1 if O_i is not yet empty or there is still time, otherwise proceed to Stage 3.

There are three major steps in Stage 3: prepare the result as the cache of x_i, which are as follows.

1. For every parent set in C_i, if there exist a subset with a higher score, delete the parent set.

2. Sort C_i by local scores in descending order.

3. Return C_i as the cache of x_i.

Finally Insightful Searching return $\{C_1, \ldots, C_n\}$ as the output. Entropies can be deleted from caches if necessary. Deleting caches will use time but save memory.

4.4 Using Orderings as Queues

This section is about ordering-based heuristics and how they find quick solutions for OptimalStructure. The orderings discussed in this thesis are not necessarily topological orderings because there are two different

usages, one of which use orderings as queues. I will briefly introduce the conventional usage first and proceed to detailed introduction and discussion of the unconventional usage.

An *ordering* is a list of nodes in a specific hierarchy, denoted as a lone symbol $\prec$. In this thesis, I will call the position of a node in an ordering a *rank*; therefore, an ordering is also a set of node-rank pairs. Given an ordering $\prec$, if x_a is the ith node in $\prec$, then node x_a is paired with rank r_i; if two nodes, x_a, which is paired with rank r_i, and x_b, which is paired with rank r_j are in a relative positioning such that $i < j$, then x_a is called a *higher node* of x_b while x_b a *lower node* of x_a. Given a node as the point of reference, the side where higher nodes reside is the *higher side* and lower nodes the *lower side*. The ordering defined here should be discriminated from a *topological ordering* that it does not specify the direction of arcs.

The following is an example ordering:

r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8
x_3	x_6	x_1	x_8	x_7	x_5	x_4	x_2

where the upper row shows the ranks listed from high to low, and the lower row shows the nodes arranged in an arbitrary order. Node x_3 has rank r_1 meaning it is the highest ranked node. Node x_1 has the rank r_3 while x_8 has the rank r_4 meaning x_1 is a higher node of x_8 and x_8 a lower node of x_1. Let node x_7 be the point of reference, the higher side has nodes x_3, x_6, x_1, and x_8, while the lower side has nodes x_5, x_4, and x_2. Also, x_3 has only the lower side and x_2 only the higher side. Later when I am going to discuss ordering-based algorithms, I will use the rank of a node and symbols such as r_i or $*r_i$ to refer to the node in that rank. Consequently, I say "given (an ordering of node) $\{r_1,\ldots,r_n\}$" instead of "$\{x_1,\ldots,x_n\}$" because nodes are not necessarily in an order such as $1, 2, 3, \ldots n$.

The conventional usage of ordering is called *ordering-as-constraint* in this thesis. Using orderings as constraints appeared as early as the earliest literature about learning Bayesian networks [23, 15]. An ordering becomes a topological ordering when combined with the following rule: arcs can only go from a higher node to a lower node or the vice versa but not both. For convenience, I will specify the rule as "only high to low" in this thesis. A topological ordering can be used as a constraint—only the directed acyclic graph that has the topological ordering may become a candidate structure. Therefore, only the local structures that are compatible with the topological ordering may be selected as parts of a global structure. A local structure is compatible with a topological ordering if and only if the parent nodes are all higher nodes of the child node in the ordering. Any learning method that uses a topological ordering and can only produce topological ordering compatible structures can be categorised as an ordering-as-constraint method. Most ordering-based methods are ordering-as-constraint methods.

An ordering constraint is able to narrow the range of search down to acquire an acyclic-only space, and finding the best structure in this space is not necessarily NP-hard [15]; however, the best structure in the narrower space might not be as good, needless to say the optimal. Ordering constraints are "quite restricting" Scanagatta *et al.* commented [39]. They proposed ASOBS, which does not work with ordering constraints.

ASOBS adopts a different usage of orderings that I call *ordering-as-queue*. I will firstly present the ASOBS algorithm and then proceed to contrasting "ordering-as-queue" with "ordering-as-constraint" such that a better apprehension of the former can be acquired. This paves the way to finding an apt ordering strategy for ASOBS.

ASOBS differs from all ordering-as-constraint methods that it uses a Boolean matrix, denoted as T, as the constraint. Given an ordering of nodes $\prec = \{r_1, \ldots, r_n\}$ where r represent nodes of which the positions are not clear, ASOBS always starts with the highest node r_1 then proceeds one by one to r_n. For each node r_i, ASOBS will select the best parent set sanctioned by Boolean matrix T for T knows which node is which node's parent or ancestor. Every entry in T can be either $True$ or $False$. If entry T_{r_i,r_j} is $True$, then r_i has become either a parent or an ancestor of r_j since the previous selections and r_j will not be permitted to become a parent or an ancestor of r_i in the later selections. The content of T is the constraint that guarantees acyclic structures, and it will be updated after every selection. Updating this Boolean matrix T appears to be the key to ensuring acyclic structures.

The input of ASOBS consists of $\prec = \{r_1, \ldots, r_n\}$ and caches $C_1, \ldots, C_n$ where each cache contains a indefinite number of doubles in the form of (local score, parent set) sorted by local scores in descending order. (The entropy data added by Insightful Searching can be ignored.) For convenience, I assume the caches are already arranged according to the ordering such that C_i is the cache of r_i.

The output of the algorithm is a list of n triples in the form of (local score, node, parent set).

ASOBS has two stages as follows.

1. Initialisation.
2. Acyclic selection.

In the first stage, the algorithm prepares the constraint for achieving "acyclic selection"—a Boolean matrix T. It sets every entry in the Boolean matrix as $False$ because no node has parents initially and entries such as T_{r_i,r_i} will always be *False* since self-loops are not allow either. A list G of triples (local score, node, parent set) will also be initialised. G will be the output.

Algorithm 2 Acyclic Selection Obeying Boolean-matrix Sanctioning

INPUT: a set of nodes in an ordering $\prec = \{r_1, \ldots, r_n\}$ and corresponding caches $C_1, \ldots, C_n$ where each C_i is a list of doubles such as (score, parent set)

OUTPUT: a list G of triples in the form (local score, node, parent set)

initialise n by n Boolean matrix T where all entries are set to $False$
initialise local structure list G
for every node r_i in $\{r_1, \ldots, r_n\}$ **do**
    Find the (s_i, Π_i) from C_i such that score s_i is highest among the parent sets that satisfy **either** Π_i is empty **or** T_{r_i, x_P} is $False$ for every node x_p in Π_i
    $G \leftarrow (s_i, r_i, \Pi_i)$ where r_i is the ith node but not the rank!
    if Π_i is not empty **then**
        initialise list *descendants*
        descendants $\leftarrow r_i$ ▷ add r_i to the list
        for every node r_j in $\{r_1, \ldots r_n\}$ **do**
            if T_{r_i, r_j} is $True$ **then**
                descendants $\leftarrow r_j$ ▷ add all descendants of r_i to the list
            end if
        end for
        initialise list *ancestors*
        AncestryTracing$(G, ancestors, \prec, r_i, r_i)$
        for every node x_a in *ancestors* and x_d in *descendants* **do**
            if T_{x_a, x_d} is $False$ **then**
                $T_{x_a, x_d} = True$
            end if
        end for
    end if
end for
return G

procedure AncestryTracing$(G, ancestors, \prec, x_k, r_i)$
    $G \rightarrow \Pi_k$ where Π_k is the parent set of x_k
    for every node x_p in Π_k such that $x_p \notin ancestors$ **do**
        ancestors $\leftarrow x_p$ ▷ add an parent of x_k to the list
        for every node r_j in $\{r_{i-1}, r_{i-2}, \ldots, r_1\}$ **do** ▷ r_j has a parent set
            if x_p is r_j **then** ▷ if x_p also has a parent set
                AncestryTracing$(G, ancestors, \prec, x_p, r_i)$
            end if
        end for
    end for
end procedure

In the second stage, the algorithm repeats the following steps for every node r_i one by one according to the ordering $\prec$, which acts as a queue here. The steps are as follows.

1. The algorithm finds the (s_i, Π_i) from C_i such that score s_i is highest among the parent sets which satisfy either Π_i is empty or T_{r_i,x_P} is $False$ for every node x_p in Π_i, then adds the finding to G. If Π_i is empty, then it skips step 2-4 and proceeds to the selection for the next node in the ordering. (pseudo-code line 4-6)

2. The algorithm initialises a *descendants* list and adds r_i and all of its descendants recorded by the Boolean matrix T to that list. (line 7-13)

3. The algorithm initialises an *ancestors* list, it adds the parents of r_i to the list (because the parents of r_i are ancestors of the descendants of r_i) and searches for all of the ancestors of r_i through Depth-First Search by tracing the ancestry of each parent that is also a higher node of r_i. Only r_i and its higher nodes have made selections hence gotten parents so far. All ancestors of r_i found will be added to the *ancestors* list. This is the key step because new parent/ancestor-child relationships have to be registered in Boolean matrix T in order to avoid cycles. (line 14-15 and all lines of procedure AncestryTracing)

4. For every node r_a in *ancestors* list and r_d in the *descendants* list, the algorithm sets T_{r_a,r_d} as $True$, if the entry is not $True$ already. (line 16-20)

At last, ASOBS returns G as the output.

The algorithm takes $O(nzc+n^2)$ time so the asymptotic time complexity is $O(zc)$, where z represents the number of parent sets in a cache, the number may differ from cache to cache; c represents the number of records in the Boolean matrix to be checked before a parent set is sanctioned, t, too, varies from selection to selection, n is the number of nodes (a constant), thus $O(n^2)$ is time needed for ancestry tracing and updating the Boolean matrix.

ASOBS specifies what to do when an ordering is provided[9], but nothing about what kind of orderings may suit it or how to acquire them. What makes a good ordering for ASOBS then? Ordering strategies for ASOBS is one of the open questions to consider. The authors of ASOBS did not verify in their paper [39] the ordering strategies that could have suited ASOBS. Is it possible to use existent strategies devised for ordering-as-constraint

[9] There appear to be confusion in published literature [26], suggesting that ASOBS was used as a control group in the experiments about ordering strategies. The ASOBS algorithm itself tells nothing about how to acquire orderings - it takes an ordering as a part of the input. It is not appropriate to make a contrast between ASOBS with a system that includes an ordering strategy.

methods? To investigate this matter, I will first contrast ordering-as-queue with ordering-as-constraint.

The fundamental difference between ordering-as-constraint and ordering-as-queue, as hinted by the nomenclature, is on how they use orderings. In ASOBS, an ordering is a queue for commencing local structure selections, hence the naming—ordering-as-queue. The ordering in the input only specifies the priority of each node - it is not a constraint! The real constraint is the content of ancestor/parent-descendant-relationship Boolean matrix, which will become stricter as more selections are done. The highest node has the most freedom of choice meaning all local structures in its cache are available, because the Boolean matrix is mostly "empty" (all entries are $False$) at the beginning, while the lowest node *might* has the least freedom, it might have only a few choices left in its cache. This constraint imposed by the Boolean matrix is in a way similar, or I would rather say "opposite", to the constraint imposed by a topological ordering. In cases of ordering-as-constraint, the highest node has the least freedom of choice—it can only select the local structure with the empty parent set yet the lowest node has the most freedom.

Boolean matrix constraints are dynamic while ordering constraints are static. Boolean matrices develop and the search spaces converge. This is best illustrated using animations. Since that is very difficult here, I will use a series of van diagrams (Figure 3).

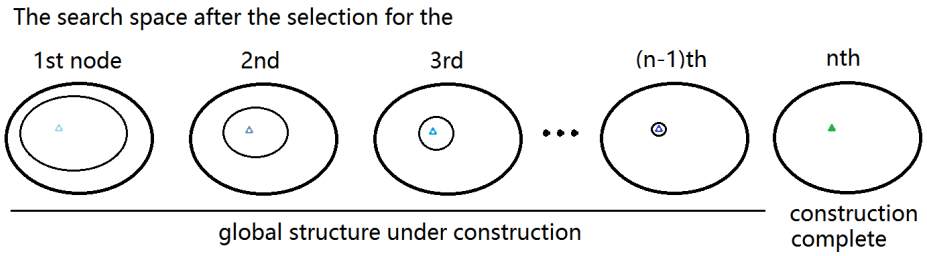

Figure 3: A search space in ordering-as-queue is not static - it is shrinking (converging) while a global structure is emerging. An ordering determines how the search space should shrink.

The dynamic-static contrast leads to the biggest difference in selecting parent sets. In ordering-as-queue, the selections made for higher nodes will shape the Boolean matrix, and how it is shaped affects the selections for lower nodes. In order words, the selections made for lower nodes depend on the selections made for higher nodes. In cases of ordering-as-constraint, selections are mutually independent.

Selections being not mutually independent will attenuate the effectiveness of all strategies that improve structures by making adjustment to orderings while exploiting the decomposability of metrics. In cases of ordering-as-queue,

after an ordering has been adjusted, all selections of the highest ranked of which the node has been changed to the lowest rank in the ordering have to be redone. Let r_i be the highest rank changed—its node has been changed from x_a to x_b, then the algorithm has to redo the selections from r_i to r_n. Figure 4 illustrates an example, the initial ordering has had two nodes swapped their positions. After the change, in cases of ordering-as-constraint, only two nodes x_8 and x_1 need to have selections redone, since every node else's set of higher nodes stays the same meaning its best parent set under ordering constraint stays the same; while in cases of ordering-as-queue, x_8 as well as all of its lower nodes have to go through re-selections, because after x_8 selects another local structure with a different set of parents, Boolean matrix T will deviate from its past development.

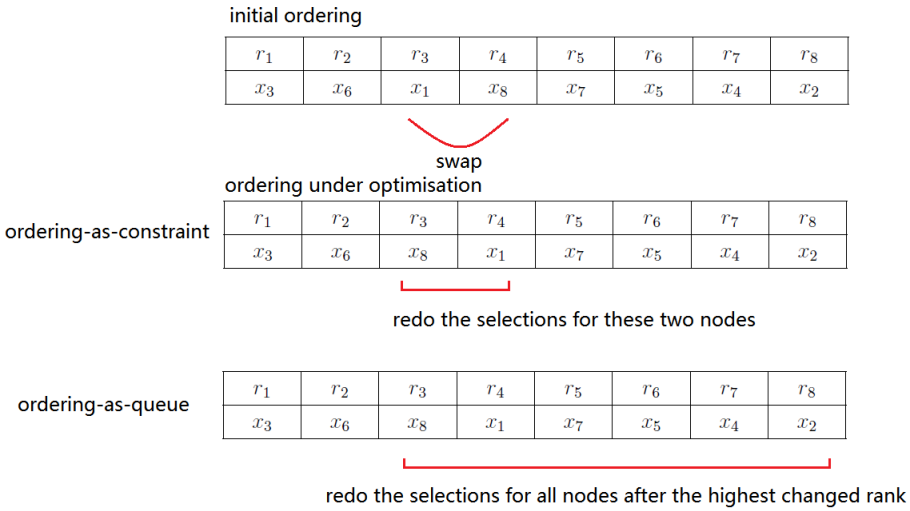

Figure 4: The amounts of work after the ordering adjustment are different.

Lastly, the known ordering-as-queue algorithm—ASOBS has a disadvantage against ordering-as-constraint algorithms incurred by non-independent selections. ASOBS has to run in serial: using orderings as queues is not only its feature but also its restriction.

For all those stringent conditions one has to comply, an ordering strategy for ordering-as-queue might not come easy. I will go through the key points I have notices.

Where to start the optimisation is a tough question, because the higher the rank changed, the greater the possibility of deviation from the past development, the more computation may be incurred. To predict whether this deviation will be a win or lose might require more work than what can be done within polynomial time. It is like the science fictions where protagonists

went back in time to alter history, the further back they had gone, the more they subjected to Butterfly Effect, the more unpredictable the consequences would have become.

An ordering in cases of ordering-as-queue contains much less useful information. Since an ordering is not a constraint, it by itself tells nothing about possible parent-child relationships, which is why all ordering strategies that exploit such information, such as order MCMC [18] and its variants, will be rendered useless. In ordering-as-constraint, a node's higher nodes are known to be its only pool of parents, hence adjusting the rank of that node adjusts the pool. The adjustment is itself a piece of useful information that is available before the new global score is computed. In ordering-as-queue, adjustment leads to implicit changes in the Boolean matrix and the result of the change—the global score becomes available only after all due re-selections are performed. Peers who have also studied ASOBS remarked in their paper that their ordering strategies for ordering-as-constraint algorithms had provided merely "marginal" improvement when yoked with ASOBS [45].

The peculiar mechanism and confounding behaviour of ordering-as-queue make it hard to agree with Scanagatta *et al.*'s arguments [39] that ASOBS will always outperform Ordering-Based Search—an typical ordering-as-constraint algorithm [44]. Their arguments were formalised as Theorem 3 and Theorem 4 in the corresponding paper. Theorem 3 says that ASOBS will outperform Ordering-Based Search using the same ordering because there is no ordering constraint to limit the pool of parent sets to one side of a node. However, this is not necessarily true. Given the same ordering $\prec$ of two nodes as follows:

r_1	r_2	r_3
x_3	x_2	x_1

and with this ordering, one can construct a maximum in-degree 2 Bayesian network where two independent node x_2 and x_3 are the parents of x_1, the original structure is the highest scored structure; however, the data set might contains errors such that x_3's best parent set is $\{x_1\}$ and x_2's best the empty set. In this case, OBS has a chance to find the best structure while ASOBS has none. Theorem 4 says ASOBS works better with the greedy swap strategy than Ordering-Based Search. However, considering the fact that, in cases of ordering-as-queue, there are more due re-selections after a swap. In a toe-to-toe competition between ASOBS and Ordering-Based Search, it is not appropriate to assume that Ordering-Based Search will wait for ASOBS to complete its re-selections before taking the next move.

Given that Theorem 3 and 4 in the paper [39] do not necessarily hold, the assumption that an ordering that is good for Ordering-Based Search is also good for ASOBS has lost its bolsters. This means an ordering strategy customised for ASOBS is necessary. ASOBS has been shown to outperform Ordering-Based Search in several empirical studies [39, 45, 26], which suggests that ASOBS indeed has its potential. Therefore, investigating what orderings

are good for ASOBS and how to find them will be the theme for the rest of this thesis.

5 Ordering Strategies for Ordering-as-queue

Ordering strategies for ASOBS is an incipient research topic. The full potential of ordering-as-queue is yet to be discovered. Because ordering-as-queue works so differently, one has to also think differently in search of an apt strategy. This thesis focus on heuristics, therefore I will think in the spirit of heuristic.

This section includes a brief look at an published ordering strategy [45] for ASOBS, then a presentation of a new ordering strategy with the corresponding algorithms. Both strategies use the idea of *score ranking* but work differently on other aspects. The new ordering strategy was tested with results, which appear in Section 5.3.

5.1 A Strategy in Existence

This section includes a brief look at an existing ordering strategy—*BestFirst-Based ordering generation* devised by Walter Perez Urcia and Denis Deratani Mauá [45]. Published literature that discusses ordering strategies for ASOBS is wanting.

Another concept—*score ranking* may be easily confused with ranks of nodes, yet it is important for understanding the following content, so I will elaborate it here. A *ranking* is a list of scores sorted in descending order, which is similar to rankings in competitive games. The "score" here might be a local score or a global score. Every score has a *place* in a ranking. For example, in the ranking: {-96, -97, -98, -99}, *score* -97 has *place* 2 and *score* -99 has *place* 4. For simplicity, I will not say "1st place", "2nd place", "3rd place", and so on, like people usually do in reporting a game.

BestFirst-Based ordering generation is an *initialisation* heuristic to be specific. Its purpose is to construct a set of orderings for ASOBS in advance of running ASOBS. The key idea is using inversed places. In constructing an ordering, BestFirst-Based ordering generation selects local structures according to scores and then add scores to a ranking. The place of each score will be inversed to acquire a probability, the higher the place, the higher the probability. The acquired probabilities are then used for generating orderings.

BestFirst-Based ordering generation might be too complicated to be used as an ordering improvement strategy. It needs a sorting subroutine, it involves computing and using probabilities, it requires a cap on maximum in-degree; and, its "parent set should not contain visited node" condition is confounding. Its asymptotic time complexity is $O(n^2k)$ where k is the integer specifying the maximum in-degree.

Using probabilities might be a feasible idea for an initialisation strategy but not an *improvement* strategy that aims at improving scores on the run. Pairing a node with a rank using probabilities may bring conflicts because two nodes might be appointed to the same rank. For example, a conflict appears when a node x_a had been assigned to rank r_i, because it had had the highest probability to be in r_i, but later, x_b was also assigned to r_i for the same reason. Certainly, it is possible to mark r_i as occupied and let x_b reconsider its position; however, when probabilities are used thus drawing a rank becomes a stochastic process, the next rank dawn for x_b is unnecessarily not r_i, and if x_b has the highest probability to be in r_i, then there might be a long wait until x_b eventually finds an available rank. As available ranks run out, conflicts might occur more and more often, compromising the performance. Another possible counter measure is to reduce the probabilities of all remaining nodes being paired with an occupied rank to zero. This might require more computation on the re-normalisation of probabilities.

5.2 A Computationally Simple Strategy

Since optimising an ordering is tricky in ordering-as-queue, it might be better to forsake the idea. Instead, an algorithm can test a large amount of orderings then learn some patterns, so there comes the idea of *Randomised Pairing Greedy Weight*. It uses inversed places of global scores, but as weights instead of probabilities. Its purpose differs from BestFirst-Based ordering generation that it is not an initialisation strategy that generates orderings in advance, but an improvement strategy that makes improvements on the run. It can be easily formulated as an anytime algorithm.

An ordering tells nothing about parent-child relationships does not mean it contains no useful information. Each ordering leads to a score. With a large amount of orderings tested, one can learn a set of node-rank pairs that leads to a higher global score.

With n nodes and ranks here are n^2 pairs. Each pair will be assigned a weight computed with inversed average node places of the global scores of the orderings where that pair has appeared. Pairing a rank with two nodes is forbidden, thus finding an optimal set of pairs with the highest total weight might be hard. Pairing will be done in a randomised order with the greedy strategy. Therefore, I call the strategy and the corresponding algorithm Randomised Pairing Greedy Weight with the acronym RPGw. That being said, the algorithm uses the strategy only in half of the cases because it needs some random orderings to learn from. Next, I will present the algorithm.

The input of the algorithm includes an ordering $\prec$, a real number *score*, which is the score procured by the ordering $\prec$, a list *Ranking* where scores automatically appear in descending order; two tables $Frequency$ and $SumofPlaces$ that records frequencies of pairs and summations of places, for example, $Frequency(x_i, r_j) = 3$ means node x_i has been paired with r_j

for 3 times, and $SumofPlaces(x_i)(r_j) = 4000$ means the sum of the places of the global scores procured when x_i was paired with rank r_j is 4000. The algorithm returns no output, it can only manipulates the data in the input.

In the ordering, r_j is the rank of the jth node but which one the jth node is remains unclear, so notation $*r_j$ shall represent the jth node. For example, in the following ordering

r_1	r_2	$r_3,\ldots,r_n$
x_b	x_a	$x_c,\ldots,x_d$

r_1 is the rank of x_b but $*r_1$ represents x_b.

Algorithm 3 Randomised Pairing Greedy Weight

This algorithm is for providing ASOBS with orderings. It manipulates the data in the input and has no output. The symbol r in this algorithm refers only to ranks and $*r$ is used to represent nodes.

INPUT: a list *Ranking*, two n by n tables $Frequency$ and $SumofPlaces$, a real number *score*, an ordering $\prec$ of nodes

OUTPUT: None.

1: insert *score* into *Ranking* and acquire the *place* of the score
2: **for** $j = 1, \ldots, n$ **do**
3: $Frequency(*r_j, r_j)$ += 1 ▷ $*r_j$ is the node paired with rank r_j in $\prec$
4: $SumofPlaces(*r_j, r_j)$ += *place*
5: **end for**
6: initialise *coin*
7: flip *coin*
8: **if** *coin* shows *head* **then**
9: randomly shuffle $\prec$
10: **else**
11: initialise a queue of ranks $Q = \{r_1, \ldots, r_n\}$
12: randomly shuffle Q
13: initialise list $rankless = \{x_1, \ldots, x_n\}$
14: **for** each rank r_j in Q **do**
15: search a x_h in *rankless* that maximises $w_{x_h,r_j} = \frac{Frequency(x_h,r_j)}{SumofPlaces(x_h,r_j)}$
16: modify $\prec$ by pairing rank r_j with node x_h
17: delete x_h from *rankless*
18: **end for**
19: **end if**

The algorithm has three major steps as follows.

1. The algorithm inserts *score* into *Ranking*, it thus acquires the place of *score*. This step takes at most $O(l)$ time where l is the number of scores (same as the number of orderings tested) in *Ranking*. (pseudo-code line 1)

2. The algorithm uses the place of *score* and the ordering $\prec$, which has full information on current node-rank pairs, to update the two tables. This step takes at most $O(2n^2)$ time. (line 2-5)

3. The algorithm initialises a fair coin then then flips it(line 6-7). If the coin lands on $Head$ the algorithm randomly shuffles the ordering, which takes $O(n)$ time using the fastest shuffle algorithm (line 8-9). If coin lands on $Tail$, the algorithm proceeds to the following two steps (line 10-18) :

   (a) The algorithm initialises another list of ranks Q called a *queue of ranks* and shuffles it randomly. This Q will be used as the queue for pairing nodes with ranks. The algorithm initialises another list of nodes *rankless* to memorise which nodes have not yet gotten an rank.

   (b) For each rank r_j from the first in Q to the last, the algorithm selects the node x_h from the node *rankless* with the biggest weight. The weight w_{x_h,r_j} of pair x_h-r_j can be computed using the following equation:

$$w_{x_h,r_j} = \frac{Frequency(x_h, r_j)}{SumofPlaces(x_h, r_j)} \tag{27}$$

   which is actually the inverse average place of pair x_h-r_j. The algorithm then deletes x_h from *rankless*. Shuffle takes $O(n)$ and pairing takes $O(n^2)$.

The algorithm takes at most $O(3n^2 + n + l)$ time, giving asymptotic time complexity $O(n^2 + l)$ (the size of the score ranking list is independent of the number of nodes and it is possible or even more likely to have cases where $n^2 < l$).

Finally, RPGw joins forces with ASOBS (**Algorithm 4**). Together they improve the solutions to OptimalStructure. It is formed as an anytime algorithm. The input consists of n *caches* and a time limit t. The output is a list G of local structures that constitute a global structure.

The combined algorithm can be easily parallelised. If there are multiple processors available, each processor can run an instance of ASOBS using an ordering provided by RPGw. After finishing the test of an ordering, RPGw processes the result and provides a new ordering for the next test.

5.3 A Controlled Experiment on Ordering Strategies

It had been reported that "randomly generated ordering" was the recommended strategy [45]. Therefore, generating random orderings would have been served as the control group.

Algorithm 4 ASOBS+RPGw Combined Algorithm

This algorithm combines ASOBS and RPGw as a system that handles OptimalStructure. Function **ASOBS** is **Algorithm 2** and Function **RPGw** is **Algorithm 3**

INPUT: caches $C_1, \ldots, C_n$

OUTPUT: a list G of local structures that consists a global structure

initialise *time* which records the time passed
initialise local structure lists G, S
initialise score list *Ranking*
initialise n by n tables $Frequency$ and $SumofPlaces$
initialise ordering $\prec$
initialise *score*, $topscore = -\infty$
while $time < t$ **do**
    $S =$ **ASOBS**$(caches, \prec)$
    score = the summation of local scores saved in S
    if $score > topscore$ **then**
        $topscore = score$
        $G = S$
    end if
    RRGw$(score, Ranking, Frequency, SumofPlaces, \prec)$
end while
return G

Algorithm 5 ASOBS using randomly generated orderings

This algorithm uses random orderings and ASOBS to compute solutions for OptimalStructure. Function **ASOBS** is **Algorithm 2**.

INPUT: caches $C_1, \ldots, C_n$

OUTPUT: a list G of local structures that consists a global structure

initialise *time* which records the time passed
initialise local structure lists G, S
initialise ordering $\prec$
initialise *score*, $topscore = -\infty$
while $time < t$ **do**
    $S =$ **ASOBS**$(caches, \prec)$
    score = the summation of local scores saved in S
    if $score > topscore$ **then**
        $topscore = score$
        $G = S$
    end if
    randomly shuffle $\prec$
end while
return G

The experiment was about comparing"ASOBS+RPGw" (**Algorithm 4**) with "ASOBS+random ordering" (**Algorithm 5**). The objective was to compare their aptitudes for improving scores and their highest scores achieved within a given time limit.

The software and hardware environments were identical for both groups. The software—**Balestasis**[10] used for the experiment had a complete implementation of all algorithms needed for the experiment, and it was running in experiment mode during the experiment. The hardware environment included a Virtualbox virtual machine running 64bit Linux, which was hosted by a personal computer with an AMD 8-core CPU. The virtual machine had access to all physical cores and 9600 Megabyte of random access memory.

Table 1 shows the experiment settings. The experiment had involved 6 data sets [11] from 5 Bayesian networks of different backgrounds and complexities. They were: Alarm [5], ANDES [13], Diabetes [2], Munin [3], Random2000-4-5000 (abbreviated as Rand2K in Table 1) [39]. Unfortunately, it was unclear whether the experimental data were following the assumptions required by metric BIC. However, this could not have affected the validity of the experiment results since ASOBS and RPGw do not take raw data as their input.

For each data set, the experiment software had first produced caches using Insightful Searching. The implementation in the software was slightly different from the original algorithm in the pre-pruning condition $s^* + \frac{\ln N}{2}(o_i - 1) \cdot q \leqslant \min\{\eta_{x_i}, \eta_{\Pi'}, \eta_{\{x'\}}\}$ where there was no N multiplier. This could not have affected the validity of the results because Insightful Searching had been the same for both groups. The non-strict time limit (Table 1 Time limit CC column) on cache production was 5 seconds per node (actual running times could have been less than 5 seconds) for all data sets, except "Rand2K" where the limit was 60 seconds (actual running times were around 77 seconds, the last rounds of searching had started within the limit but finished 17 seconds later).

After cache construction, the software ran "random ordering + ASOBS" and " RPGw + ASOBS" each for 3 times. The strict time limit (Table 1 Time limit OS column) was the same for each run of the same data. These time limits had been set so that they roughly scaled according to the network sizes, which had assured that there would have been enough time for each run to test enough orderings. These time limits were strict meaning only the orderings that were found within the limits would have been saved.

Results were plotted as graphs (Figure 5). In the graphs, x-axes are scales for the number of orderings tests. The reason for this presentation is, on the one hand, time scales of testing orderings (the unit is second per ordering)

[10] Currently available at Sourceforge https://sourceforge.net/projects/balestasis/

[11] Alarm (10000 instances) and Diabetes (1000 instances) data sets were acquired from https://www.cs.york.ac.uk/aig/sw/gobnilp/data/; Andes, Diabetes (5000 instances), Munin, Random2000-4-5000 data sets were acquired from http://blip.idsia.ch/jobs/supp/

Table 1: Experiment Settings

Network	Network specifications			#Instances	Time limit(s)	
	n	$\#\theta$	MID	N	CC	OS
Alarm	37	509	4	10000	5	8
ANDES	223	1157	6	5000	5	128
Diabetes	413	429409	2	1000	5	320
Diabetes	413	429409	2	5000	5	320
Munin	1041	80592	3	5000	5	1800
Rand2K	2000	?	4	5000	60	3600

Symbol n stands the number of nodes and $\#\theta$ the number of parameters. "MID" stands for max in-degree of the network. Symbol N stands for the number of instances. Column CC shows the per node time limits (not strict) for running Insightful Searching and column OS the time limits (strict) for running ASOBS+RPGw or ASOBS+random ordering strategy.

Table 2: Experiment Results

Network	ASOBS+RPGw		ASOBS+Random	
	Avg. $\#\prec$	Avg. score	Avg. $\#\prec$	Avg. score
Alarm	63753	-106187	190217	-106194
ANDES	207388	-573938	1767648	-576290
Diabetes(1k)	296630	-556488	2138844	-559261
Diabetes(5k)	297630	-3966413	3327682	-3977037
Munin	215512	-5001171	2266433	-5040984
Rand2K	93692	-9506640	275027	-9542713

There were 3 runs for each setting. "Avg. $\#\prec$" stands for the average number of orderings tested and "Avg. score" stands for the average score.

had a huge range—-from 0.000001s/o to 0.1s/o and it was inconvenient to track events happening in microseconds; on the other hand, computation was supposed to concentrate on ASOBS, because ASOBS has $O(zc)$ asymptotic time complexity (z stands for the sizes of caches, which are exponential of n in worst cases) while ordering strategies has $O(n)$ (random) or $O(n^2 + l)$ (RPGw); therefore, the amount of orderings tested should be adequate as a speed indicator.

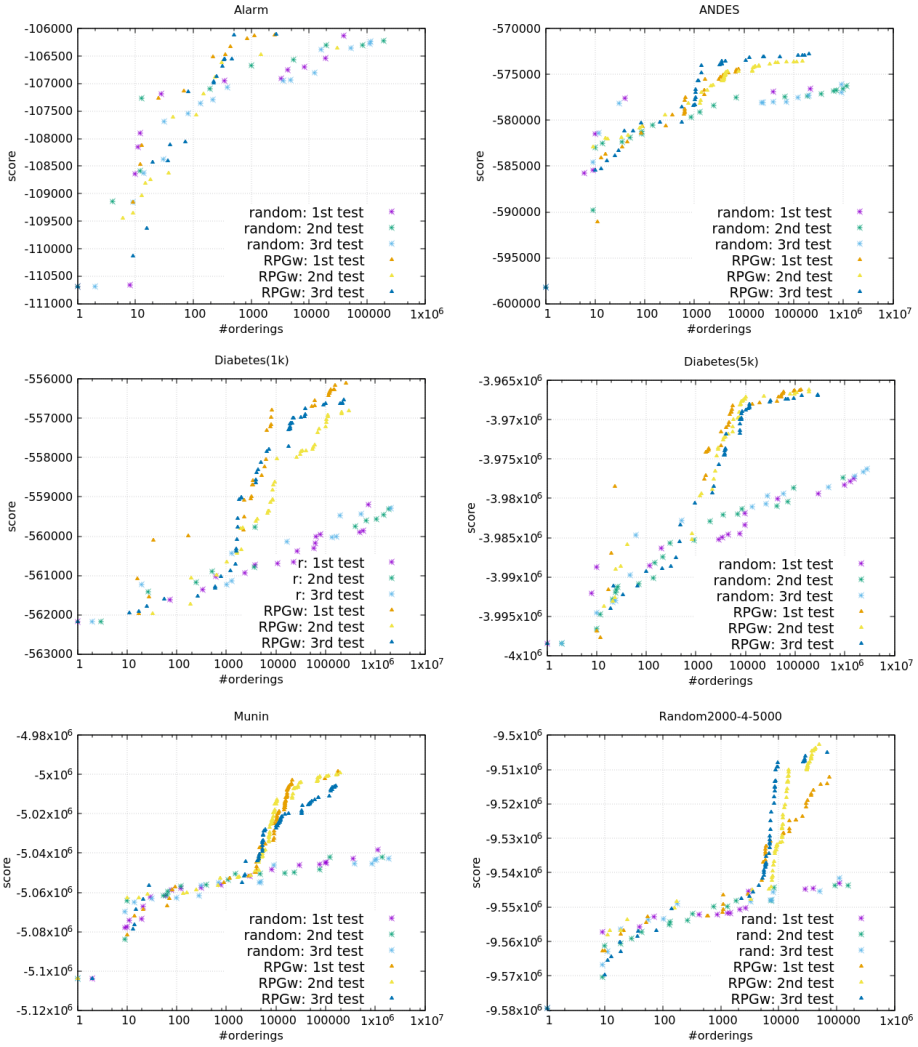

Figure 5: Experiment results plotted as graphs where RPGw (triangles) is shown being able to learn from existing solutions in order make improvements.

RPGw is slower than random shuffling. Therefore, given the same set of caches and time limits, RPGw cannot test as much ordering as random

shuffling. However, as Table 2 shows, ASOBS+RPGw had achieved higher scores with the same time limits in all settings in this experiment.

In the graphs (Figure 5), every point represents an event where an improvement occurred. For example, a point (1000, -106200) suggests an improved score -106200 were found at approximately the 1000th ordering tested. If readers want to find out roughly the time used to reach an improvement, it can be calculated this way: calculated the average speed of testing orderings then used the speed value and a point in the plot to figure out the time used for achieving the improvement that point is representing. For example, if point (1000, -106200) represents an improvement by RPGw achieved at the 1000th ordering tested, the average ordering testing speed is roughly 64000/8 = 8000 (orderings per second), then the time used is roughly 1000/8000 = 0.125s. (Beware of the inaccuracy of this estimation!)

The graphs show that RPGw indeed had a stronger tendency toward making improvements. It had improved scores faster and more frequently (triangular dots are denser) once it had started picking patterns out of procured solutions. As a contrast, randomised ordering strategy was susceptible to stagnation.

6 Finale

6.1 Conclusion

In this thesis. I have shown essential theories and fast heuristics for learning Bayesian networks. Among those, there are two algorithms, Insightful Searching and ASOBS, that have brought two ideas into the picture: guided search and ordering-as-queue, the latter of which remains deficient in apt ordering strategies. In hope of complementing the set of heuristic algorithms for fast learning, I proposed Randomised Pairing Greedy Weight strategy with an algorithmic implementation, which has brought more or less potential for further advancement—nothing but small steps.

6.2 Discussion

Now there is a set of heuristic algorithms that learns Bayesian networks from raw data pretty fast. For what can it be used? Its results are not accurate or precise enough for reliability-critical applications but it might be useful in some time-critical applications, for example, robotics or video game AI. The current set misses the component for handling incomplete data sets. If anyone were to fill this gap, it would have had more practical usages. Networks learnt from raw data might not be directly applicable to real-world applications, but they can be used as references for more accurate and precise learning approaches. For example, networks learnt from raw data may be used for deriving prior probabilities required by Bayesian metrics.

Table 3: Fast Learning by Insightful Searching+ASOBS+RRGw

Network	Input		BIC score and gap			Time(s)
	n	N	solution	optimal	gap	
Alarm	37	1000	-11943	-11783	1.4%	1.11
Mildew	35	1000	-59048	-57238	3.2%	3.90
Water	32	1000	-14053	-13290	5.7%	1.00

At last, it can be used for learning networks of "thousands of variables" [39], which is its original purpose.

6.3 Outlook

There is still much to be done. The entire algorithm set may become truly useful after acquiring the capacity for handling incomplete data sets. Using orderings as queues is in its early stage and requires more research. When more apt ordering strategies are developed for an ordering-as-queue algorithm such as ASOBS, we may expect a toe-to-toe competition between ordering-as-queue and ordering-as-constraint factions. RPGw is a simple yet primitive strategy - it has space for improvement. For example, using a fair coin might not be the best choice.

At last, I am going to present a few more results (Table 3). These results are not from a controlled experiment and hence have little scientific value. They serve more as a demonstration.

Table 3 shows the performance of Insightful Searching, ASOBS [39], and RPGw implemented in **Balestasis** software as a complete Bayesian network learning system for learning three networks: Alarm [5], Mildew[12], and Water [24]. The running time column show the time taken from issuing learning commands to the completion of encoding Bayesian networks (parameters estimated) as files saved on the disk drive. The BIC scores were compared with known optimal scores[13] solved by Gobnilp[16, 17, 42] using caches constructed during another work [4]. This was achieved with a personal computer of 2017. After improvement, refinement and proper implementation, the algorithms presented in this thesis may eventually help develop strong artificial intelligence. A robot may be able to reveal dependencies and perform inference with merely a quick glance.

[12] Developed by Finn V. Jensen, Jørergen Olesen, and Uffe Kjærulff.

[13] Optimal scores are available at https://www.cs.york.ac.uk/aig/sw/gobnilp/

Acknowledgements

I am grateful to Mikko Koivisto and Petri Myllymäki for supervising this thesis, and to Juho-Kustaa Kangas for sharing experience in experimenting. I thank my parents for supporting my studies including this thesis in Finland.

References

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

[2] Andreassen, S., Hovorka, R., Benn, J., Olesen, K. G., and Carson., E. R.: *A model-based approach to insulin adjustment.* In Proceedings of the 3rd Conference on Artificial Intelligence in Medicine, pages 239–248, 1991.

[3] Andreassen, S., Jensen, F. V., Andersen, S. K., Falck, B., Kjærulff, U., Woldbye, M., Sørensen, A. R., Rosenfalck, A., and Jensen, F.: *MUNIN — an expert EMG assistant.* In Desmedt, John E. (editor): *Computer-Aided Electromyography and Expert Systems*, chapter 21. Elsevier Science Publishers, Amsterdam, 1989.

[4] van Beek, P. and Hoffmann, H.: *Machine Learning of Bayesian Networks Using Constraint Programming*, pages 429–445. Springer International Publishing, Cham, 2015.

[5] Beinlich, I. A., Suermondt, H. J., Chavez, R. M., and Cooper, G. F.: *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, 1989.

[6] Bouckaert, R.: *Bayesian belief networks: From construction to inference.* PhD thesis, 1995.

[7] Buntine, W.: *Theory refinement on bayesian networks.* In *Proceedings of the Seventh Conference (1991) on Uncertainty in Artificial Intelligence*, pages 52–60, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[8] de Campos, C. P. and Ji, Q.: *Efficient structure learning of bayesian networks using constraints.* Journal of Machine Learning Research, 12:663–689, July 2011.

[9] de Campos, C. P., Zeng, Z., and Ji, Q.: *Structure learning of bayesian networks using constraints.* In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 113–120, New York, NY, USA, 2009. ACM.

[10] de Campos, L. M.: *A scoring function for learning bayesian networks based on mutual information and conditional independence tests.* Journal of Machine Learning Research, 7:2149–2187, December 2006.

[11] Chen, E. Y., Choi, A., and Darwiche, A.: *Learning Bayesian Networks with Non-Decomposable Scores*, pages 50–71. Springer International Publishing, Cham, 2015.

[12] Chickering, D. M., Heckerman, D., and Meek, C.: *Large-sample learning of bayesian networks is np-hard.* Learning Locally Minimax Optimal Bayesian Networks, 5:1287–1330, December 2004.

[13] Conati, C., Gertner, A. S., VanLehn, K., and Druzdzel, M. J.: *On-Line Student Modeling for Coached Problem Solving Using Bayesian Networks*, pages 231–242. Springer Vienna, Vienna, 1997.

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

[15] Cooper, G. F. and Herskovits, E.: *A bayesian method for the induction of probabilistic networks from data.* Machine Learning, 9(4):309–347, 1992.

[16] Cussens, J., Haws, D., and Studený, M.: *Polyhedral aspects of score equivalence in bayesian network structure learning.* Mathematical Programming, pages 1–40, 2016.

[17] Cussens, J., Järvisalo, M., Korhonen, J. H., and Bartlett, M.: *Bayesian network structure learning with integer programming: Polytopes, facets, and complexity.* 2016.

[18] Friedman, N. and Koller, D.: *Being bayesian about network structure. a bayesian approach to structure discovery in bayesian networks.* Machine Learning, 50(1):95–125, 2003.

[19] Friedman, N. and Yakhini, Z.: *On the sample complexity of learning bayesian networks.* In *Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence*, UAI'96, pages 274–282, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[20] Geiger, D., Heckerman, D., and Meek, C.: *Asymptotic Model Selection for Directed Networks with Hidden Variables*, pages 461–477. Springer Netherlands, Dordrecht, 1998.

[21] Heckerman, D.: *A tutorial on learning with bayesian networks.* Technical report, March 1995.

[22] Heckerman, D. and Chickering, D. M.: *Learning bayesian networks: The combination of knowledge and statistical data.* In *Machine Learning*, pages 20–197, Boston, USA, 1995. Kluwer Academic Publishers.

[23] Herskovits, E.: *Computer-based Probabilistic-network Construction.* PhD thesis, Stanford, CA, USA, 1991. UMI Order No. GAX92-05646.

[24] Jensen, F. V., Kjærulff, U., Olesen, K. G., and Pedersen., J.: *Et forprojekt til et ekspertsystem for drift af spildevandsrensning (an expert system for control of waste water treatment - a pilot project).* Technical report, Aalborg, Denmark, 1989.

[25] Lam, W. and Bacchus, F.: *Learning bayesian belief networks: An approach based on the mdl principle.* Computational Intelligence, 10:269–293, 1994.

[26] Lee, C. and van Beek, P.: *Metaheuristics for score-and-search bayesian network structure learning.* 2017.

[27] McGill, W. J.: *Multivariate information transmission.* Psychometrika, 19(2):97–116, 1954.

[28] Minsky, M. L.: *Steps toward artificial intelligence.* Proceedings of the IRE, 49(1):8–30, Jan 1961.

[29] Newell, A.: *Human Problem Solving.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1972.

[30] Newell, A., Shaw, J. C., and Simon, H. A.: *Empirical explorations of the logic theory machine: A case study in heuristic.* In *Papers Presented at the February 26-28, 1957, Western Joint Computer Conference: Techniques for Reliability*, IRE-AIEE-ACM '57 (Western), pages 218–230, 1957.

[31] Newell, A., Shaw, J. C., and Simon, H. A.: *Elements of a theory of human problem solving.* Psychological Review, Vol 65(3):151–166, 1958.

[32] Newell, A. and Simon, H. A.: *The logic theory machine: A complex information processing system.* 1956.

[33] Niinimäki, T., Parviainen, P., and Koivisto, M.: *Structure discovery in bayesian networks by sampling partial orders.* Journal of Machine Learning Research, 17(1):2002–2048, January 2016.

[34] Papadimitriou, C. H. and Steiglitz, K.: *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[35] Pearl, J.: *Bayesian networks: A model of self-activated memory for evidential reasoning.* 1985.

[36] Pearl, J.: *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[37] Rissanen, J.: *Paper: Modeling by shortest data description.* Automatica, 14(5):465–471, September 1978.

[38] Robinson, R. W.: *Counting labeled acyclic digraphs*, pages 239–273. New York, 1973.

[39] Scanagatta, M., de Campos, C. P., Corani, G., and Zaffalon, M.: *Learning bayesian networks with thousands of variables.* In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (editors): *Advances in Neural Information Processing Systems 28*, pages 1864–1872. Curran Associates, Inc., 2015.

[40] Schwarz, G.: *Estimating the dimension of a model.* The Annals of Statistics, 6(2):461–464, March 1978.

[41] Silander, T., Roos, T., Kontkanen, P., and Myllymäki, P.: *Factorized normalized maximum likelihood criterion for learning bayesian network structures.* 2008.

[42] Studený, M.: *How matroids occur in the context of learning bayesian network structure.* In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, UAI'15, pages 832–841, Arlington, Virginia, USA, 2015. AUAI Press.

[43] Suzuki, J.: *A construction of bayesian networks from databases based on an mdl principle.* In *Proceedings of the Ninth International Conference on Uncertainty in Artificial Intelligence*, UAI'93, pages 266–273, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[44] Teyssier, M. and Koller, D.: *Ordering-based search: A simple and effective algorithm for learning bayesian networks.* In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, pages 584–590, Arlington, Virginia, USA, 2005. AUAI Press.

[45] Urcia, W. P. and Mauá, D. D.: *Improving acyclic selection order-based bayesian network structure learning.* 2016.