Induction of Bayesian Networks with *a priori* Domain Knowledge

Master's Thesis
Cognitive Artificial Intelligence
Utrecht University

Carsten Riggelsen

June 28, 2002

1st supervisor: Prof. Dr. Ir. L.C. van der Gaag
2nd supervisor: Dr. V. van Oostrom

Acknowledgement

Thanks to all (old) study friends from Århus Universitet: Tandlægehøjskolen, Dat/Mat, Mat/Øk, for making studying in Århus interesting and fun. Thanks to my supervisors Linda v.d. Gaag, Dirk Thierens and Vincent v. Oostrom. Also thanks to my CKI friends Janneke, Job, Linda, Martijn and Tinka for making 4 years of studying in The Netherlands worthwhile.

Til min familie; min mor, min far og specielt min lillesøster, Helle.

© 2002 Carsten Riggelsen

Contents

1 Introduction **3**
1.1 Knowledge formalism . . . 3
1.2 Knowledge acquisition . . . 4
1.3 Knowledge induction . . . 4
1.4 Relation with Cognitive Artificial Intelligence . . . 5

2 Bayesian Networks **6**
2.1 DAGs and d-separation . . . 6
2.2 Bayesian networks . . . 8
2.3 Equivalence . . . 9
2.4 General considerations . . . 11

3 Induction **12**
3.1 Inductive methods . . . 13
3.1.1 Frequentist . . . 14
3.1.2 Bayesian . . . 16
3.1.3 Comparison . . . 17
3.2 Structure priors . . . 20
3.2.1 The PML case . . . 20
3.2.2 The PB case . . . 20
3.3 Search methods . . . 21
3.3.1 Search strategies . . . 21
3.3.2 Score decomposition . . . 22

4 Priors **23**
4.1 Domain Experts . . . 24
4.2 Knowledge Structures . . . 24
4.3 Chunks . . . 25
4.3.1 Combination . . . 26
4.3.2 Chunks as priors . . . 27
4.3.3 Chunks as building blocks . . . 30

5 Implementation & search **32**
5.1 Evolutionary algorithms . . . 33
5.2 EA & chunks . . . 33
5.2.1 Mutation & recombination . . . 34
5.2.2 Selection & reinsertion . . . 35
5.3 EA implementation . . . 36
5.4 Experimental settings . . . 38
5.5 Results & discussion . . . 39
5.5.1 EA results . . . 40
5.5.2 General remarks . . . 44

6 Conclusion **46**

Bibliography **48**

A Implementational issues **49**
A.1 Edge representation 49
A.2 Acyclic and connected tests 49
A.3 Population pools 50
A.4 Mutation points 50

Chapter 1

Introduction

This thesis is about *Bayesian Networks* (sometimes referred to as Probabilistic Networks or Causal Networks) from an *inductive* point of view under the assumption that we have some kind of *a priori* knowledge at hand before we apply induction. In more elaborate terms, in this thesis we consider the following:

> Assume that we deal with a (limited) knowledge domain and assume that we have a database containing cases describing outcomes of situations handled according to the rules and methodologies valid within that domain. Furthermore assume that we *a priori* have a rough idea of how some of the relations are or could be (whether causally related, correlated, dependent, etc.) between the essential concepts handled within the domain. We want to build a Bayesian network that adequately models the knowledge domain in such a way that the resulting network can be used for reliable inference and decision support. This we accomplish by using the *a priori* knowledge as a (partial) best guess of how the resulting (final) Bayesian network could or will look, and apply an inductive algorithm feeding it with the cases in the database such that the algorithm can revise, refine or consolidate the prior knowledge, and potentially fill in any missing information. We thus don't start our induction from scratch, but we are actually giving qualified guesses of how we expect the final network to look.

Before we look deeper into this in the forthcoming chapters, we briefly sketch a few points regarding the background and motivation of the issues mentioned above:

1.1 Knowledge formalism

Bayesian Networks can be seen as an important element of modern *Expert Systems* (a more general term is Knowledge Based Systems). The traditional Expert System dates back to the late 70ties, early 80ties and usually belong to the group called rule-based systems that employ production-rules (if... then... else-rules) as a way of reasoning, using forward and backward chaining to combine the different rules.[1] This type of system might suffice when dealing with simple domain knowledge. A refinement of the rule-based systems is to add certainty factors to the production rules as a method of reasoning with (un)certainty, and combining the reasoning algorithms with some algebra-based method that use these factors as basis of certainty calculations. MYCIN is an often cited example of an early Expert System that

[1] Some expert systems are based on other principles, but they are limited to a specific domain or are too simple to be of any real interest.

employs such calculations. In general though it turns out that there are several (profound) problems with this sort of extension to the normal rule-based systems.

The modern Expert System approach takes a less rigid view of knowledge compared to the traditional approach. The natural way of representing knowledge (or at least more natural than production rules) is to view it as a mesh of relations between essential knowledge items. This is conveniently done in a Directed Acyclic Graph (DAG), where the nodes represent the essential knowledge items ('concepts', 'variables') and the directed arcs represent the relations between these nodes (from parent node to child node). This relation can be causal, but does not necessarily have to be. The certainty factors is in this framework called subjective probabilities and represent for each relation at hand given an assignment of parent node(s) what the probability/chance/risk is that its child node has a certain assignment. The Bayesian network can now be interpreted as being 'just' a graphical representation of a joint probability distribution. The inference algorithms in this modern approach is therefore tightly coupled to the already well founded probability theory.

1.2 Knowledge acquisition

Compared to production rules, Bayesian networks are an intuitively appealing and easy to use tool to communicate problems or questions with the domain expert when eliciting domain knowledge. Its clarity makes it easier for domain experts to grasp which can positively affect the process of acquiring domain knowledge. Knowledge acquisition is however still generally problematic, also within the Bayesian network knowledge formalism, and the acquisition is considered far from trivial and knowledge elicitation is known to be prone to misinterpretations and errors.

Knowledge acquisition for Bayesian networks essentially means capturing the domain concepts or variables and their values, eliciting relations between those variables, and finally eliciting the subjective probabilities along the relations. Within the Bayesian network framework full-fledged (semi)-automatic knowledge acquisition tools and programs still have to be developed. (Semi)-automatic knowledge acquisition programs for specific rule-based systems exist (MOLE, YAST), but they are very limited in scope, not flexible with respect to uncertainties, and are not generally applicable to different knowledge domains.

1.3 Knowledge induction

Induction is the process of reasoning from a finite number of cases to the general 'rule'. This way of generalizing of course does not necessary result in *the* correct 'rule'. Datamining and different forms of machine learning are all based on induction principles. The finite number of cases can be viewed as examples of what the 'rule' deduced in the 'past'.

In many knowledge domains records are kept of the outcomes of an episode that has been handled according to the 'rules' and principles valid within that domain (for instance patient records in medicine). If we assume that we have a database of our domain with some records, such that each record has attributes that coincide with our domain concepts or variables, we can use this database to induce the right 'rule' that created the data in the first place. The 'rules' express (indirectly) the relations we need in order to model the domain in terms of a Bayesian network.

In this way we could ease (optimistically: skip) the troublesome knowledge acquisition task mentioned above. Several methods and algorithms have been proposed that induce Bayesian networks from databases. Unfortunately the idea of fully automatically inducing Bayesian networks is to many users (often experts themselves) reason enough not to use the induced

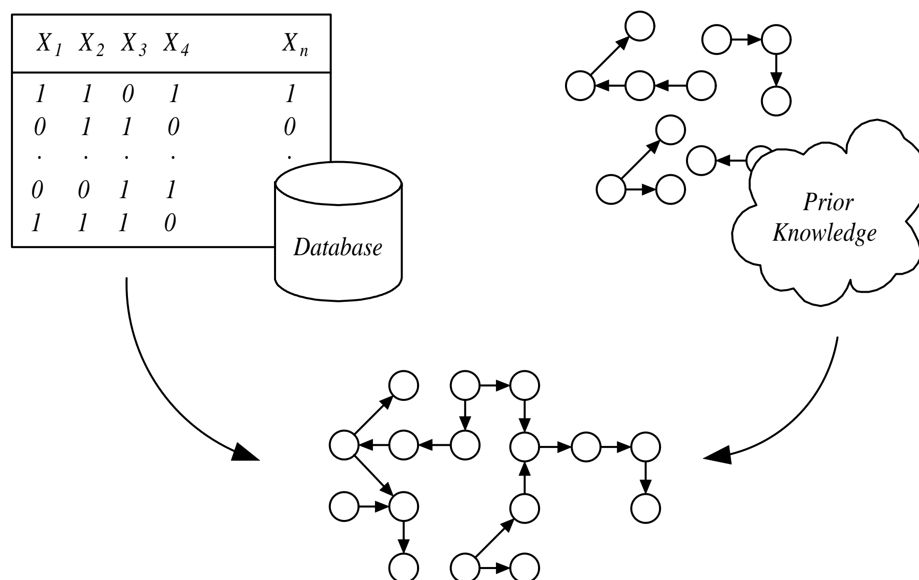


Figure 1.1: The combination of data and prior knowledge

system for decision support. This is especially the case with life critical expert systems. Of course the suspicion that the model may not be fully reliable is quite legitimate considering the fact that induction does not a priori produce the correct generalizations. We may indeed end up with a faulty model when we consider data alone.

To overcome the problem of possibly wrongly induced Bayesian networks and still keeping the time spent on knowledge acquisition at a minimum we suggest first to elicit all knowledge pieces from the expert that are of utmost importance, obvious, or otherwise essential for the final Bayesian network. The second stage is to apply an inductive algorithm that refines, revises, consolidates or complements this *a priori* knowledge on the basis of our database. Figure 1.1 illustrates that idea.

1.4 Relation with Cognitive Artificial Intelligence

Expert systems have traditionally been closely related to the AI-community. The so called *knowledge acquisition bottleneck* has been seen as a serious setback of the whole idea of knowledge-based systems. Since the upcoming of Bayesian networks new interest in the area has emerged. Bayesian networks is a better way of representing (expert) knowledge compared to other knowledge representation formalisms, but in the Bayesian network formalism the knowledge acquisition bottleneck persists. The cognitive aspect of knowledge acquisition is very important [6] as cognitive models (how do experts perceive uncertainty? causality? independence?) help us gain insight in how experts think and reason. We can combine AI ideas, especially machine learning techniques, and cognitive models to develop methods and techniques that can be used to make better (reliable) knowledge-based systems that better reflect or mimic the real nature of decision making.

Chapter 2

Bayesian Networks

Before we begin we first define our notation. With uppercase letters X_i, U we denote a (discrete) stochastic variable. The corresponding values of variables we denote with lowercase letters x_i, u. When we talk about a set of variables we use bold uppercase $\mathbf{X}, \mathbf{U}$. The lowercase bold letters denote an assignment of the set of variables, $\mathbf{x}, \mathbf{u}$. We also refer to this as a configuration of the variables in $\mathbf{X}$ and $\mathbf{U}$.

With Ω_X we denote the state space of a stochastic variable X. A set of variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ has outcome space $\Omega_{\mathbf{X}}$. We use $\Pr(U = u) = \Pr(u)$ to denote the subjective probability that $U = u$ where $\Pr(\cdot)$ obeys the basic three axioms of probability theory.

2.1 DAGs and d-separation

We will start off with the definitions of acyclic graphs and d-separation:

Definition 1 (Directed graph) *A* directed graph *is a pair* $Gr = (\mathbf{X}, \bar{E})$ *where* $\mathbf{X}$ *is a finite set of variables called* nodes *and* $\bar{E}$ *is a set of ordered pairs* (X_i, X_j) *called* arcs. X_i *is called the* parent *of* X_j, *and* $\mathbf{P}_{X_j}$ *is the* parent set *and is the set of all the immediate parents of* X_j.

Definition 2 (DAG) *Let* $Gr = (\mathbf{X}, \bar{E})$ *be a directed graph. A* path *is a sequence of nodes,* $X_1, \ldots, X_m, m > 0$ *such that* $(X_i, X_{i+1}) \in \bar{E}$ *with* $1 \leq i < m$. *Gr is called an* acyclic directed graph (DAG) *if it contains no paths* $X_1, \ldots, X_m, m > 1$ *such that* $X_1 = X_m$.

A DAG is basically a graph with no way of returning to the "starting point" if we admit to the direction of the arcs. Note that in the above definitions there is a one-to-one correspondence between the nodes and the stochastic variables.

Before we will define the concept of a Bayesian network we will look at DAGs from a practical (knowledge) modelling point of view. Let the variables X_i be the essential concepts of our knowledge domain. For ease of exposition we will interpret an edge $X_1 \rightarrow X_2$ as a causal relation between X_1 and X_2 — thus X_1 causes X_2.

We will look at the three connections of X_1, X_2, X_3 depicted in figure 2.1. The *serial* connection says that X_1 causes X_2 that in turn causes X_3. Obviously knowledge about X_1 will cause a change in our beliefs in X_2 that will cause a change in our beliefs in X_3. Conversely, knowledge about X_3 will also influence our beliefs in X_1 through X_2. If however we have hard evidence about X_2 (we are 100% certain) then knowledge about X_1 will not change our beliefs of X_3 because X_2 is already supplying full support to X_3. The converse is also the case: knowledge about X_3 can't "pass" on to X_1 to cause a change in our beliefs of X_1 because X_2 is already "at full peak". We call this property *d-separation* (dependence separation) and say that X_1 and X_3 are *d-separated* by X_2 (double-circled in the figure).

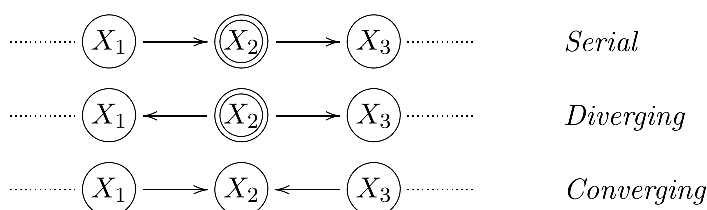


Figure 2.1: Possible connections

If we examine a *diverging* connection we have a similar situation. A change of belief in X_1 (X_3) causes a change in our beliefs in X_3 (X_1) through X_2. If we know X_2 we can however block the connection. Again we say that X_1 and X_3 are *d-separated* by X_2 (double-circled).

A *converging* connection has a more tricky interpretation. When we keep in mind that we are dealing with causal relations, we make the observation that knowing a cause of X_2 (and not knowing anything about X_2) — let's say X_1 — won't change our belief in any of the other causes of X_2 — here only X_3. The situation changes when we *do* have knowledge about X_2. If we also have knowledge about one of its causes — say X_1 — it does indeed change our belief in the other causes of X_2 — here X_3. Somehow we are maybe able to "explain away" that X_3 is a (major) cause of X_2 when we also have knowledge of one of its other causes. Further we make the observation that X_2 might have received evidence from elsewhere (if it has descendants). In that case we indirectly have gained knowledge about X_2, and consequently have a similar situation as we have just sketched. In contrast to the serial and diverging connections we block belief changes between X_1 and X_3 in the absence of knowledge about X_2 (or knowledge about one of its descendants), and we observe belief changes if we have knowledge about X_2 (maybe indirectly by its descendants).

We have analyzed the different DAGs from a intuitive point of view. In formal terms this just translate into the so-called *d-separation criterion* defined below:

Definition 3 (d-separation) *Let* $Gr = (\mathbf{X}, \bar{E})$ *be a DAG and let* $\mathbf{V}, \mathbf{Y}, \mathbf{Z} \subseteq \mathbf{X}$ *be three disjoint sets. The set* $\mathbf{Y}$ d-separates *the sets* $\mathbf{V}$ *and* $\mathbf{Z}$ *if for each* V_i *and each* Z_i *every connection* $V_i, \ldots, X_i, X_j, X_k, \ldots, Z_i$ *in Gr the following conditions hold:*

1. *The connection* X_i, X_j, X_k *is* serial *or* diverging *and* X_j *is in* $\mathbf{Y}$.
2. *The connection* X_i, X_j, X_k *is* converging *and neither* X_j *nor its descendants are in* $\mathbf{Y}$.

We can now relax our assumption that the edges represent causal relations. As long as we commit to the *d-separation criterion* everything will be all right.

The *d-separation* statements constrain how variables can undergo belief[1] changes when evidence is entered. The d-separation statements can be seen as expressing *graphical independencies* between variables in the DAG. In *statistical* terms X_i and X_k are *statistically independent* given X_j if $\Pr(X_i|X_j, X_k) = \Pr(X_i|X_j)$ which basically says "knowing X_j with certainty, and then getting to know X_k will not change our belief in X_i".

It makes sense to let *graphical independency* reflect (imply) *statistical independency* of the variables in our domain. For example we can express that "X_1 and X_3 are d-separated given X_2" (serial/diverging) in terms of statistical independence. This then translates into the statistical formulas[2] $\Pr(X_1|X_2, X_3) = \Pr(X_1|X_2)$ and as $\Pr(X_3|X_2, X_1) = \Pr(X_3|X_2)$.

[1] "Belief" is quantified in terms of (subjective) probability. We will omit the philosophical discussion as to the legitimacy of this.

[2] A whole axiomatic system exists that captures the concept of qualitative independence [8]. We will not go into detail about this.

2.2 Bayesian networks

A Bayesian network is a convenient and compact way of representing the relations between a number of variables. We have defined what we mean by DAG, d-separation and conditional independence, and we are now able to define the concept of a Bayesian network [2]:

Definition 4 (Bayesian network) *A* Bayesian network Bn *over a set of variables* $\mathbf{X}$ *is a pair* $Bn = (Bn_s, Bn_p)$ *with:*

1. $Bn_s = (\mathbf{X}, \bar{E})$ *is a DAG called the network structure of* Bn*.*
2. $Bn_p = \{\Pr(x_i|\mathbf{p}_{X_i})|X_i \in \mathbf{X}\}$ *is the set of local subjective probabilities.*

Bn_s is the DAG structure that expresses the graphical independencies of the variables (nodes) in our domain. This is the qualitative part of our domain. Bn_p is the set of subjective probabilities for each parent/child-configuration in the DAG, and is the quantitative measure of belief. Sometimes (context makes it clear when) we will treat the elements of the set Bn_p as variables. We also refer to these variables as *parameters* of the structure Bn_s.

From this definition and the definition of d-separation the following theorem can easily be derived by applying the 'chain rule' [8]:

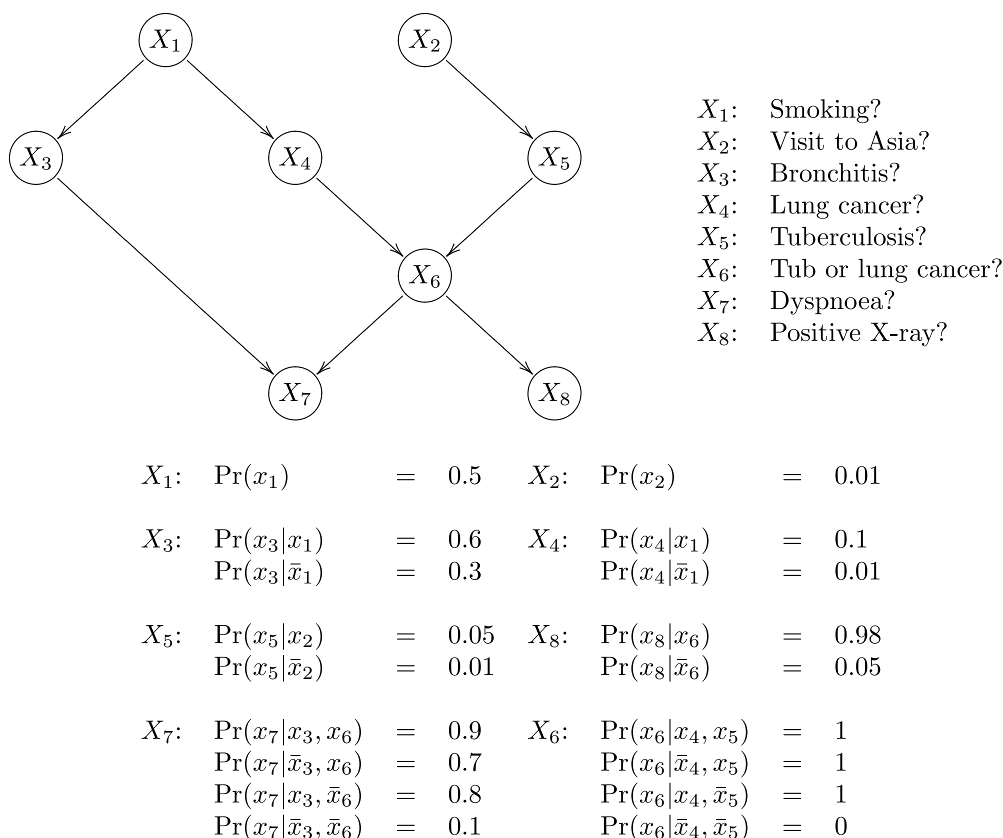
Theorem 1 (Joint distribution) *Let* $\mathbf{X}$ *be a set of domain variables. Let* $Bn = (Bn_s, Bn_p)$ *be a Bayesian network over* $\mathbf{X}$ *then we have:*

$$\Pr(\mathbf{X}|Bn) = \prod_{X_i \in \mathbf{X}} \Pr(X_i|\mathbf{P}_{X_i})$$

This definition gives the functional decomposition of a joint probability distribution. When we apply theorem 1, we are in fact just viewing Bayesian networks as a way of representing a joint probability distribution — nothing more (and nothing less). It is important to understand this fact, because we can then use axioms, laws and well-known facts from probability theory to reason (not only in the sense of inference) about Bayesian networks. Conceptually we may however think about Bayesian networks as representing the relations between domain concepts, causal influences or otherwise, as we may profit from this intuitive interpretation when building Bayesian networks in collaboration with domain experts [6].

Obviously it's now possible to calculate any probability by applying some basic algebra using laws of conditional probability. When we are dealing with many variables, complex structure and the cardinality of the variables is big, this method of calculating the conditional probabilities is infeasible (remember we are usually interested is several conditional probabilities, and not just a single calculation). Several (general) inference algorithm have been developed that optimize inference by exploiting conditional independence, most notably the (rather intuitive) method created by J. Pearl (1988) [8] and another (less intuitive approach) created by S.L. Lauritzen & D.J. Spiegelhalter (1988) [5]. Unfortunately the problem is NP-hard in general, but in many real-life applications the structure is often simple enough, and the optimized inference algorithms can be applied successfully (efficiently i.e. in polynomial time) [9].

The ASIA network depicted in figure 2.2 is an example of a (small) Bayesian network capturing domain knowledge about some possible diseases in connection with a visit to Asia. The variables X_i are binary variables with x_i denoting 'true' and $\bar{x}_i$ denoting 'false'. Also note that the local probabilities sum to unity (add up to 1).



X_1:	$\Pr(x_1)$	=	0.5	X_2:	$\Pr(x_2)$	=	0.01
X_3:	$\Pr(x_3 \mid x_1)$	=	0.6	X_4:	$\Pr(x_4 \mid x_1)$	=	0.1
	$\Pr(x_3 \mid \bar{x}_1)$	=	0.3		$\Pr(x_4 \mid \bar{x}_1)$	=	0.01
X_5:	$\Pr(x_5 \mid x_2)$	=	0.05	X_8:	$\Pr(x_8 \mid x_6)$	=	0.98
	$\Pr(x_5 \mid \bar{x}_2)$	=	0.01		$\Pr(x_8 \mid \bar{x}_6)$	=	0.05
X_7:	$\Pr(x_7 \mid x_3, x_6)$	=	0.9	X_6:	$\Pr(x_6 \mid x_4, x_5)$	=	1
	$\Pr(x_7 \mid \bar{x}_3, x_6)$	=	0.7		$\Pr(x_6 \mid \bar{x}_4, x_5)$	=	1
	$\Pr(x_7 \mid x_3, \bar{x}_6)$	=	0.8		$\Pr(x_6 \mid x_4, \bar{x}_5)$	=	1
	$\Pr(x_7 \mid \bar{x}_3, \bar{x}_6)$	=	0.1		$\Pr(x_6 \mid \bar{x}_4, \bar{x}_5)$	=	0

Figure 2.2: The Asia network [5]

2.3 Equivalence

Theorem 1 states that a Bayesian network defines a (or decomposes an existing) joint probability distribution. It says nothing about uniqueness. And indeed — different Bayesian networks can represent the same joint distribution. We define *equivalent* structures as follows:

Definition 5 (Equivalence) *Two DAGs Bn_s^* and Bn_s are* equivalent *if they express the same graphical independencies.*

From equivalent structures we can read the same d-separation statements. This means that the joint distributions of the corresponding Bayesian networks must have equivalent (in the logical sense) functional decompositions, because the decomposition reflects these d-separation statements. The following example illustrates these points.

In figure 2.3 the first three DAGs express the same d-separations statements. X_2 and X_3 are d-separated given X_1 and it is easy to see that the equivalence holds:[3]

$$\begin{aligned} \Pr(X_2)\Pr(X_1|X_2)\Pr(X_3|X_1) &\equiv \Pr(X_3)\Pr(X_1|X_3)\Pr(X_2|X_1) \\ &\equiv \Pr(X_1)\Pr(X_2|X_1)\Pr(X_3|X_1) \end{aligned}$$

We also say that the first three Bayesian networks (the DAG plus the corresponding local probabilities) are *statistically indistinguishable*. The fourth graph expresses different d-separation statements and we also see that:

$$\Pr(X_2)\Pr(X_1|X_2)\Pr(X_3|X_1) \not\equiv \Pr(X_2)\Pr(X_3)\Pr(X_1|X_2, X_3)$$

[3] Recall that conditional probability is defined as $\Pr(X|Y) = \frac{\Pr(X,Y)}{\Pr(Y)}$.

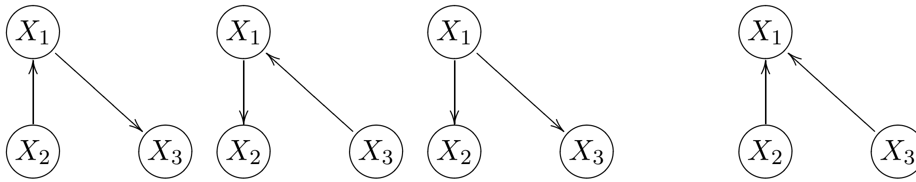


Figure 2.3: First three DAGs are equivalent

In general, if we have a Bayesian network $Bn = (Bn_s, Bn_p)$ then for any DAG, Bn^*_s expressing the same d-separation statements as Bn_s, we can *transform* the set of probabilities Bn_p into a set of probabilities Bn^*_p using a series of equations (as the above equivalence suggests) such that Bn and Bn^* represent the same joint probability distribution. For instance given the first network in the figure, we can calculate all the local probabilities of the second network and third network (after all the distribution $\Pr(X_1, X_2, X_3)$ is given by the first Bayesian network, and any local probability needed for the second and third network can be calculated using laws of conditional probability).

Also note that we can obtain equivalent DAG structures by a series of arc reversals, as long as we don't alter the d-separation statements. These statements are unaltered if the DAGs have the same set of edges (an arc in either direction implies an edge) and the same set of *v-structures* [3], that is a converging connection who's tails are not connected by an arc. In the figure we see that the any of the first 3 DAGs can be obtained by reversing the arcs, and we see that they all have the same set of v-structures, namely none. The fourth graph on the other hand also has the same set of edges, but additionally has a v-structure at variable X_1.

n	DAG Eq. classes	DAGs
1	1	1
2	2	3
3	11	25
4	185	543
.	...	...
8	212.133.402.500	783.702.329.343
9	326.266.056.291.213	1.213.442.454.842.881

Table 2.1: Number of DAGs vs. DAG Eq. classes

In table 2.1 some examples of the number of DAGs and DAG equivalence classes[4] are listed as the number of nodes n increase [12]. We see that the number of possible graphs drastically grows as n becomes bigger, both in the DAG case and (less, but still remarkable) in the case of DAG equivalence classes. If we create a Bayesian network by hand in collaboration with an expert we can use this to our advantage. The equivalent structures allows us to rearrange our DAG to an equivalent DAG if the expert has trouble with assessing some probabilities in the 'original' DAG (which for instance could have been induced from data). Maybe with some arrow reversals we can obtain an equivalent DAG with which the expert is more comfortable and therefore is able to asses the local probabilities more accurately. From an inductive point however we will later see that the number of possible DAG structures and statistical indistinguishability can be a problematic issue.

[4]Each class can be represented by a so called *essential graph*, a graph with directed *and* undirected edges [12]. Any DAG in a given class can be derived from the essential graph of that class.

Another point to consider in connection with equivalence is if we interpret the arcs in the DAG strictly causal or if we interpret them as representing some other relation where the direction is essential. The d-separation statements in general don't capture the direction of arcs. Sometimes however not only the probability distribution is a matter of interest, but the actual structure of the DAG itself is just as important (causality does not allow for an arrow reversal as is often possible in DAGs without destroying the graphical independencies). The right joint distribution with the wrong DAG is therefore considered wrong. This matter is currently an active area of research — we refer to [3]. We would like to point out though that generally it is impossible to induce the right causal DAG from databases alone.

2.4 General considerations

The statistical independencies induced by the d-separation statements of a Bayesian network makes the representation of the joint distribution compact, and we can exploit the independencies for (efficient) inference. Of course we can only exploit the independencies we read off the DAG *if* the variables involved are indeed statistically independent in the domain we are modelling. We therefore assume that *all* graphical independencies are *valid* statistical independencies.

In theory there are sets of independency statements (of a domain) [8] that can't be translated into d-separation statements such that all independency statements are implied in the same DAG. If we can't model all independencies of a domain simultaneously in a DAG, we at least want to represent as *many* independencies in the DAG as we can without somehow indirectly inducing a graphical independency that implies an invalid independency statement of the domain — we then have a so-called a *minimal I-map* (independence map). If the graphical independencies *all* imply statistical independencies we have an *I-map*. If the reverse is *also* the case — the statistical independencies all imply graphical independencies we have a *P-map* (perfect map).

All complete DAGs — that is a DAG with edges between all nodes — express no (graphical) independencies according to definition 3. Consequently any complete DAG of n nodes can represent any joint distribution of n variables, but in a quite non-compact way. The compactness was one of the advantages of using Bayesian networks in the first place.

Chapter 3

Induction

In this chapter we will review methods of how to induce Bayesian networks from data. When building Bayesian networks in collaboration with domain experts we usually distinguish the construction of the DAG on the one hand and assessing the conditional probabilities on the other. Those two processes are in great extent thought to be independent of each other. Usually the DAG is first constructed and in a later stage the probabilities are assessed of the already constructed DAG. When we want to induce Bayesian networks from data however, we cannot make such a strong distinction between the quantitative and the qualitative part.

In chapter 2 we argued that a Bayesian network can be regarded as a joint probability distribution. When constructing a Bayesian network in collaboration with domain experts we are actually trying to capture the domain knowledge in terms of a joint probability distribution. Of course the actual expert knowledge does not necessarily have the form of a joint probability distribution (most probably not). We are just trying to express her knowledge in *terms of a joint probability distribution* — in our case we do that using Bayesian networks because such networks makes the elicitation of the expert knowledge and the translation into a joint probability distribution easier (yet not unique). We ease the process of capturing her knowledge in probabilistic terms by conceptually distinguishing the DAG construction — making the causal (or otherwise) relations — and assessing the probabilities.

A database on the other hand contains a number of cases (records) each representing an outcome of a situation. A database is therefore nothing more than a collection of *outcomes* of the underlying joint probability distribution that created the cases. The (causal) relations that an expert can give are *not* explicitly given in the database, but have to be distilled from the database. The database however only contains the outcomes of some N cases of a probability distribution and we know that statistical conditional independencies can give rise to different equivalent DAGs. We therefore do not necessarily obtain the DAG that originally was the underlying structure when the database was created.

If we have managed to capture the DAG structure of the underlying probability distribution of the database we still need the local conditional probabilities for a Bayesian network. This can be done by simply counting from the database. We see that both structure as well as local probabilities are distilled from the same source namely the database (and we will see that they are actually interwoven with each other) whereas when eliciting from experts we are trying our best not to mix the two things up.

If we assume that we actually got hold of the right structure — by data or otherwise — and that we have some rough estimates of the local probabilities, we can use a database from the same joint probability distribution to *refine* these initial rough local probability assessments. This way of updating the local probabilities can be used in domains where the structure is

well known, but the local probabilities are uncertain. In this setting we can use the database to revise the local probabilities as the database of cases grows (incrementally or in batches). We refer to [5] for discussions about how this can be done by applying Bayes theorem, but we will not go into this here (although it actually follows from the 'Bayesian approach' we will discuss below).

3.1 Inductive methods

Before we discuss the different methods of induction, we will define the concept of a *database*:

Definition 6 (Database) *A* database $D = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N\}$ *is a number of identically distributed and independent samples from the joint probability distribution of* $\mathbf{X} = \{X_1, \ldots, X_n\}$*. An element of* D *is called a* case*.*

We assume that the cases are *independently sampled*. This means that the outcome of one case does not influence the outcome of another case. Additionally we assume that the cases are *identically distributed* which means that the cases are *time invariant*. Observing a certain case now is just as probable as seeing the case any time in the future or past.

Also note that we are only considering discrete variables. If we have a database with continuous variables one has to transform these into discrete variables [2], e.g. using intervals. In addition we demand that there are no missing values in the database. All variables of a case are known or observed. If there are cases with missing values one has to fill in the missing value or give a qualified guess as what the value is (would have been). Another option is to simply remove the cases with missing value from the database. This or course depends on the total number of cases in the database versus the number or cases with missing values. In general induction based on small-sized databases is less accurate than large-sized databases, and therefore we maybe can't afford to discard cases. Another point to consider is that a missing value might be *informative* in the sense that the value is missing under certain circumstances, for instance when some other variables in a case take on certain values. In a situation like this removal of the case is the same as removing an informative structure or relation of the underlying distribution. The missing value is then actually implying a value of that variable. Finally we want to note that methods do exist that can induce Bayesian networks with missing values (to a certain degree) based on Monte-Carlo simulation techniques like Gibbs sampling [9], but that these methods are (still) computationally expensive.

We know that a joint probability distribution can be represented in the form of a Bayesian network $Bn = (Bn_s, Bn_p)$. As stated in theorem 1 we can therefore decompose the joint distribution of $\mathbf{X}$ as $\Pr(\mathbf{X}|Bn) = \prod_{X_i} \Pr(X_i|\mathbf{P}_{X_i})$. A *case* in the database is thus a configuration (realization) of this joint distribution. The database D is a conjunction of cases. All cases are independent of each other such that we can write $\Pr(D) = \Pr(\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N) = \Pr(\mathbf{x}^1)\Pr(\mathbf{x}^2)\cdot\ldots\cdot\Pr(\mathbf{x}^N)$. Knowing that all cases are instantiations of the same joint distribution of $\mathbf{X}$ that we decompose according to Bn we can write $\Pr(D|Bn) = \Pr(\mathbf{x}^1|Bn)\Pr(\mathbf{x}^2|Bn)\cdot\ldots\cdot\Pr(\mathbf{x}^N|Bn)$. Here we have only considered cases that actually exist in D. We can however generalize to all cases, including hypothetical cases — cases that *could* have been an outcome of our joint distribution of $\mathbf{X}$ and therefore *could* have been in D. We can therefore write $\Pr(D|Bn) = \prod_{\mathbf{x}} \Pr(\mathbf{x}|Bn)^{N_{\mathbf{x}}}$, the product of the joint probabilities of all possible configurations of $\mathbf{X}$ where $N_{\mathbf{x}}$ is the number of instances of $\mathbf{x}$ in the database D with N cases (and $N_{\mathbf{x}} = 0$ if $\mathbf{x}$ is not in D). We see that this in fact is the likelihood of multinominal sampling from the joint distribution of $\mathbf{X}$.

There are three main approaches to learning a Bayesian network (excluding simulation techniques):

Independence tests are trying to test for statistical conditional independencies in the database. With a finite sized database a *significance threshold* is used. When the independence test is under the threshold the variables considered are taken as being statistically (conditionally) independent. The χ^2-test (chi squared) is an example of a test that can be used to find such independencies [12]. We will not discuss this method of induction in this sequel.

Frequentist approach assumes that the cases and frequencies in the database represent *the right* joint probability distribution, and that the database indeed reflects the right relations of our domain.

Bayesian approach assumes that the distributions in the database are uncertain, that is here we don't take the database as being *the* right distribution, but assume that the database just *revises* an *a priori* distribution over the parameters. *Bayes theorem* tells us how this prior distribution and the distribution reflected by the database are related.

When we induce Bayesian networks we use *scoring functions* for comparing two potential DAG structures to tell which is better or worse. We use a different scoring function depending on our inductive approach. We will have a look at both the Bayesian and frequentist way of scoring, starting with the latter.

3.1.1 Frequentist

The frequentist point of departure is the *maximum likelihood*. Here we try to find a Bayesian network that could have produced the cases in the database — that is, we try to maximize $\Pr(D|Bn)$, where $Bn = (Bn_s, Bn_p)$. This probability we derived earlier on, namely $\Pr(D|Bn) = \prod_{\mathbf{x}} \Pr(\mathbf{x}|Bn)^{N_{\mathbf{x}}}$. Now assume that the structure Bn_s of Bn is given. In that case we have to calculate for all possible configurations $\mathbf{x}$ of $\mathbf{X}$ the probability $\Pr(\mathbf{x}|Bn)^{N_{\mathbf{x}}}$. The probability $\Pr(\mathbf{x}|Bn)$ decomposes according to theorem 1 like $\Pr(\mathbf{x}|Bn) = \prod_{i=1}^{n} \Pr(x_i|\mathbf{p}_{X_i})$ where the parent configuration $\mathbf{p}_{X_i}$ is dictated by the structure Bn_s. The local probabilities can now be counted from the database with the formula $\Pr(x_i|\mathbf{p}_{X_i}) = \Pr(x_i, \mathbf{p}_{X_i}) / \Pr(\mathbf{p}_{X_i})$ and we get:

$$
\begin{aligned}
\Pr(D|Bn) &= \prod_{\mathbf{x}} \Pr(\mathbf{x}|Bn)^{N_{\mathbf{x}}} \\
&= \prod_{\mathbf{x}} \prod_{i=1}^{n} \Pr(x_i|\mathbf{p}_{X_i})^{N_{\mathbf{x}}} \\
&= \prod_{\mathbf{x}} \prod_{i=1}^{n} \left(\frac{\Pr(x_i, \mathbf{p}_{X_i})}{\sum_{x_i} \Pr(x_i, \mathbf{p}_{X_i})} \right)^{N_{\mathbf{x}}} \\
&= \prod_{\mathbf{x}} \prod_{i=1}^{n} \left(\frac{N_{x_i, \mathbf{p}_{X_i}}}{N_{\mathbf{p}_{X_i}}} \right)^{N_{\mathbf{x}}}
\end{aligned}
$$

To simplify this last product let's enumerate the different configurations of child node X_i and its parent set $\mathbf{P}_{X_i}$. Let Ω_{X_i} be ordered $x_{i,1}, \ldots, x_{i,r_i}$ and let $\Omega_{\mathbf{P}_{X_i}}$ be ordered $\mathbf{p}_{X_i,1}, \ldots, \mathbf{p}_{X_i,q_i}$ where r_i and q_i are the cardinality of Ω_{X_i} and $\Omega_{\mathbf{P}_{X_i}}$, respectively. We denote by N_{ijk} the number of occurrences in the database where variable i has the kth configuration

and the parent set of variable i has jth configuration, and $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$. We then have with $\frac{0}{0}^0 = 1$ [2]:

$$\prod_{\mathbf{x}} \prod_{i=1}^{n} \left(\frac{N_{x_i, \mathbf{p}_{X_i}}}{N_{\mathbf{p}_{X_i}}} \right)^{N_{\mathbf{x}}} = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \prod_{k=1}^{r_i} \left(\frac{N_{ijk}}{N_{ij}} \right)^{N_{ijk}}$$

Usually the logarithm is taken of this result, and we obtain the *maximum log-likelihood*:

$$\log \Pr(D|Bn) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \cdot \log \left(\frac{N_{ijk}}{N_{ij}} \right)$$

If the structure Bn_s of Bn is given and we extract the probabilities from the database to obtain Bn_p of Bn we can measure how likely Bn could be the underlying joint probability distribution of the database — in how far the cases could have been sampled from Bn. To maximize the log-likelihood we therefore have to *find* such a structure Bn_s so that $\Pr(D|Bn)$ is maximal. In chapter 2 we claimed that a complete DAG over n variables can represent *any* joint probability distribution of n variables. A 'solution' to the log-likelihood maximization of a database with n variables is therefore a complete DAG structure Bn_s. This is of course an inadequate 'solution' to the problem, as is any structure that expresses more dependencies than necessary.

In a structure with many arcs many probabilities have to be assessed. For a given variable X_i with cardinality r_i which has a parent set with cardinality q_i it is obvious that for each configuration j of parent set $\mathbf{p}_{X_i,j}$ an assessment has to be made for all $r_i - 1$ child configurations $x_{i,k}$ (the last r_i doesn't have to be assessed because we sum to unity). For node X_i a total number of $q_i \cdot (r_i - 1)$ probabilities have to be assessed. For all nodes it follows that $K = \sum_{i=1}^{n} q_i \cdot (r_i - 1)$, called the *degree of freedom* of Bn. An accurate estimation of these K probabilities is obtained only when the number of cases in D approaches infinity. Obviously when K is big more cases are needed to achieve an accurate estimate of all the probabilities, but since we have a finite number of cases in our database the accuracy of the K probabilities will be less accurate. We will therefore favor a sparse structure over a complex structure taking into account the number of cases N in the the database. Usually this is done by adding a penalty function to the maximum likelihood measure [5]. We use the degree of freedom of Bn as a measure of complexity and multiply this by a penalty function $p(N)$, where N is the number of cases in D.

We then have the *penalized maximum likelihood*:

Definition 7 (PML measure) *Let D be a database with N cases. Let $Bn = (Bn_s, Bn_p)$ be a Bayesian network. Let n, q_i, r_i and the counts N_{ijk} and N_{ij} be as before. Let $K = \sum_{i=1}^{n} q_i \cdot (r_i - 1)$ and $p(N)$ be a penalty function. The* penalty maximum likelihood *is then defined as*

$$\text{PML}(D, Bn) := \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \cdot \log \left(\frac{N_{ijk}}{N_{ij}} \right) - K \cdot p(N)$$

When $p(N) = \frac{1}{2} \cdot \log(N)$ then PML is equivalent to the *Minimum Description Length* [2] measure (MDL).[1]

When we want to learn a Bayesian network Bn from a database D from a frequentist point of view, we conclude that our aim is to maximize the PML measure:

$$\arg\max_{Bn_s} \text{PML}(D, (Bn_s, Bn_p))$$

[1] There are several schemes of the PML measure that have been derived from different schools of science — the MDL is from coding theory.

We want the structure Bn_s such that the PML is maximal. When we have found the Bn_s that maximizes PML we can add the corresponding Bn_p to obtain a full Bayesian network $Bn = (Bn_s, Bn_p)$.

3.1.2 Bayesian

The Bayesian way of looking at things is less 'ad hoc' than the frequentist approach. From a knowledge acquisition point of view the Bayesian way is theoretically more attractive. Bayesians assume that a database D just *revises* our *a priori* knowledge where frequentists believe that the database conveys the ultimate truth.

If we equal knowledge with a joint probability distribution (which of course is a rather controversial claim) Bayesians are uncertain about the local probabilities represented by the database. The local probabilities in Bn_p is a set, but here we write $\overrightarrow{Bn}_p$ to denote the corresponding 'local probability vector', such that each element of Bn_p is now uniquely indexed. We now define a *uncertainty probability distribution* over the vector $\overrightarrow{Bn}_p$. The point is that the database can revise our uncertainty belief about $\overrightarrow{Bn}_p$. Thus D can strengthen our prior uncertainty belief (consolidate) or weaken our prior uncertainty belief that $\overrightarrow{Bn}_p$ is *the* right 'set' of local probabilities to decorate our Bn_s. Bayes theorem tells us how our prior uncertainty is revised once we take the database into consideration. The revised uncertainty about $\overrightarrow{Bn}_p$ is called the *a posteriori* uncertainty. Thus with Bayes theorem we obtain:

$$\Pr(\overrightarrow{Bn}_p|D, Bn_s) = \frac{\Pr(\overrightarrow{Bn}_p|Bn_s) \cdot \Pr(D|(Bn_s, \overrightarrow{Bn}_p))}{\Pr(D|Bn_s)}$$

The probability $\Pr(D|(Bn_s, \overrightarrow{Bn}_p)) \equiv \Pr(D|Bn)$ was derived earlier on when discussing the database — it's the likelihood of multinominal sampling. The probability $\Pr(\overrightarrow{Bn}_p|Bn_s)$ is the (smooth) *prior uncertainty probability distribution*: given a structure Bn_s we define a distribution over the local probabilities Bn_p of that structure. The *posterior uncertainty probability distribution* $\Pr(\overrightarrow{Bn}_p|D, Bn_s)$ is the new uncertainty probability distribution when we have seen D. The probability $\Pr(D|Bn_s)$ is the *marginal likelihood* and equals the integral:

$$\Pr(D|Bn_s) = \int \Pr(D|(Bn_s, \overrightarrow{Bn}_p)) \cdot \Pr(\overrightarrow{Bn}_p|Bn_s)\ d\overrightarrow{Bn}_p$$

This integral gives us a measure of how well the structure Bn_s could have been responsible for creating D. Consequently this measure can be used to find the Bayesian network Bn [5]: We find a structure Bn_s and calculate the integral. To calculate the integral we see that we have to calculate the likelihood $\Pr(D|(Bn_s, \overrightarrow{Bn}_p))$ and combine this with the prior $\Pr(\overrightarrow{Bn}_p|Bn_s)$. Once we have found such a structure we can use the $\overrightarrow{Bn}_p$ that maximizes the marginal likelihood to add the local probabilities, and we then have a full $Bn = (Bn_s, Bn_p) \equiv (Bn_s, \overrightarrow{Bn}_p)$.

The marginal likelihood integral depends on our choice of the prior uncertainty distribution [3]. Normally a *conjugate prior* is chosen. This means that the posterior distribution is of the same family as the prior distribution. When dealing with databases as defined in definition 11 — a multinominal sample — the *Dirichlet distribution* is a conjugate prior distribution. This distribution allows us to calculate the above integral in closed form (*only* when no missing values in D). Hence let $\Pr(\overrightarrow{Bn}_p|Bn_s) := \mathrm{Dir}(\overrightarrow{Bn}_p|\vec{\alpha}_{\mathbf{x}})$, where $\vec{\alpha}_{\mathbf{x}}$ is the vector containing the *virtual counts*, that is the number of counts of each possible $\mathbf{x}$ (constrained by the Bn_s) before observing the actual counts in D. These counts thus give the number of times we believe to have "seen" $\mathbf{x}$ in the past, and upon these counts we base our prior uncertainty probability distribution. The counts are revised once the database is taken into account, which can be

seen [9] by calculating the posterior probability $\Pr(\overrightarrow{Bn_p}|D, Bn_s) = \mathrm{Dir}(\overrightarrow{Bn_p}|\vec{\alpha}_{\mathbf{x}} + \vec{N}_{\mathbf{x}})$ which again has the same form as the prior, but now with the added counts from the database.[2]

It is rather awkward to estimate counts of full cases $\mathbf{x}$ in $\vec{\alpha}_{\mathbf{x}}$. We can simplify this if we assume *parameter independence* [5]. This states that the local conditional probabilities are independent, $\Pr(Bn_p|Bn_s) = \prod_{i=1}^{n}\prod_{j=1}^{q_i}\Pr(\{\Pr(X_i|\mathbf{p}_{X_i,j})\}|Bn_s)$. We then assume that all $\Pr(\{\Pr(X_i|\mathbf{p}_{X_i,j})\}|Bn_s)$ are Dirichlet distributions:

$$\mathrm{Dir}(\{\Pr(X_i|\mathbf{p}_{X_i,j})\}|\alpha_{ij1},\ldots,\alpha_{ijr_i}) = \frac{\Gamma(\alpha_{ij})}{\prod_{k=1}^{r_i}\Gamma(\alpha_{ijk})}\cdot\prod_{k=1}^{r_i}\Pr(x_{i,k}|\mathbf{p}_{X_i,j})^{\alpha_{ijk}-1}$$

and the posterior is $\mathrm{Dir}(\{\Pr(X_i|\mathbf{p}_{X_i,j})\}|\alpha_{ij1}+N_{ij1},\ldots,\alpha_{ijr_i}+N_{ijr_i})$. We then get by integrating the marginal likelihood [4]:

$$\Pr(D|Bn_s) = \prod_{i=1}^{n}\prod_{j=1}^{q_i}\frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij}+N_{ij})}\prod_{k=1}^{r_i}\frac{\Gamma(\alpha_{ijk}+N_{ijk})}{\Gamma(\alpha_{ijk})}$$

with $\alpha_{ij} = \sum_{k=1}^{r_i}\alpha_{ijk}$, $\alpha_{ijk} > 0$ being the virtual counts where variable i has the kth configuration and the parent set of i has the jth configuration. The $\Gamma(\cdot)$ is the gamma density function which is $(n-1)!$ for positive integers. We will take the logarithm of this result, and we obtain the *prior Bayesian measure*:

Definition 8 (PB measure) *Let D be a database with N cases. Let $Bn = (Bn_s, Bn_p)$ be a Bayesian network. Let n, q_i, r_i, N_{ijk} and N_{ij} be as before, and let α_{ij} and α_{ijk} be the virtual counts. The* prior Bayesian measure *is then defined as:*

$$\mathrm{PB}(D, Bn) := \log\Pr(D|Bn_s) = \sum_{i=1}^{n}\sum_{j=1}^{q_i}\log\frac{(\alpha_{ij}-1)!}{(\alpha_{ij}+N_{ij}-1)!} + \sum_{k=1}^{r_i}\log\frac{(\alpha_{ijk}+N_{ijk}-1)!}{(\alpha_{ijk}-1)!}$$

When $\forall i,j,k\ \alpha_{ijk} = 1$ then the PB measure corresponds to the *Bayesian measure* discussed in [2] and [4]. Later we'll discuss the virtual counts and what influence they have etc.

When we want to learn a Bayesian network Bn from a database D from a Bayesian point of view, we conclude that our aim is to maximize the PB measure:

$$\arg\max_{Bn_s}\mathrm{PB}(D, (Bn_s, Bn_p))$$

We want the structure Bn_s such that the PB is maximal. When we have found the Bn_s that maximizes PB we add the corresponding Bn_p to obtain $Bn = (Bn_s, Bn_p)$.

3.1.3 Comparison

The ML measure relies heavily on the database to find the structure. This means that it will try to find a Bayesian network that fits the data as well as possible. The PML measure incorporates a penalty measure to counteract this problem of over-fitting. The question is however how we should interpret such a penalty measure from a probabilistic point of view. The ML measure has a clear probabilistic interpretation. When adding an extra term like the penalty function we solve the over-fitting problem in an ad hoc manner and we step away from the pure ML probabilistic point of view. Another problem is *what* function to choose as penalty. Several opinions exist on that matter. AIC, MDL, and BIC are all measures

[2]The Dirichlet distribution has (a) *sufficient statistics* which means that it is independent of D and the prior observed cases given the counts $\vec{\alpha}_{\mathbf{x}} + \vec{N}_{\mathbf{x}}$, i.e. the statistics (the counts that are a function of prior cases and D) summarizes everything we need to know.

[5] that belong to the ML family with some sort of penalty function added with a different interpretation of the penalty term. The most often used penalty function is $p(N) = \frac{1}{2} \cdot \log(N)$ simply because it often performs the best.

The PB measure is built around the idea that the database just revises knowledge. In this approach we don't start from scratch and try to learn a Bayesian network. The prior uncertainty probability distribution can potentially be a strong bias especially when we are dealing with small databases. The assessment of the virtual counts is however quite problematic. First of all we see that the prior is conditioned on a structure Bn_s. We thus assume that we have the right structure before we decide on the virtual counts. Of course if we don't have the right structure we probably can't estimate the prior counts either. Unfortunately we find the most optimal structure as a function of the prior and the likelihood. Consequently we have to define priors for *all* potential network structures. A workaround to this problem is to simply assign the same virtual counts to all networks as done in [4] — they assign a virtual count of 1 to each α_{ijk} — we refer to this as the *K2 metric*. We then end up with a so called *non-informative prior* because it doesn't capture real domain knowledge. Assigning such a non-informative prior to each network introduces a problem pertaining to equivalent networks as defined in chapter 2. Remember that when we are inducing from data, we are in fact just seeing some N outcomes (N drawings) of a joint probability distribution. We know from chapter 2 that several different Bayesian networks with equivalent structures could have defined this very probability distribution. When we are trying to find a DAG structure Bn_s of $Bn = (Bn_s, Bn_p)$ there are could be several Bn_s that are optimal from a statistical point of view, namely all equivalent DAGs. If we have an equivalence class we therefore want all DAGs in that class to have the same prior probability, i.e. the prior counts assigned to the DAGs in the class must be related such that the resulting prior uncertainty distribution of equivalent DAGs is the same. If we assign $\alpha_{ijk} = 1$ to all DAGs without taking into account the equivalent DAGs we are introducing a bias — some DAGs are chosen over other DAGs although they might be just as good an 'explanation' (because the counts of the DAGs are then unrelated within an equivalence class). In chapter 2 we mentioned that if Bayesian networks have equivalent DAGs it is possible to transform the set of local probabilities of one Bayesian network into set of local probabilities of another Bayesian network (and vice versa), such that all the Bayesian networks are statistically indistinguishable. This 'trick' is a way of avoiding the problems we just discussed. We identify one DAG from each possible equivalence class and assign the virtual counts to that DAG. We transform these counts into counts (through a series of equations) for all other DAGs in the class.

In [10] an approach is taken based on such a transformation. In collaboration with domain experts the prior virtual counts should be assessed for a suitable complete[3] DAG, i.e. in this setting we have to elicit virtual counts that are actually based on expert knowledge about past events.[4] Then under certain assumptions virtual counts for *any* DAG can be derived such that any two DAGs in the same equivalence class are *likelihood equivalent*. However, the number of virtual counts that need to be assessed is huge to say the least, and consequently far too difficult to elicit from domain experts (or literature for that matter). The method proposed is therefore mainly of theoretical interest. The specification of these *informative* priors proposed by [10] is referred to as the *BDe metric* (Bayesian Dirichlet equivalent). It is an informative metric because it captures domain knowledge believed to be valid.

Another non-informative metric is proposed by W. Buntine called the *BDeu metric* (uniform). This metric is a special case of the BDe metric [12]. No assessments have to be made,

[3]Note that all complete DAGs are equivalent.

[4]To be precise, an overall *equivalent sample size virtual count* is estimated and all other virtual counts are estimated as fractions of that count.

X_1	X_2	X_3
1	1	0
1	1	1
1	0	1
1	0	0
0	1	0
0	0	0
0	0	1
0	0	1

$$X_1 \longrightarrow X_2 \longrightarrow X_3 \quad \text{(a)}$$

$$X_1 \longleftarrow X_2 \longrightarrow X_3 \quad \text{(b)}$$

Figure 3.1: Sample database and DAGs

but all virtual counts are set to $\alpha_{ijk} = \frac{\alpha}{r_i \cdot q_i}$, where r_i and q_i are as before. The α is the *equivalent sample size* and is a measure of how big an influence the prior should have on the result. Experiments have shown [9] that the PB measure is very sensitive to the α. Small differences (± 1) can have drastic effects on the resulting network found. It is quite difficult to estimate the α such that we get an optimal network, and because BDeu is a non-informative metric no domain expert can help us decide on the right α. The BDe metric is also sensitive to the virtual counts, but here the domain expert should help (!) decide upon the right number.

To illustrate the difference between the K2 metric and the BDeu metric we will look at the example depicted in figure 3.1. For DAG (a) we calculate the PB quality (WLOG without taking the logarithm) with the K2 metric on basis of the cases (consisting of binary variables) given in the table left. Note that $q_i = 1$ if the parent set $\mathbf{P}_{X_i} = \emptyset$:

$$\overbrace{\frac{(2-1)!}{(2+8-1)!}4!4!}^{X_1} \cdot \overbrace{\frac{(2-1)!}{(2+4-1)!}3!1!\frac{(2-1)!}{(2+4-1)!}2!2!}^{X_2} \cdot \overbrace{\frac{(2-1)!}{(2+5-1)!}3!2!\frac{(2-1)!}{(2+3-1)!}1!2!}^{X_3} \quad = \quad \frac{24}{25}\frac{1}{9!6!}$$

We do the same for the equivalent DAG (b):

$$\overbrace{\frac{(2-1)!}{(2+5-1)!}3!2!\frac{(2-1)!}{(2+3-1)!}1!2!}^{X_1} \cdot \overbrace{\frac{(2-1)!}{(2+8-1)!}5!3!}^{X_2} \cdot \overbrace{\frac{(2-1)!}{(2+5-1)!}3!2!\frac{(2-1)!}{(2+3-1)!}1!2!}^{X_3} \quad = \quad \frac{1}{9!6!}$$

Here we observe that these two numbers differ although the DAGs are equivalent. To illustrate that the BDeu metric assigns the same quality to equivalent networks, we calculate the PB quality with an equivalent sample size of $\alpha = 4$. First we calculate for DAG (a):

$$\frac{(4-1)!}{(4+8-1)!}5!5! \cdot \frac{(2-1)!}{(2+4-1)!}3!1!\frac{(2-1)!}{(2+4-1)!}2!2! \cdot \frac{(2-1)!}{(2+5-1)!}3!2!\frac{(2-1)!}{(2+3-1)!}1!2! \quad = \quad \frac{48}{11!6!}$$

In the same way for DAG (b) we calculate:

$$\frac{(2-1)!}{(2+5-1)!}3!2!\frac{(2-1)!}{(2+3-1)!}1!2! \cdot \frac{(4-1)!}{(4+8-1)!}6!4! \cdot \frac{(2-1)!}{(2+5-1)!}3!2!\frac{(2-1)!}{(2+3-1)!}1!2! \quad = \quad \frac{48}{11!6!}$$

We see that both DAG (a) and (b) receives the same quality due to the BDeu metric.[5]

From a knowledge modelling point view the PB measure is interesting when used in conjunction with an informative prior such that two equivalent DAGs receive the same prior. We

[5] [2] showed that a version of the PML measure assigs the same quality to equivalent structures.

see that the BDe metric would be the best choice. Assessing the virtual counts of a complete network is unfortunately infeasible in a real-life setting. This leaves us with the K2-metric and the BDeu metric. The K2-metric is not likelihood equivalent. Furthermore it is not informative. This leaves us with the BDeu metric. This is a non-informative measure being likelihood equivalent. One may wonder what the whole idea then is of using the PB measure if we don't apply an informative prior. We might as well learn from data only anyway (so the PLM measure) if we don't have any qualified knowledge about what the uncertainty probability distribution looks like. In fact this is the critique put forward by the frequentists.

In [9] it is claimed (prove based on Laplace's approximation) that the PB measure and the PML with penalty function $p(N) = \frac{1}{2} \cdot \log(N)$ (the MDL measure) are asymptotically equivalent. Hence with very large databases we don't have to estimate priors. In reality of course we rarely have such large databases at our disposal, and the result is therefore only of theoretical interest.

3.2 Structure priors

The structure of a Bayesian network in both the PML and in the PB case is dependent on the counts in the database. It is therefore possible to bias the structure choice by influencing the counts. In the PB case this can be done by applying Bayes theorem on prior knowledge about the distribution of the counts in the database. In the PML case however we don't use Bayes theorem. Our only option is to include in the database any knowledge we might have about the counts (adding 'perfect' cases) before we apply the optimization of the PML measure. This is a more direct way of including prior knowledge compared to the PB case.

Another way of influencing the structure choice is to define so-called *structure priors*. We define a probability distribution over structures such that the most probable structure underlying the domain we are modelling (and which the database is a based upon) receives the highest probability. Thus for Bayesian networks $Bn = (Bn_s, Bn_p)$ we define $\Pr(\cdot)$ on the structure space $\mathcal{S}$. Note that we are only defining this distribution over graphs that are potential structures of a Bayesian network i.e. DAGs. We will look at how to implement this in the PML case and in the PB case.

3.2.1 The PML case

The PML measure includes the ML term and the penalty term. We can think of the probability distribution over structures as another form of penalty where the most unlikely structures punish the PML the most. We can therefore add a 3rd term to the PML measure, and we get (when we take the logarithm) [2]:

$$\mathrm{PML}_s(D, Bn) := \log \Pr(Bn_s) + \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \cdot \log\left(\frac{N_{ijk}}{N_{ij}}\right) - K \cdot p(N)$$

3.2.2 The PB case

The natural way of specifying model priors in the Bayesian setting is to use Bayes theorem on the marginal likelihood. We then get

$$\Pr(Bn_s|D) = \frac{\Pr(Bn_s) \cdot \Pr(D|Bn_s)}{\sum_{s \in \mathcal{S}} \Pr(s) \cdot \Pr(D|s)} = c \cdot \Pr(Bn_s, D)$$

The denominator $\Pr(D)$ is obviously constant, and we can concentrate on maximizing the term $\Pr(D, Bn_s)$ when dealing with the same database D and we get:

$$\text{PB}_s(D, Bn) := \log \Pr(Bn_s) + \sum_{i=1}^{n} \sum_{j=1}^{q_i} \log \frac{(\alpha_{ij} - 1)!}{(\alpha_{ij} + N_{ij} - 1)!} + \sum_{k=1}^{r_i} \log \frac{(\alpha_{ijk} + N_{ijk} - 1)!}{(\alpha_{ijk} - 1)!}$$

From a theoretical point of view we again see that the Bayesian approach is less ad hoc compared to the PML approach. When $N \to \infty$ however we see that the implementation of the structure prior in the PML makes sense (due to MDL $\approx$ PM when $N \to \infty$).

3.3 Search methods

The PB_s and PML_s are *scoring functions*. When we traverse the solution space we use such a function to tell us how fit a potential solution is. In chapter 2 we saw that the number of DAGs is huge ((super)exponential in the number of variables). Even for few variables it's computationally impossible to evaluate all structures. In [2] it is shown that finding a structure with the highest quality is NP-hard, so we will not find the *best* network by systematically ('brute force') traversing the solution space. We can deal with this situation in two ways:

1. We can search for **one best** Bayesian network, knowing that we probably will not find the best network, but still accept the returned network as being the best hypothesis. We ignore alternatives. This approach is referred to as *model selection* [9].

2. We can search for a small **set of good** networks, i.e. Bayesian networks that all score quite good, and we can somehow weigh these networks to obtain an average.[6] This approach is called *model averaging* [3].

When we here say *good* or the *best* network we usually think about how well the Bayesian network can account for the database in a statistical way. From that point of view we are not really concerned with how the DAG structure looks of the Bayesian network (except that it of course has to describe the database). If we are trying to gain insight or would like to confirm the relations between the variables of the database we are not satisfied with just *any* structure although it might describe the counts in the database. If we are interpreting the arcs as causal relations, it is extremely critical that the arcs are directed in the right way. If we are just trying to see which variables are related, the direction is maybe less important — we are just interested to see if there is an edge or not. If we want to use the the network found for prediction, that is to apply the Bayesian network in a knowledge based setting for decision support, the decision maker or domain expert will probably want to confirm or validate the induced network and see if it corresponds to her prior experience. This of course can only be done if the structure somehow reflects the right relations (directed or not) and not just any relations.

3.3.1 Search strategies

We require some sort of search strategy to find the best or a set of good networks. One way to find such networks is to apply a *search heuristic* and hope that we come across a satisfying network. The K2 heuristic [4], and the B heuristic (due to W. Buntine) [1] are two algorithms developed specifically for inducing structures of Bayesian networks. Both algorithms are *greedy search algorithms* that, that is they incrementally add arcs to the current DAG structure

[6]Somehow average the statistical predictive performance (inference results) of the set of Bayesian networks.

according to the gained increase in quality (using a scoring function). The K2 additionally assumes an ordering of the variables (nodes) which of course decreases the number of possible structures. This ordering should be given by a domain expert — the question is however if this is possible.

The removal of an arc is not possible in the standard implementation of these two algorithms, nor is the the reversal of an arc. Both heuristics search in the DAGs solution space and not in the DAG equivalence classes solution space, and might 'try' several different DAGs that are equivalent (which from a computational point of view is a waste of time). Finally we would like to remark that the aforementioned K2 and B search heuristics in the standard implementation don't find a set of good networks, but only return a single network (model selection).

Alternatively one can use 'general' algorithms that explore the solution space or optimize the scoring function, like tabu search, and simulated annealing. In chapter 5 we will consider evolutionary algorithms as a way of optimizing the scoring function.

As discussed in chapter 2, we are interested in finding (a) minimal I-map(s) of D. With a finite size database (N finite cases) however [2] showed that no matter which search strategy we employ we will not necessarily obtain a minimal I-map when using the Bayesian or frequentist quality measure. With an infinite database the quality measures will prefer a minimal I-map, but this of course depends on the search strategy we use (if we systematically run through all possible DAGs, we then *will* find a minimal I-map).

3.3.2 Score decomposition

When comparing structures we have to calculate PB or PML (we for the moment exclude the structure prior terms). When these formulas can be written as a product (or sum when we take the logarithm) of measures each of which is a function of one node and its parents [10], we say that the measure is *decomposable*. We see that the PML measure decomposes:

$$\begin{aligned}
\mathrm{PML}(D, Bn) &= \sum_{i=1}^{n}\left[\sum_{j=1}^{q_i}\sum_{k=1}^{r_i} N_{ijk}\cdot\log\left(\frac{N_{ijk}}{N_{ij}}\right)\right] - \sum_{i=1}^{n}\left[q_i\cdot(r_i-1)\cdot p(N)\right] \\
&= \sum_{i=1}^{n}\left[\sum_{j=1}^{q_i}\sum_{k=1}^{r_i} N_{ijk}\cdot\log\left(\frac{N_{ijk}}{N_{ij}}\right) - q_i\cdot(r_i-1)\cdot p(N)\right] \\
&= \sum_{i=1}^{n}\mathrm{pml}^{D}_{Bn_s}(X_i, \mathbf{P}_{X_i})
\end{aligned}$$

And we see that the PB measure decomposes:

$$\begin{aligned}
\mathrm{PB}(D, Bn) &= \sum_{i=1}^{n}\left[\sum_{j=1}^{q_i}\log\frac{(\alpha_{ij}-1)!}{(\alpha_{ij}+N_{ij}-1)!} + \sum_{k=1}^{r_i}\log\frac{(\alpha_{ijk}+N_{ijk}-1)!}{(\alpha_{ijk}-1)!}\right] \\
&= \sum_{i=1}^{n}\mathrm{pb}^{D}_{Bn_s}(X_i, \mathbf{P}_{X_i})
\end{aligned}$$

We can thus calculate the quality per node of a network, and sum the quality to obtain the quality of the entire network. When we compare networks, we then just have to compare those nodes that have different parent sets. Computationally this partial calculation saves a lot of time.

Chapter 4

Priors

In this chapter we will consider structure priors in more detail. In this sequel we assume no prior information about the counts in the database that can bias structure selection (as briefly discussed in the preceding chapter).

Without any structural priors, the database obviously has to reflect the 'truth', that is, it should not be too strongly biased. We thus assume that the database is sampled from $\Pr(\mathbf{X}|Bn)$, where Bn is some Bayesian network. In reality this is of course not the case. If we consider the database as a collection of observations (patient records) we mostly don't have a representative sample of the (supposedly) underlying joint distribution. If we induce the Bayesian network on this biased database we will not get the right Bayesian network. A related issue is the sample size — the number of cases in the database. We earlier mentioned the *degree of freedom* K, the number of local probabilities that have to be assessed of the network. The bigger K the more cases are needed in D to obtain reliable probabilities which we need to find the structure. Another issue to take into consideration is the *noise* that any real-life database contains — cases that are exceptions to the general rule somehow or are very rare and not considered to be 'stereotypical' for the domain to be modelled. If we model too much noise we might end up with a Bayesian network that is too specific, and we can't use the network for general predictive decision support. On the other hand we can't afford to be too general because we might not capture the relations that *are* important for prediction.

We earlier mentioned that we can't systematically explore all structures to find the one (or a set) with the highest (good) quality. With search heuristics we should not expect to find *the real* structure underlying the database. If we can somehow decrease the number of possible structures that have to be considered — narrow our solution space — our chance of finding the right structure of course increases. Also for more general search algorithms narrowing the search space will increase our chance of finding a suitable structure.

One way of influencing the induction of Bayesian networks when dealing with databases exhibiting the above mentioned 'problems' is to use structure priors. With such a prior we can reinforce or punish structures that obey certain structural criteria. These criteria have to be elicited from the domain expert in advance before applying the induction. In sum, the role of structure priors is:

1. To influence a biased, noisy or too specific database.
2. To complement a database with a limited number of cases.
3. To 'guide' the search (heuristic) to find (an) optimal/good solution(s).

4.1 Domain Experts

The initial reason for automatically building Bayesian networks from data lies in the cumbersome, tiresome, time-consuming and often impossible knowledge-acquisition task we have to go through to create a hand-made network. The idea of automatically building networks from data is a nice idea, but due to the problems mentioned above is not feasible. When building knowledge-based systems we are modelling knowledge on a *symbolic level* and not on a *sub-symbolic level* (a black box approach). The domain expert should be able to recognize the structure of the induced Bayesian network for verification and validation purposes before the approval is given to employ the Bayesian network in a decision support setting. Sometimes obvious but important relations valid in a knowledge domain are not identifiable in the automatically constructed network because the relations might not be strongly supported by the database due to the problems discussed in the previous section. Such missing relations might be crucial to the expert or decision maker (but not necessarily for a correct inference result), and can very well make the difference between approval and disapproval of the whole decision-support system [6]. Experts often interpret the arcs of a Bayesian structure as representing causal relations between variables. From this we profit when we build a hand-made network, but when inducing the structures from data we don't rely on the causal (intuitively appealing) interpretation of the arcs. We may induce a structure where the arcs are reversed yielding a quite counterintuitive structure when interpreted as a causal network. Experts are reluctant to accept such a network because after all it 'looks' different than they expect (we can however sometimes find an equivalent structure that might look more familiar to the expert).

Mostly the structural features that are considered absolutely essential or obvious should be quite easy to elicit: Experts can often say which relations are absent for sure or which relations are absolutely certain to be part of the structure, for instance which direction an arc will have etc. If we model these essential or obvious features as a part of the structural prior, such that any induced structure not obeying these features is penalized, we are able to strongly influence how the resulting network will look.

4.2 Knowledge Structures

The structure of a Bayesian network is a mesh of relations. Still, often sub-structures can be identified that conceptually seem to belong together or are more tightly coupled than other relations. Usually the single arcs are considered the smallest piece of 'information' when building Bayesian networks. From a cognitive perspective it is however doubtful if expert knowledge is best expressed (or elicited) in terms of single arcs between concepts (variables, nodes). Here we will briefly discuss a possible cognitive model that handles *knowledge constructs* in contrast to the single relational approach.

Within well known cognitive ACT (Adaptive Control of Thought) framework of Anderson et al. [1] declarative knowledge — knowledge one can verbalize — is located in *chunks*. A chunk combines a limited number of tightly related declarative knowledge atoms into an compact integral declarative information unit. These chunks store the concepts and relations between the concepts that are relevant to the currently active knowledge construct. 'Currently active' refers to the present cognitive state (of the expert), e.g. when discussing a certain issue or thinking about a certain process.

The chunks are said to have *configural properties*. This means that the ordering of the knowledge atoms within that unit is important — we are not dealing with simple *sets*, but with structured pieces of related knowledge. Chunks also have *hierarchical relationships* with

each other. Two chunks can be combined to form a *superchunk*. These superchunks can again be combined to form supersuperchunks etc. From a cognitive point of view, hierarchical relating chunks makes it possible for the expert to keep relevant knowledge quickly accessible, e.g. while actively solving a problem. The expert can however only focus on one single chunk at a time if precision is important.

If we can elicit entire chunks from experts, we can use these chunks as the basic *building blocks* of Bayesian networks rather than single relations. With chunks we namely have a number of cognitively closely related concepts that naturally belong together (associatively relevant to each other), and therefore should be 'easy' to elicit.

4.3 Chunks

In formal (structural) terms we will define a *structural chunk* as a sub-DAG that is embedded in the structure of the Bayesian network (the super-DAG) we are trying to induce. A structural chunk can take on one of several *chunk graph configurations* (corresponding to the configural properties). The *combination* (corresponding to the hierarchical relationships) of all structural chunks that have all been assigned a chunk graph configuration is then Bn_s of the Bayesian network $Bn = (Bn_s, Bn_p)$ we are looking for. We will first give a few definitions:

Definition 9 (Edge variable) *Let $\mathbf{X}$ be a finite set of nodes ordered according to $X_1, \ldots, X_n$. An* edge variable $\epsilon^i_j, j > i$ *between node X_i and node X_j is a stochastic variable with state space* $\Omega_{\epsilon^i_j} = \{\mathsf{down}, \mathsf{up}, \mathsf{none}\}$*. The* edge variable set *is the set* $\mathcal{E}_\mathbf{X} = \{\epsilon^1_2, \ldots, \epsilon^1_n, \epsilon^2_3, \ldots, \epsilon^2_n, \ldots, \epsilon^{n-1}_n\}$ *of all edge variables between all nodes in $\mathbf{X}$.*

An edge variable corresponds to an edge between two nodes in $\mathbf{X}$. The cardinality of the edge variable set is therefore $|\mathcal{E}_\mathbf{X}| = \frac{n \cdot (n-1)}{2}$, which is the number of edges between n nodes. We will now group edge variables into disjoint sets called *chunks*:

Definition 10 (Chunk) *Let $\bigcup_{k=1 \ldots m} \mathcal{C}^k_{\mathbf{X}_k} \subseteq \mathcal{E}_\mathbf{X}$ such that $\mathcal{C}^i_{\mathbf{X}_i} \cap \mathcal{C}^j_{\mathbf{X}_j} = \emptyset$ for all $i \neq j$, where $\mathbf{X}_k$ is the set of nodes between which the edge variables are defined in $\mathcal{C}^k_{\mathbf{X}_k}$ and $\mathbf{X} = \bigcup_{k=1 \ldots m} \mathbf{X}_k$. The set $\mathcal{C}^k_{\mathbf{X}_k}$ is then called a* (structural) chunk.

We will omit the subscript of $\mathcal{C}^k_{\mathbf{X}_k}$ when the context makes it clear what we mean. Note that when we decide how to *chunk* the edge variables, we only demand that we have to select a subset of the edge variables, $\bigcup_{k=1 \ldots m} \mathcal{C}^k \subseteq \mathcal{E}$ (thus we are not actually partitioning $\mathcal{E}$), but that all the chunks together have to concern all nodes, $\mathbf{X} = \bigcup_{k=1 \ldots m} \mathbf{X}_k$.

By $\bar{\mathcal{C}}^k$ we denote a configuration of all edge variables in $\mathcal{C}^k$. We may think of a configuration of an edge variable between node X_i and X_j as representing the arc (direction of the 'edge') between those two nodes. Formally we will define this idea in terms of *chunk graphs*.

Definition 11 (Chunk graph) *The pair $CGr = (\mathbf{X}_k, \bar{\mathcal{C}}^k)$ is called a* chunk graph (configuration) *if the corresponding directed graph $Gr = (\mathbf{X}_k, \bar{E})$ is acyclic, where the chunk configuration $\bar{\mathcal{C}}^k$ and the edge set $\bar{E}$ are related in the following way, $\forall \epsilon^i_j \in \mathcal{C}^k$:*

- $\epsilon^i_j = \mathsf{down} \Leftrightarrow (X_i, X_j) \in \bar{E}$
- $\epsilon^i_j = \mathsf{up} \Leftrightarrow (X_j, X_i) \in \bar{E}$
- $\epsilon^i_j = \mathsf{none} \Leftrightarrow (X_i, X_j), (X_j, X_i) \notin \bar{E}$

Definition 11 links chunk configurations with (sub)-DAGs. If a configuration $\bar{\mathcal{C}}^k$ of chunk k is transformed into the edge set $\bar{E}$ according to the definition and the result is a DAG, we call $CGr = (\mathbf{X}_k, \bar{\mathcal{C}}^k)$ a chunk graph. For instance if we have $\bar{\mathcal{C}}^k = \{\epsilon_2^1 = \mathsf{up}, \epsilon_3^1 = \mathsf{down}, \epsilon_3^2 = \mathsf{none}\}$ then $\bar{E} = \{(2,1),(1,3)\}$ which is acyclic, and $CGr = (\{X_1, X_2, X_3\}, \bar{\mathcal{C}}^k)$ is a chunk graph. If $\bar{\mathcal{C}}^k = \{\epsilon_2^1 = \mathsf{down}, \epsilon_3^1 = \mathsf{up}, \epsilon_3^2 = \mathsf{down}\}$ then $\bar{E} = \{(1,2),(3,1),(2,3)\}$ which is cyclic and therefore not a chunk graph.

4.3.1 Combination

It is not guaranteed that the combination[1] of two chunk graphs is also a chunk graph; equivalently: we don't necessarily obtain a DAG by combining two DAGs. We will analyze the criteria under which we can and can't combine two chunk graphs such that the resulting graph is also a chunk graph.

Let $CGr' = (\mathbf{X}_1, \bar{\mathcal{C}}^1)$ and $CGr'' = (\mathbf{X}_2, \bar{\mathcal{C}}^2)$ be two chunk graphs, and $Gr' = (\mathbf{X}_1, \bar{E}^1)$ and $Gr'' = (\mathbf{X}_2, \bar{E}^2)$ the corresponding DAGs. The question is if $CGr''' = (\mathbf{X}_1 \cup \mathbf{X}_2, \bar{\mathcal{C}}^1 \cup \bar{\mathcal{C}}^2)$ is also a chunk graph thus if $Gr''' = (\mathbf{X}_1 \cup \mathbf{X}_2, \bar{E}^1 \cup \bar{E}^2)$ is a DAG.

We have to realize that to create a cycle between Gr' and Gr'' there has to be a directed path from Gr' to Gr'' and another directed path from Gr'' back to Gr'. Consequently the two DAGs have to at least share two common nodes. If $|\mathbf{X}_1 \cap \mathbf{X}_2| < 2$ we can't introduce any cycles.

Let now $\mathrm{In}_{Gr}(X_k)$ and $\mathrm{Out}_{Gr}(X_k)$ be the *in* and *out* degree of node X_k of a graph Gr, which is the number of incoming resp. outgoing arcs of a node X_k.

We then look at the case where $|\mathbf{X}_1 \cap \mathbf{X}_2| \geq 2$. For any two common nodes $X_i, X_j \in (\mathbf{X}_1 \cap \mathbf{X}_2)$, $X_i \neq X_j$ if $\mathrm{In}_{Gr'}(X_i) \neq 0$ and $\mathrm{Out}_{Gr''}(X_i) \neq 0$ (there is a path from Gr' through X_i to Gr'') and $\mathrm{In}_{Gr''}(X_j) \neq 0$ and $\mathrm{Out}_{Gr'}(X_j) \neq 0$ (there is another path from Gr'' through X_j to Gr') then there *might* be a cycle $X_i, \ldots, X_j, \ldots, X_i$ in Gr''' where the first three dots indicate a path coming from Gr'' and the last three dots indicate a path coming from Gr' as illustrated in figure 4.1.

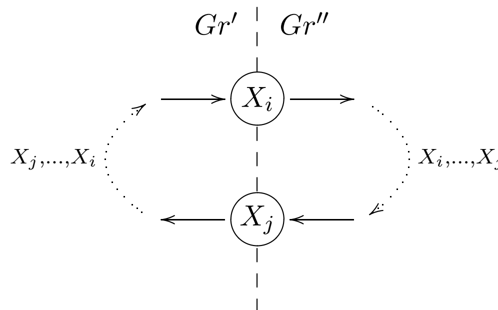


Figure 4.1: The combined Gr''' of Gr' and Gr''

When we combine chunk graphs we have to check for common nodes of the kind illustrated in the figure above (also in the symmetrical case) to see if the resulting combined graph stays acyclic. If the graph is acyclic, we call CGr''' the combined chunk graph of CGr' and CGr''. Note that any substructure of a DAG is again a (sub)-DAG. We can therefore 'split' combined chunk graphs, and we end up with several (sub) chunk graphs.

[1] By 'combination' we mean the union of nodes and the union of edge variables.

4.3.2 Chunks as priors

We want to use chunk graphs to model prior structure information, and we also want to build Bayesian networks by combining chunk graphs. First let's deal with chunk graphs as structure priors.

We define a probability distribution $\Pr(\cdot)$ on the chunk graph space $\mathcal{S}_{\mathcal{C}^i}$ such that $\Pr(\cdot)$ obeys the basic axioms of probability theory. For example we might have a chunk $\mathcal{C} = \{\epsilon_2^1, \epsilon_3^1\}$ on $\mathbf{X} = \{X_1, X_2, X_3\}$. The probabilities of all possible (they are all chunk graphs ⇒ DAGs) configurations of that set are:[2]

$$\begin{array}{lcl}
\Pr(\{\epsilon_2^1 = \mathsf{up}, \epsilon_3^1 = \mathsf{down}\}) & = & 1/3 \\
\Pr(\{\epsilon_2^1 = \mathsf{up}, \epsilon_3^1 = \mathsf{up}\}) & = & 1/3 \\
\Pr(\{\epsilon_2^1 = \mathsf{up}, \epsilon_3^1 = \mathsf{none}\}) & = & 0 \\
\Pr(\{\epsilon_2^1 = \mathsf{down}, \epsilon_3^1 = \mathsf{down}\}) & = & 1/3 \\
\Pr(\{\epsilon_2^1 = \mathsf{down}, \epsilon_3^1 = \mathsf{up}\}) & = & 0 \\
\Pr(\{\epsilon_2^1 = \mathsf{down}, \epsilon_3^1 = \mathsf{none}\}) & = & 0 \\
\Pr(\{\epsilon_2^1 = \mathsf{none}, \epsilon_3^1 = \mathsf{down}\}) & = & 0 \\
\Pr(\{\epsilon_2^1 = \mathsf{none}, \epsilon_3^1 = \mathsf{up}\}) & = & 0 \\
\Pr(\{\epsilon_2^1 = \mathsf{none}, \epsilon_3^1 = \mathsf{none}\}) & = & 0
\end{array}$$

We see that the corresponding DAGs $X_2 \rightarrow X_1 \rightarrow X_3$ and $X_2 \rightarrow X_1 \leftarrow X_3$ and $X_2 \leftarrow X_1 \rightarrow X_3$ are all even probable configurations of chunk $\mathcal{C}$ and that any other configuration can be excluded.

Note that chunks can be singleton sets. In that case we are treating the edge variable (corresponding to the edge) as a chunk, and we can still talk about probabilities of the configurations of that edge (variable taking on a certain direction).

The probability of a chunk graph is a measure of our belief that this chunk graph is part of the real structure. We can think of the probabilities of chunk graphs in two ways. If we interpret the extremes 0 and 1 as representing 'absolutely impossible' and 'absolutely certain', anything in between can be interpreted as 'possible'. The higher the number the stronger is our belief that we are dealing with the right chunk graph configuration. On the other hand we might not be able to nuance our estimate as to which chunk graph configuration is more probable than another between the extremes 0 and 1. Often we can however say what is *not possible* — which configuration is surely not the right chunk graph configuration. All such configurations we can assign a probability of 0. Keeping in mind that we have to sum to unity, all other chunk graphs of the chunk, we assign an uniform probability (as done above — all three chunk graph configurations receive the same probability).

A chunk graph can now be interpreted as a way of specifying prior structure information. Recall that a chunk is a unit of 'information' that is somehow tightly coupled. This coupling can be artificial or can be naturally occurring within the domain we are modelling. With 'artificial' we mean that we can define the chunks in a way that we think is appropriate for specifying priors, whereas 'naturally occurring' means that the chunks are given beforehand and actually define pieces of knowledge that can't be split.

We will discuss a few possible chunk graphs. Figure 4.2 depicts three different chunks $\mathcal{E} \supseteq \mathcal{C}^1 \cup \mathcal{C}^2 \cup \mathcal{C}^3 = \{\epsilon_p^j\} \cup \{\epsilon_k^j\} \cup \{\epsilon_l^k, \epsilon_p^k, \epsilon_p^l\}$ with $\mathbf{X} = \mathbf{X}_1 \cup \mathbf{X}_2 \cup \mathbf{X}_3 = \{X_j, X_p\} \cup \{X_j, X_k\} \cup \{X_k, X_l, X_p\}$. We have for each chunk defined the following probabilities for all chunk graphs of these chunks (3^1, 3^1 and $3^3 - 2 = 25$ DAGs):

[2] We actually defined $\Pr(\cdot)$ on chunk graphs which is a pair, but here we leave the node set $\mathbf{X}$ implicit.

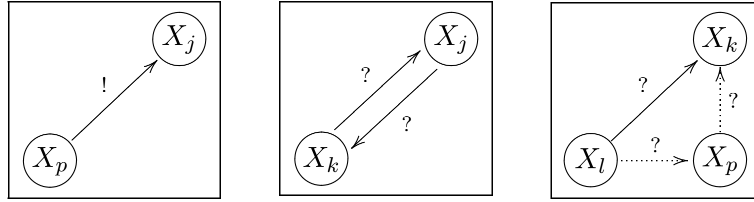


Figure 4.2: Left to right: $\mathcal{C}^1$, $\mathcal{C}^2$ and $\mathcal{C}^3$

$$\mathcal{C}^1: \begin{array}{lcl} \Pr(\{\epsilon_p^j = \mathsf{down}\}) & = & 0 \\ \Pr(\{\epsilon_p^j = \mathsf{up}\}) & = & 1 \\ \Pr(\{\epsilon_p^j = \mathsf{none}\}) & = & 0 \end{array} \quad \mathcal{C}^2: \begin{array}{lcl} \Pr(\{\epsilon_k^j = \mathsf{down}\}) & = & 0.5 \\ \Pr(\{\epsilon_k^j = \mathsf{up}\}) & = & 0.5 \\ \Pr(\{\epsilon_k^j = \mathsf{none}\}) & = & 0 \end{array}$$

$$\mathcal{C}^3: \begin{array}{lcl} \Pr(\{\epsilon_l^k = \mathsf{down}, \epsilon_p^k = \mathsf{down}, \epsilon_p^l = \mathsf{down}\}) & = & 0 \\ \ldots & = & 0 \\ \Pr(\{\epsilon_l^k = \mathsf{up}, \epsilon_p^k = \mathsf{none}, \epsilon_p^l = \mathsf{none}\}) & = & 0.5 \\ \ldots & = & 0 \\ \Pr(\{\epsilon_l^k = \mathsf{none}, \epsilon_p^k = \mathsf{up}, \epsilon_p^l = \mathsf{down}\}) & = & 0.5 \\ \ldots & = & 0 \\ \Pr(\{\epsilon_l^k = \mathsf{none}, \epsilon_p^k = \mathsf{none}, \epsilon_p^l = \mathsf{none}\}) & = & 0 \end{array}$$

With chunk $\mathcal{C}^1$ we are absolutely sure that there is an arc from X_p to X_j, and all other configurations we consider impossible. When two variables are strongly correlated, we might not be sure about the direction of an arc. With chunk $\mathcal{C}^2$ we can represent such a situation. Here we are unsure if there is an arc from X_j to X_k or vice versa. With chunk $\mathcal{C}^3$ we can represent a situation where we are unsure how variable X_k is influenced. X_k receives an arc directly from X_l or (exclusively) indirectly through X_p.

Notice that we have assigned a uniform probability to all chunk graph configurations we consider possible, which we can interpret as modelling an 'or' operator. Theoretically we could assign any value between 0 and 1 and thereby model a 'biased or', that is an 'or' operator that "prefers" one chunk graph configuration over another. For $\mathcal{C}^2$ we could have $\Pr(\{\epsilon_k^j = \mathsf{down}\}) = 0.75$ and $\Pr(\{\epsilon_k^j = \mathsf{up}\}) = 0.25$ when we think that $X_j \to X_k$ is more probable.

With chunks we can model all sorts of structural priors. Prior structure knowledge can come in several forms. If we for instance could elicit the ordering of all or some of the variables in our domain we can also consider this as a constraint on the possible structures of the Bayesian network. The following example illustrates this.

Let $\mathbf{X} = \{X_1, \ldots, X_5\} = \mathbf{X}_1 \cup \ldots \cup \mathbf{X}_{10}$ be the set of variables and each chunk containing a single edge variable $\mathcal{C}^1 = \{\epsilon_2^1\}, \ldots, \mathcal{C}^{10} = \{\epsilon_5^4\}$. Assume that we have been able to elicit the ordering of the variables in $\mathbf{X}$ as $X_1, \ldots, X_5$. One way of expressing this, is by assigning for all chunks graphs of kth chunk, $\Pr(\bar{\mathcal{C}}^k) = \Pr(\{\epsilon_j^i = \mathsf{down}\}) = 0.5$ and $\Pr(\{\epsilon_j^i = \mathsf{none}\}) = 0.5$, $\Pr(\{\epsilon_j^i = \mathsf{up}\}) = 0$. This states that there could be an arc from a node lower in the ordering to a node higher in the ordering *or* there might not be an arc at all. Arcs from a node higher in the ordering to a node lower in the ordering is impossible, and we assign a probability of 0 to these chunk graphs.

If we only have an ordering of some of the variables, we can also model this in terms of chunks. We will not go into further detail here (see chapter 5).

If we combine chunk graphs we also want to be able to calculate the prior probability of the resulting graph. We defined a probability distribution on chunk graphs, but saw that the

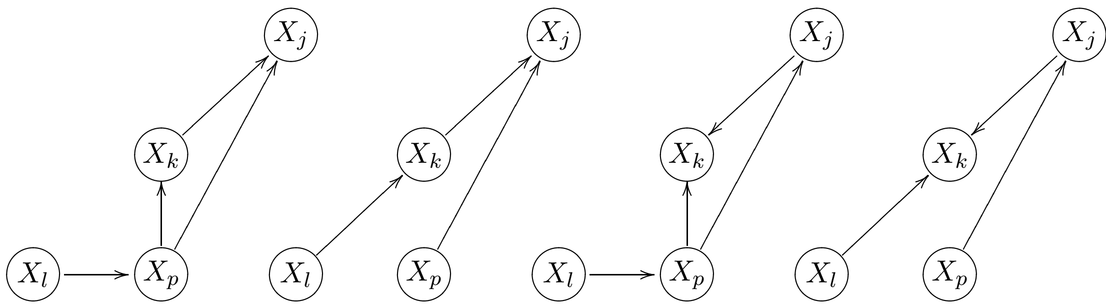


Figure 4.3: Combined chunk graphs (DAGs)

result of combining chunk graphs does not necessarily yield chunk graphs again. First realize that when estimating the probabilities $\Pr(\bar{\mathcal{C}}^i)$ we implicitly assume that the concerned chunks graphs potentially could be part of the final structure. When we estimate the probabilities we therefore implicitly assume that there at least is (are) *some* combination(s) of the chunks graphs that *can* form (a) combined chunk graph(s) of the concerned chunks, because after all the estimates is our belief that the chunk graphs indeed will be part of the final structure (when $\Pr(\bar{\mathcal{C}}^i) > 0$ at least). Also observe that chunks are units of knowledge thought to contain semantically closely related knowledge atoms. This means that different chunks must have different semantic properties or they would not be different chunks. This in turn means that chunk graphs of different chunks are independent of each other.[3]

We can thus conclude that the estimates of chunk graphs probabilities are (indirectly) conditioned on the fact that they could have been *derived* from the fully acyclic structure which we are inducing, and furthermore that chunk graphs of different chunks are *independent* of each other. We then have that *if* the combination of chunk graphs is again a chunk graph (and therefore acyclic) we can change the configuration of the chunks independent of each other. This means that if $\forall\, \bar{\mathcal{C}}^1 \cup \ldots \cup \bar{\mathcal{C}}^m \in \mathcal{S}_{\mathcal{C}^1 \cup \ldots \cup \mathcal{C}^m}$ we can write (implying $\bar{\mathcal{C}}^i \in \mathcal{S}_{\mathcal{C}^i}$):

$$\Pr(\bar{\mathcal{C}}^1, \ldots, \bar{\mathcal{C}}^m) = c \cdot \Pr(\bar{\mathcal{C}}^1) \cdot \ldots \cdot \Pr(\bar{\mathcal{C}}^m)$$

Here c is a normalization constant to make sure that $\Pr(\mathcal{S}_{\mathcal{C}^1 \cup \ldots \cup \mathcal{C}^m}) = 1$, i.e. that the prior probabilities of all possible combined chunk graphs indeed sum to unity (axiom of probability theory). The fact that some chunk graphs are not allowed in combination with each other forces us to introduce such a constant. We may think of c as the scaling constant (≥ 1) that "stretches" the distribution of the combined chunk graphs over the whole interval 0–1. When we take the logarithm of the above, we get:

$$\log \Pr(\bar{\mathcal{C}}^1, \ldots, \bar{\mathcal{C}}^m) = \log c + \log \Pr(\bar{\mathcal{C}}^1) + \ldots + \log \Pr(\bar{\mathcal{C}}^m)$$

For the corresponding prior DAG is then $\log \Pr(Bn_s) \equiv \log \Pr(\bar{\mathcal{C}}^1, \ldots, \bar{\mathcal{C}}^m)$.

If we combine the three chunk graphs in figure 4.2, we see in figure 4.3 that there are four combined chunk graphs with prior probability bigger than zero. All 4 (super) chunk graphs have the combined prior $1 \cdot 0.5 \cdot 0.5 = 0.25$. Had we assigned $X_j \to X_k$ of chunk $\mathcal{C}^2$ the prior probability 0.75, the last two chunk graphs would have a combined prior probability of $1 \cdot 0.5 \cdot 0.75 = 0.375$ and the first two chunk graphs a combined prior probability of $1 \cdot 0.5 \cdot 0.25 =$

[3]Independence has formal normative counterpart. In "reality" however this formal independence is never 100% satisfied. Descriptively we therefore handle a rather vague form of independence, and we just assume formal independence even though this might not be the case.

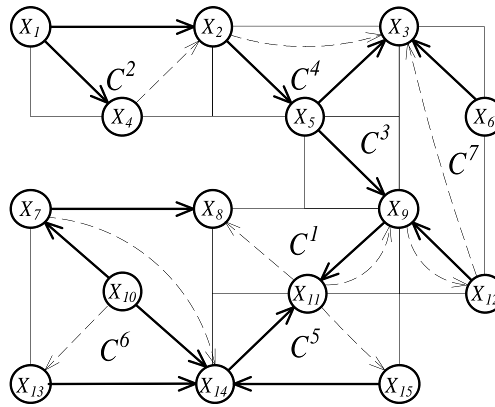


Figure 4.4: The idea behind chunking

0.125. In that case the last two DAGs would be the prior best choice. Notice that the probabilities of all possible combined chunk graphs sum to unity (no cyclic structures can be formed), and the normalization constant is $c = 1$.

4.3.3 Chunks as building blocks

The entire Bn_s structure we are trying to induce is now penalized according to the current configuration state of the chunks discernible in Bn_s.

The Bn_s of $Bn = (Bn_s, Bn_p)$ is the DAG structure implied by the combined chunk graph $CGr = (\mathbf{X}_1 \cup \ldots \cup \mathbf{X}_m, \bar{\mathcal{C}}^1 \cup \ldots \cup \bar{\mathcal{C}}^m)$ of $\mathbf{X}$. When we build structures our methodology is to change the configurations of those m chunks and effectively altering the sub structures of the potential Bn_s. Traditional induction methods usually employ three operators on single arcs only: adding an arc, removing an arc and reversing an arc (and sometimes removing and/or reversing an arc is not even considered). In our case we are handling chunks rather than single arcs, and we are constraining the possible configurations (not necessarily uniformly) of such a chunk and not 'wasting time' considering (sub) structures that are *a priori* improbable (impossible).

Edge variables that do not belong to any chunk are implicitly assigned the value none with a prior probability of 1, i.e. we exclude the possibility of there being an arc at all. If we are unsure about the configuration of an edge variable and want to let the database decide on its status, we would explicitly have to chunk the edge variable in a singleton chunk, or we could enclose it in an existing chunk somehow. The chunks are literally thought of as being the building blocks that constrain the possible combined structures we *can* construct. Anything that is not defined in terms of chunks (chunk graphs) will not be considered (hence implicitly assigned none) and will therefore not be part of the final combined structure.

In figure 4.4 the main idea behind chucking is depicted (of seven chunks). Dotted arcs illustrate potential arcs of other configurations of a chunk. The edge variable between for instance X_4 and X_7 does not belong to any chunk, and is assigned none with absolute certainty.

We can't noncritically change the configuration of a chunk $\mathcal{C}^k$ because this could introduce a cycle. We first have to find the common nodes of $\mathcal{C}^k$ and the rest of the chunks, $\mathbf{X}_k \cap (\mathbf{X}_1 \cup \ldots \cup \mathbf{X}_{k-1} \cup \mathbf{X}_{k+1} \ldots \cup \mathbf{X}_m)$ and then proceed with the analysis discussed earlier to see if we indeed stay acyclic.

Another issue to consider has to do with the connectedness of the chunk graph (or its corresponding DAG). When a node X_i of a DAG has a degree of zero it is independent of

any other nodes in the DAG. There is no relation between this node and any other nodes. It can't receive evidence, and can't 'send' evidence to the other variables. We can make a similar observation about sub-DAGs: If we have two disjoint sets of nodes such that all nodes in each set are connected and form two sub-DAGs, the two sub-DAGs are independent of each other if there is no edge between any (two) nodes of the disjoint sets. If we *a priori* assume that all variables of a domain are related somehow (why would they otherwise belong to the same domain?), it makes reasonable sense to assume that all nodes are connected: there exist a connection from any node X_i to any other node X_j of $\mathbf{X}$. We always have to test if changing the configuration of a chunk implies a disconnected DAG.[4]

Let's summarize what we have: Let $Bn = (Bn_s, Bn_p)$ be a Bayesian network, such that $Bn_s = (\mathbf{X}, \bar{E})$ is the corresponding *connected* DAG of the combined chunk graph $CGr = (\mathbf{X}_1 \cup \ldots \cup \mathbf{X}_m, \bar{\mathcal{C}}^1 \cup \ldots \cup \bar{\mathcal{C}}^m)$. The quality of the structure Bn_s (WLOG the PML_s measure here) with chunk structure priors is then defined as:

$$\begin{aligned}\mathrm{PML}_s(D, Bn) &:= \log \Pr(\bar{\mathcal{C}}^1, \ldots, \bar{\mathcal{C}}^m) + \sum_{i=1}^{n} \mathrm{pml}^D_{Bn_s}(X_i, \mathbf{P}_{X_i}) \\ &= \log c + \sum_{l=1}^{m} \log \Pr(\bar{\mathcal{C}}^l) + \sum_{i=1}^{n} \mathrm{pml}^D_{Bn_s}(X_i, \mathbf{P}_{X_i})\end{aligned}$$

This states that to calculate the quality of a network, we sum the quality of all n nodes, add the sum of the logarithm of the prior probability of all m chunk graphs that constitute the graph, and finally add the logarithm of the normalization constant c. To determine c we have to form all possible combined chunk graphs of the m chunks i.e. sum their combined probability to find out how much "to stretch the distribution" to be able to fill the whole range 0–1. Obviously it is generally infeasible to form all possible combined chunk graphs. In our case however we only use the PML_s (and PB_s) measure to compare potential structures and we can therefore scale the measure with a constant without any problems (positive monotonic transformation i.e. rank order-respecting). In practice this means that we can skip the calculations of c and just assign $c = 1$ (and $\log 1 = 0$) when comparing structures with the quality measures. If we want to interpret the quality measures as probabilities, we are however forced to calculate c.

If we are only modifying the configuration of some chunk $\mathcal{C}^l_{\mathbf{X}_l}$ of an already constructed network (and assume we stay acyclic and connected), we know that some nodes $\mathbf{Y} \subseteq \mathbf{X}_l$ potentially changed parent sets. For these nodes $\mathbf{Y}$ we recalculate $\mathrm{pml}^D_{Bn_s}(Y_i, \mathbf{P}_{Y_i})$. The remaining nodes have unaltered data quality. Furthermore we obtain the new combined prior by replacing the prior value of the old configuration with the new prior of the new configuration.

[4]Our demand that the graphs stays connected can be seen as a global constraint governing how chunks can be combined. Note though that in contrast to the global demand that the structure is acyclic, connectedness is not dictated by the Bayesian network formalism, but is a pure pragmatic constraint that seems to make sense when modelling real life domains (in fact there might be other appropriate global constraints depending on the knowledge domain).

Chapter 5

Implementation & search

This chapter describes how we have implemented the ideas about structural priors and induction in general as we have discussed in the preceding chapters.

Usually when dealing with induction of Bayesian networks a greedy search heuristic is applied that incrementally builds the structure, starting from a single node and adding arcs according to the increase in quality yielded. The *local neighborhood* is the set of potential structures that can be reached by the addition of a single arc from the current *partial solution structure*. The structure in the local neighborhood that increases the quality the most is chosen as the new structure, and the process is repeated. Because we are only considering the local neighborhood, the chance is high that we may choose to add an arc that from a local perspective seems like a good choice, but globally might be a bad choice. Obviously when we choose to add a certain arc at a given moment, we constrain the possible arcs we can add at a later point. We would have to 'backtrack' to undo some 'offensive' arc that we had previously added, and rebuild from that point on to escape a local optimum. Unfortunately we usually don't know which arc is the offensive one, and we would not know when to begin the rebuilding process. It is also possible to take a less 'local' approach when deciding on how (where) to add arcs, and then "look" several steps ahead instead of only one (as done in [2]).

Another way to find an 'optimal' solution, is to improve an existing *non-partial solution* by iteratively modifying the configuration states of such a solution. We thus start with a full structure and apply operations that alter the current configuration of a solution to obtain a 'better' solution. The changes we apply are guided by a quality measure (can be the same as in the greedy search heuristic). When building DAGs for our Bayesian network, we have already defined the chunks, and the possible configurations that the chunks can adopt. It is then a question of modifying the configurations of those chunks guided by the PML_s or PB_s quality. When we modify the configurations of a single structure with the *hill-climbing* method, we only consider alterations resulting in a quality *increase* compared to the present quality. With this method we might get stuck in a local maximum of our solution landscape[1] and we might have to restart the search from another starting point. We can also search the space in a less deterministic manner, and introduce a random element when choosing the next structure configuration. Of course the random perturbations may not overshadow the main search strategy, or we would end up with a completely random search. Random elements can be used to bias the decision weather to reject or accept an alteration of the chunks even though the new structure does not improve the quality. This method is used in for instance *simulated annealing*.

[1] Depending of the topology of the landscape, there are other problems in connection with hill-climbing.

5.1 Evolutionary algorithms

Conventional search methods usually concentrate on modifying or incrementally building one single structure (potential solution) at a time. When we alter or incrementally build the single structure we effectively discard the old structure in favor of the new superior one. Evolutionary algorithms (EA) is a whole range of optimization techniques[2] trying to mimic Darwinian evolution. They rely on a whole population of structures (*individuals*) that *reproduce*. We select individuals in the population according to their quality (*fitness*) and copy those individuals and slightly vary the configurations of the copies. The new copies are reinserted in the population. The operators that vary the copies are the recombination and mutation operators. The recombination operator recombines the selected *parent* individuals such that the new individuals (*offspring*) contain elements (*genetic material*) from the parents. The mutation operator changes the configuration of an individual with some small probability. Mostly the offspring produced by the parents undergo mutation with some small probability.

The idea of recombination is that fit parent individuals have good genetic material and that the offspring of such parents get good genetic material from all its parents and therefore could be even better (exploitation of existing genetic material). The function of the mutation operator is to introduce new genetic material into individuals (exploration of new genetic material).

The whole process is illustrated in figure 5.1 bellow. The iterative loop is repeated until some stopping criterion is met, for instance when the population converges (we hope that we have found the optimal solution) or after a predefined number of iterations has been reached.

Figure 5.1: General flow of EA

5.2 EA & chunks

An individual of the population pool $\mathcal{P}$ is the juxtaposition of all m chunk graphs $\bar{\mathcal{C}}^1\bar{\mathcal{C}}^2\ldots\bar{\mathcal{C}}^m$ which we call the genotype.[3] The phenotype is the connected DAG corresponding to the combined chunk graph of these m chunks. The chunks can be regarded as genes of the individual. Each gene can take on one of a number of possible values (*alleles*). In our setting this corresponds to a *subset* of the possible chunks graphs one can form of the chunk in question. We once again stress that we only want *valid* individuals. This means that any combination of chunk graphs inducing a cycle or making the structure disconnected is not allowed. A gene can therefore only take on a certain value if this yields an acyclic and

[2] We refer to `http://evonet.dcs.napier.ac.uk` for resources on evolutionary computation as well as [13].

[3] Actually an individual has other attributes as we will see later on. The m chunk graphs however form the 'genetic material' which fully defines the individual.

connected structure when combined with the rest of the genes of the structure. That is why we say that only a *subset* of the possible configurations are valid (at any given time).

The selection of parents for reproduction is based on the fitness. There are several selection schemes within the EA framework. The main idea is however that individuals with better fitness should have a relatively high probability of being selected, and less fit individuals should at least have a small chance of being selected. In our case the measure of fitness is the PML_s or PB_s measure discussed in the previous chapters. When we reinsert new individuals in the (new) population we have to decide which individuals to replace which we can do in a similar way as the reproduction selection strategies, but then biased towards the least fit individuals.

In the following sections we will discuss the operators and selection mechanisms in some detail.

5.2.1 Mutation & recombination

The recombination operator recombines the chunk configurations of parents, such that the new individual(s) receive configural information about the chunks from all the parents (note that parents do not necessarily have to come in pairs). Several (standard) recombination methods have been proposed. Unfortunately a recombination operator is very much problem dependent. Clearly we demand that the recombination of valid parent individuals should always return (a) valid child individual(s). In our case this means that if we recombine connected chunk graphs the result also has to be a connected chunk graph. We thus have to define a recombination operator that copies genetic material from the parents (configural information) and at the same time we have to make sure that the result is valid. The latter has higher priority than the former, because we at no cost want to introduce an invalid individual in the population pool.

A natural way of recombining would be to select a subset of chunks from all parents and put these chunks together to form a new individual. Of course it is quite likely that we may then end up with an invalid child. We can then try to recombine the same parents once again, but then with another chunk selection. Evidently we cannot try all different chunk selection permutations (which chunks from which parents to chose), and we will have to define an upper limit on the number of times to retry before we give up. We would then have to choose another set of parents from the pool and repeat the process. Now observe that it is easier to (re)combine (merge) two similar DAGs than two very dissimilar DAGs. A valid recombination is therefore easier if we recombine similar parents than different parents. We ran several tests with a recombination similar to the operator just described. In almost all of the cases the recombination failed (with several retries per parent set), and another parents set eventually had to be selected from the population. Unfortunately the recombination often only succeeded when very similar parents from the pool were chosen. But this violates the whole idea of recombination which exactly says that we have to exploit the difference of the parents. Eventually the whole population pool is filled with very similar parents, because similar parents breed similar offspring. The goal of keeping the population diverse to exploit the genes of different individuals to potentially create an optimal offspring is in fact counteracted by the recombination operator which drives the population toward similar individuals far too fast. We therefore don't profit from such an obvious (uniform) recombination operator because the chance is high that the entire population prematurely converges. Of course we might come up with a recombination operator that indeed works on DAGs, but it is not at all evident how to construct such an operator. We therefore refrain from using a recombination operator, and will only rely on mutation (and selection/replacement).

The mutation operator is traditionally the operator that explores new points in the solution space. It introduces new genetic material, which through (later) recombination can 'spread' to other individuals in the pool. It also plays a crucial role in connection with local maxima, as it can give the iterative process a 'kick' such that we may escape from a local optimum. In our case we don't use recombination, and we therefore rely on the mutation operator as the only variation operator.

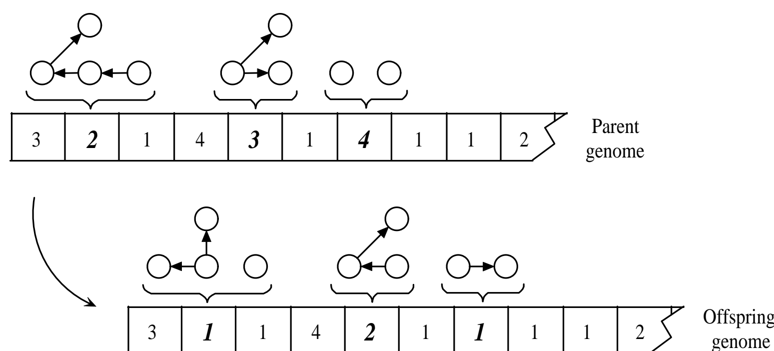


Figure 5.2: Mutation operator

The mutation operator operates on chunks where each chunk has a small chance of undergoing a configuration change. Of course we can only change the chunk configurations of the selected chunks if the change does not result in an invalid individual. Once we have chosen which chunks to mutate, we treat these chunks as a combined chunk graph, that is we form all possible combined (not necessarily connected) chunk graphs of these chunks, and try to combine each of these combined chunk graphs with the remaining part of the individual (which is guaranteed to also be a combined chunk graph). If we keep the mutation factor relatively small (relative to the number of chunks constituting an individual), and we have a limited number of possible chunk graphs per chunk (possible configurations) it is feasible to actually test all possible combined chunk graphs one can form of these mutation chunks against the rest of the individual. If we are dealing with chunks that can adopt many configurations, it can however be problematic to test all possible combined chunk graphs and we have to decrease the mutation factor accordingly, or only test a (random) fraction of the possible configurations. Figure 5.2 illustrates the mutation operator.

If there are no chunk configurations that in conjunction with the rest of the chunks form a valid individual, we can either leave the individual unaltered or we can return that the mutation failed (which in practice means to skip an iteration). If we return the individual unaltered, we risk that the process prematurely converges. It can be hard to alter a DAG structure, and the mutation operator will often give up and return the structure unaltered, and we will then have a copy of the parent individual which will be added to the pool. We therefore return that the mutation failed.

Notice that if we do not implement recombination, we are essentially only mutating single parents, and we need not select several parents at each iteration to produce offspring.

5.2.2 Selection & reinsertion

The selection operator is the factor that makes the individuals progress toward fitter individuals. If the selection pressure is too high, we prematurely concentrate on selecting highly fit individuals in the current population, ignoring the fact that the current population might

```
main(popSize, mutationFact, tour, killTour, generations) → ℒ

{Structure s consists of q_p prior quality, Q_d data quality of nodes}
{and {C̄^1, ..., C̄^m} the chunk configurations}

all-Vars-And-Sets-to-Nil()   {Set all vars/sets to 0/∅}

for 1 to popSize do   {Initialization}
    {C̄^1, ..., C̄^m} ← random-Connected-Chunk-Graph({C^1, ..., C^m})
    Ē ← to-Edge-Set({C̄^1, ..., C̄^m})
    q_p ← quality-Prior({C̄^1, ..., C̄^m})
    Q_d ← quality-Data(X, Ē)
    P ← P ∪ {[q_p; Q_d; {C̄^1, ..., C̄^m}]_s}
od

for 1 to generations do   {Evolve}
    s_old ← select-Parent(P)
    s_new ← mutate(s_old)
    if s_new ≠ fail then
        s_kill ← select-Replace-Individual(P)
        P ← (P\{s_kill}) ∪ {s_new}
    fi

    if log-Now() then
        ℒ ← ℒ ∪ log-Something(P)
    fi
od
```

not be the globally best. We therefore might select structures that now are good, but have chunk configurations that make it difficult to alter in a way that makes the fitness increase in future populations. The mutation operator can sometimes help us change the configuration of such 'tough' individuals, but this depends on how tough they are (remember that we only mutate a small number of randomly chosen chunks at a time). To give less fit individuals a chance to be selected, we use *tournament selection*. We select *tour* individuals at random (with replacement), and the best fit individual of the *tour* individuals is selected. In this way we bias selection toward more fit individuals. The size of the tour *tour* is a parameter that influences the selection pressure: if $tour = 1$ we select an individual at random, and if $tour = |\mathcal{P}|$ we select the fittest individual in the population.

We already mentioned that we need only select one parent at each step. This parent individual is mutated, and has to be reinserted in $\mathcal{P}$. We are therefore only changing $\mathcal{P}$ slightly per iteration (*steady-state EA*). Reinsertion means replacing another individual in the population (size of $\mathcal{P}$ is constant). We follow the *elitist strategy* which says that we should never replace the best individual in the population pool. We can nuance the idea of elitism by applying a bias towards replacing the least fit individuals, by using a "kill" tournament of size $killTour$. The least fit individual in the tour is then replaced by the new individual.

5.3 EA implementation

The pseudocode of the EA main program is given on page 36. It consists of an initialization section filling the population pool with random (connected) chunk graph individuals (but

---

$\mathsf{mutate}([q_p; \mathbf{Q}_d; \{\bar{\mathcal{C}}^1, \ldots, \bar{\mathcal{C}}^m\}]_s) \rightarrow [q_p^{new}; \mathbf{Q}_d^{new}; \{\bar{\mathcal{C}}^1, \ldots, \bar{\mathcal{C}}^m\}^{new}]_s$

---

if $pos > m$ **then**
  $pos \leftarrow pos - m$ {pos is globally declared}
  $[q_p^{new}; \mathbf{Q}_d^{new}; \{\bar{\mathcal{C}}^1, \ldots, \bar{\mathcal{C}}^m\}^{new}]_s \leftarrow [q_p; \mathbf{Q}_d; \{\bar{\mathcal{C}}^1, \ldots, \bar{\mathcal{C}}^m\}]_s$
  return {No mutation this time. Return unaltered}
fi
while $pos \leq m$ **do**
  $\mathbf{M} \leftarrow \mathbf{M} \cup \{\mathcal{C}^{pos}\}$ {Set of chunks to mutate}
  $\mathbf{X_M} \leftarrow \mathbf{X_M} \cup \mathbf{X}_{pos}$ {Set of touched nodes (recall that $\mathcal{C}^i_{\mathbf{X}_i}$)}
  $pos \leftarrow pos + \left\lfloor \frac{\log(1-\text{random}([0\ldots1[))}{\log(1-mutationFact)} \right\rfloor + 1$ {Find next chunk to mutate}
od

  {$\bar{\mathbf{M}}$ is current configuration of chunks in $\mathbf{M}$}
$\bar{E}^{\mathbf{M}} \leftarrow$ to-Edge-Set($\bar{\mathbf{M}}$)   {Edge set of current chunk config. to be mutated}
$\bar{E}^{\mathbf{R}} \leftarrow$ to-Edge-Set($\{\bar{\mathcal{C}}^1, \ldots, \bar{\mathcal{C}}^m\} \backslash \bar{\mathbf{M}}$)   {Remaining part of current chunk config.}

for each possible random $\bar{\mathbf{M}}^{new} \neq \bar{\mathbf{M}}$ of $\mathbf{M}$ **do**
  $\bar{E}^{\mathbf{M}^{new}} \leftarrow$ to-Edge-Set($\bar{\mathbf{M}}^{new}$)   {Edge set of mutation chunks}
  if acyclic($\mathbf{X_M}, \bar{E}^{\mathbf{M}^{new}}$) **then**   {Mutation graph is acyclic}

    {Combine new mutated sub graph and rest of old graph if valid}
   if connected($\mathbf{X}, \bar{E}^{\mathbf{M}^{new}} \cup \bar{E}^{\mathbf{R}}$) **and** acyclic($\mathbf{X_M}, \bar{E}^{\mathbf{M}^{new}} \cup \bar{E}^{\mathbf{R}}$) **then**
    $\{\bar{\mathcal{C}}^1, \ldots, \bar{\mathcal{C}}^m\}^{new} \leftarrow (\{\bar{\mathcal{C}}^1, \ldots, \bar{\mathcal{C}}^m\} \backslash \bar{\mathbf{M}}) \cup \bar{\mathbf{M}}^{new}$ {New chunk configs.}

    $\mathbf{N} \leftarrow$ find-New-Parents($\bar{E}^{\mathbf{M}}, \bar{E}^{\mathbf{M}^{new}}$)   {Nodes with new parents...}
    $\mathbf{Q}_d^{np} \leftarrow$ quality-Data($\mathbf{N}, \bar{E}^{\mathbf{M}^{new}} \cup \bar{E}^{\mathbf{R}}$)   {... data quality of those node}
    for each $v \in \mathbf{N}$ **do**   {Replace old quality with new quality}
     $\mathbf{Q}_d \leftarrow \mathbf{Q}_d \backslash \{Q_v\}_d$
    od
    $\mathbf{Q}_d^{new} \leftarrow \mathbf{Q}_d \cup \mathbf{Q}_d^{np}$

    $q_p^{new} \leftarrow$ quality-Prior($\{\bar{\mathcal{C}}^1, \ldots, \bar{\mathcal{C}}^m\}^{new}$)   {Prior quality of chunks}
    return
   fi
  fi
od

return fail   {$\bar{\mathbf{M}}^{new} = \bar{\mathbf{M}}$ thus no new valid config. found. Return fail}

---

constrained by the prior information of course) and of a section that iteratively selects, mutates and replaces individuals in the pool. Note that a mutation can fail, and that this implies that no individual is inserted (see below).

The process is logged somehow — probably the first iterations are not interesting so that we can log when we think the process becomes interesting (whatever that may be). We only logged the fittest individuals (there can be several individuals in the pool exhibiting the same quality) at each iteration. When the process stops $\mathcal{L}$ consists of the best individuals throughout the iterative process. The individuals with the highest quality in $\mathcal{L}$ are usually chosen as the result of the EA algorithm.

The mutate function (method) is the most interesting part of the program. The pseudocode of the mutation operator is given on page 37. It consists of 3 parts. The first part selects

the chunks for mutation (see the appendix for further details). The second part runs through all possible configurations of the selected chunks (randomly!) until an acyclic configuration is found, and finally tries to recombine with the rest of the chunk graph. If this does not succeed, another acyclic configuration of the mutation chunks is tried out. If we have tried out all configurations and none could be combined with the rest of the chunk graph, the mutation operator returns that the operation failed. This means that the individual "has had its chance" to be mutated, but failed. At the next iteration step the individual *could* be selected again. The third part of the function is executed when a valid combination is found. Subsequently the qualities of the nodes and priors are updated according to a quality measure discussed in chapter four.

We don't keep the prior probability of all the m (sub) chunk graphs an attribute of the individual, because it is computationally cheap to recompute (just the logarithm of a floating point) at each alteration. The quality of the n nodes are of course all kept per individual, as it expensive to calculate due to the fact that it requires querying the database several times (depending on the size of the database, the calculation of the data quality is the most time consuming computation of the whole EA process).

The EA algorithm requires us to chose several parameters: *popSize*, $mutationFact$, *tour*, *killTour* and *generations*. None of these parameters are known beforehand, but — to a certain degree — have to be found (or at least fine tuned) empirically. The parameters are not necessarily independent of the prior information supplied, as we will see later on.

5.4 Experimental settings

Our experiments were based on the Asia network from chapter two. We used the program Hugin Expert Lite[4] to create a sample database. The DAG structure and the corresponding local probabilities were entered, and a database of 500 records was sampled with no missing values. Throughout we used the PML_s quality measure because the PB_s along with the $BDeu$ metric introduces the parameter α which is very sensitive to small variations (and we would have yet another parameter in addition to the already introduced EA parameters). We used the penalty function $p(N) = \frac{1}{2} \cdot \log(N)$.

With the EA algorithm we ran several experiments based on the following different prior knowledge settings:

1. We chunked the Asia-DAG (!) such that each chunk contained a single edge variable. We thus had $8 \cdot (8-1)/2 = 28$ chunks. From each chunk we can form 3 chunk graphs (no cycle possible with one edge). We assigned a uniform prior to all chunk graphs of each chunk (1/3). This setting corresponds to no prior structural information about the network.

2. We chunked the DAG with each chunk containing one edge variable. We then supplied an ordering of the variables in terms of prior probabilities so that for instance $\Pr(\{\epsilon_2^1 = \mathsf{down}\}) = 1/3$, $\Pr(\{\epsilon_2^1 = \mathsf{up}\}) = 1/3$ and $\Pr(\{\epsilon_2^1 = \mathsf{none}\}) = 1/3$ but $\Pr(\{\epsilon_3^1 = \mathsf{down}\}) = 1/2$, $\Pr(\{\epsilon_3^1 = \mathsf{up}\}) = 0$ and $\Pr(\{\epsilon_3^1 = \mathsf{none}\}) = 1/2$.

3. The variables X_4, X_5 and X_6 are obviously semantically related and we should treat the edge variables between those variables as a chunk. The variable X_6 will not have any arcs to X_4 and X_5 (X_6 actually behaves as a logical 'or' connective). Furthermore we assume that the probability of there being no edge between X_4 and X_5 is 0.8. The

[4]The 'lite' version is obtainable as freeware from `http://www.hugin.dk`.

rest of the 25 edge variables are all chunked as singleton sets. We also assume that it is improbable that there will be an arc to the observable variables X_1 and X_2. In terms of chunk graphs probabilities this translates into (singleton chunks not mentioned receive a uniform prior on the chunk graphs):

$$\begin{aligned}
\Pr(\{\epsilon_5^4 = \mathsf{up}, \epsilon_6^4 = \mathsf{down}, \epsilon_6^5 = \mathsf{down}\}) &= 0.1 \\
\Pr(\{\epsilon_5^4 = \mathsf{down}, \epsilon_6^4 = \mathsf{down}, \epsilon_6^5 = \mathsf{down}\}) &= 0.1 \\
\Pr(\{\epsilon_5^4 = \mathsf{none}, \epsilon_6^4 = \mathsf{down}, \epsilon_6^5 = \mathsf{down}\}) &= 0.8 \\
\\
\Pr(\{\epsilon_2^1 = \mathsf{none}\}) &= 1 \\
\\
\Pr(\{\epsilon_j^1 = \mathsf{none}\}) &= 0.5 \\
\Pr(\{\epsilon_j^1 = \mathsf{down}\}) &= 0.5 \\
\\
\Pr(\{\epsilon_j^2 = \mathsf{none}\}) &= 0.5 \\
\Pr(\{\epsilon_j^2 = \mathsf{down}\}) &= 0.5
\end{aligned}$$

Due to the stochastic nature of EAs, we ran our algorithm many times with each setting to get a reliable impression of the process. Note that we did not calculate the probabilities Bn_p of the structures we induced (but this can is easily be done once we have the right structure).

5.5 Results & discussion

Before we applied the full EA methodology, we implemented a hill-climbing technique on one single individual (slight modification of the main program on page 36). The mutation operator was the same as in the EA case. Some chunks are (randomly) picked for mutation, and a (acyclic) configuration of these chunks maximizing the quality compared to the current quality is chosen. When no quality increase was possible, we tried to mutate the original individual again (again randomly chosen chunks) until a predefined number of retries had been reached. If a better configuration is indeed found, the process is repeated on the new superior individual. If the predefined mutation retry threshold has been reached and no quality increase (or valid combination) was found, we return the current individual as the optimal one. We chose a mutation factor of 0.1 (same factor as we used in the EA case — see later) and the upper limit on the number of times to retry to mutate was varied. We used the setting with no prior information to gain insight in how this implementation of hill-climbing behaves.

The results fluctuated quite a lot. When we chose to retry mutation < 25 times, an insignificant increase in quality was observed. Sometimes the returned structure was the same as the initial structure. With a retry 25–100 the gain in quality was sometimes big compared to the initial structure, but the structure returned had no similarity with the original (Asia) network whatsoever. When we retried mutation 100–125 times the increase in quality was often big, but was still quite unstable between runs and still clearly suboptimal (quality of -532 was the best found — in the optimal EA case this was -426 as we will see later). Increasing to > 150 did not seem to have any significant effect (sporadic behaviour), and the increase in quality was generally not bigger than with a retry of 125. The big fluctuations of the quality, as well as the very dissimilar topology of the 'optimal' structure returned, suggests that the hill-climbing is stuck somewhere in a local maximum and that the mutation operator can't

help escape. The process never considers 'improvements' that lower the overall quality, and as a result is too eager to reach a maximum.

The EA method is similar to this hill-climbing process, but in essence has several processes running in parallel (and EA mutate no matter quality gain or loss). The fact that we use a population pool and bias selection toward better individuals (and bias replacements toward less fit individuals) makes it easier to escape local maxima, because we don't optimize a single individual, but let all individuals have a (biased) say — also bad individuals. Each individual explores the solution space, and all the individuals can pull the rest of the individuals toward an area of interest. This "pull factor" makes the algorithm less eager, because all individuals in the pool "pull" in different directions. Fit individuals are selected more often, and consequently have a "stronger say" than less fit individuals that are selected less often (but they are still given a chance).

5.5.1 EA results

Throughout the number of generations was set to 25.000 for testing purposes, which in reality turned out to be an exaggerated number of iterations. Based on several logged runs in all three settings the optimal quality generally did not improve beyond generation 15.000 (with a few exceptions). Since mutation sometimes may fail, some iterations will not produce any new offspring, and the number of 'real' generations is therefore less than 15.000.

When prior information was available (setting 2 and 3) the population converged faster than when no priors were given (setting 1) all other parameters being equal. This can be explained by the fact that the search space has been narrowed considerably when priors are used, especially the ordering. The population size was initially set to 100 individuals, but eventually lowered to 60 in setting 2 with no obvious loss of performance. In setting 1 and 3 however the population size seemed to drastically influence the converging properties of the process, and lowering the population size implied premature convergence with what seemed sporadic random progress. To see why this is, we inspected the population pool. It turned out that all individuals had the same quality yet different chunk configurations. This means that all individuals according to the quality measure should be able to represent the same joint distribution underlying the database. With such a 'flat' population the EA algorithm was actually doing 'random walks' selecting parents from the pool at random, without being able to impose any selection pressure. Of course this sometimes results in an increase in quality, but only by chance. In a situation like this the EA cannot select good parents based on data quality alone, but has to have additional prior information which differentiates individuals that otherwise have equal data quality. When the population size increased there was 'room' for other individuals with different data quality. We however saw in chapter 2 that there are many equivalent[5] DAGs (although we are only considering connected DAGs here), so the problem may persist even with a huge population pool. Setting 3 also suffered from the same problem, but less often. In that setting we did not supply much prior knowledge, and there are still many degrees of freedom that the data alone has to account for (fill in).

The selection pressure controlled by the size of the tournament was varied from 2 to 7. In setting 2 with the population pool of 60, a tour of size 3–4 worked well, and generally the EA converged to the same optimal value. A value of 2 made the process converge more slowly, and sometimes all population individuals ended up having almost the same quality (and we are then doing random walks). In setting 1 and 3, with populations size of 100, a tour of size 2 was definitely too small as the process never seemed to return stable results. When increasing

[5] Actually the quality measure might also assign the same quality to DAGs that are not equivalent. On the basis of the counts in the database however, such networks seem to be able to account for the underlying joint distribution.

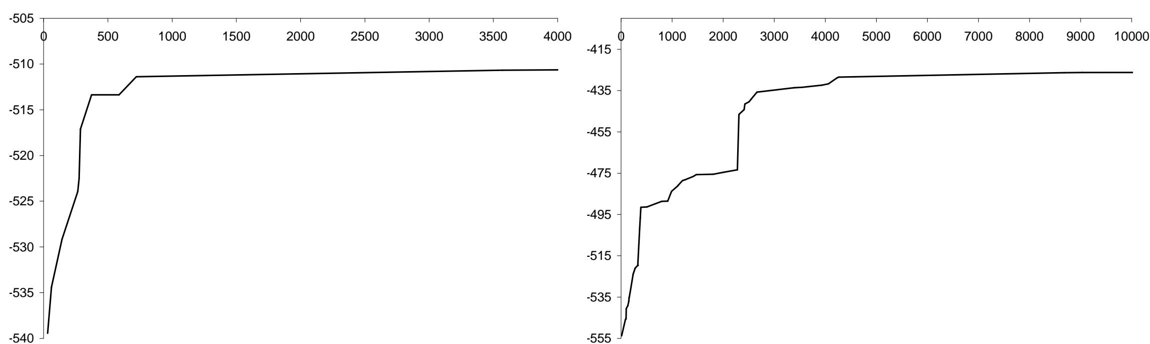


Figure 5.3: Quality as function of iterations. Left: setting 2. Right: setting 1.

the tour to 4–6 in setting 1 and 3 the process returned better individuals. In all settings a tour of 7 generally (but not always in setting 3) resulted in suboptimal quality, and the population would converge too fast. The kill tournament size was difficult to asses. It did not seem to have any significant influence on the obtained results when changing between 2–10. It did however seem to have a 'smoothing' effect. The sudden increases in quality observed when the value was > 8 high, was much less pronounced when the value was 2. When it was set to 2–4, the quality increase progressed in a more stable manner. A value of 4 was chosen as a compromise, but generally this parameter seems to have a minimal influence. We also tried a value of 1 meaning that an individual is replaced at random — including the best individual. Generally this also worked, but it took more iterations (2–3 times more) than with elitism to reach an optimal quality.

The mutation factor was varied in the range 0–0.15. A value less that 0.05 meant in most cases suboptimal quality. The quality initially seemed to steadily converge toward an optimal value, but it usually stopped within the first 2.000 iterations. Values around 0.08–0.1 generally yielded more stable results. Values between 0.1 and 0.15 sped up the process and the optimal quality was sometimes returned. Mostly however the population converged too fast and eventually got stuck. There were no obvious differences between the 3 settings. They all behaved the same with the same mutation factor.

In conclusion, the parameters found to be good, i.e. yielding stable results between different runs and high quality, were for setting 2 $popSize = 60$, $mutationFactor = 0.09$, $tour = 4$ and $killTour = 4$, and for setting 1 and 3 $popSize = 100$, $mutationFactor = 0.1$, $tour = 5$ and $killTour = 4$. We would like to note that 'stable results between runs' means that the optimal quality returned of different runs were the same within the range ± 5 (with exceptions!).

The graphs in figure 5.3 depict optimal quality (best fit individual in $\mathcal{P}$) as a function of iterations for setting 1 and 2 (typical behaviour). We see that the optimal quality for setting 2 is reached within 3.000–4.000 iterations, and converges around -510. If we are totally ignorant and have no prior knowledge, as is the case in setting 1, we see that an optimal quality -426 is reached somewhere between 9.000–10.000 iterations. We should however not compare the optimal quality between the two settings, but rather the optimal networks they return. For setting 1 is a structure with maximal quality not necessarily the structure that has most similarities with the network that created the database (the Asia network). The quality is a measure of how well the induced network could have created the database, not the original network. The more fit an individual is the better it can account for the database. We are in fact fitting the individuals to the database, and not to the original network. With a finite sized database we should not fit the database in every detail (which would result in an optimal

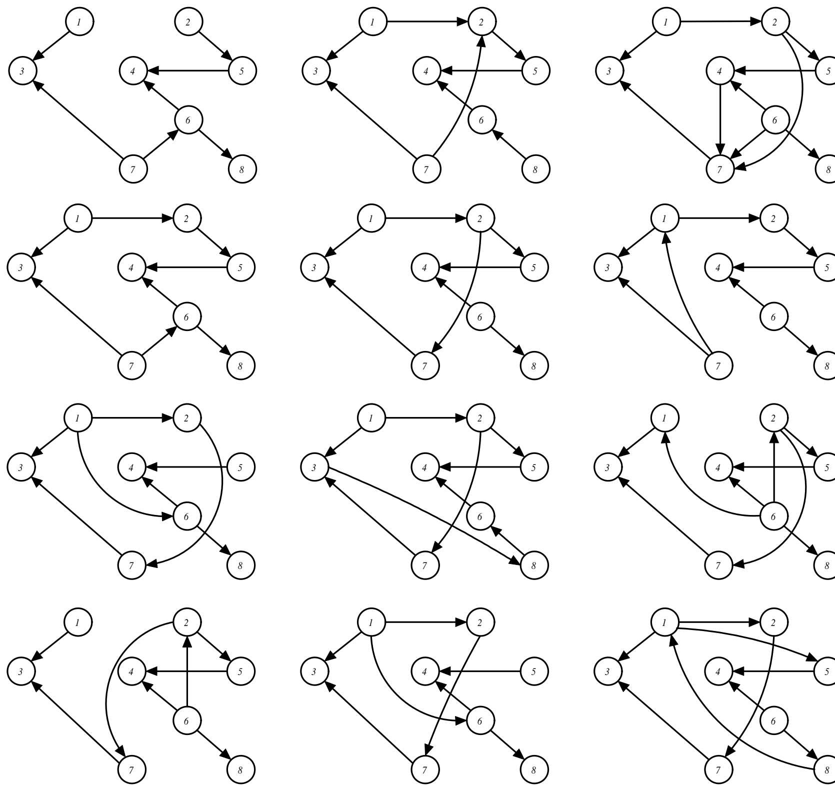


Figure 5.4: Some DAGs from setting 1

quality). Setting 2 constrains the search such that we don't necessarily capture every detail of the database, but rather obtain a more recognizable structure. The recognizable structure may however not be the structure that is optimally supported by the database. The only 'fair' comparison between the two settings would be to judge the resulting networks on similarity with the original network.

As already mentioned, there are several different (not necessarily equivalent) structures that have the same quality. Although the optimal quality of the pool might not increase (over a number of iterations), structures might in fact still be introduced that have different chunk configurations, but in a way that it does not affect the optimal quality of the pool i.e. at any given time there are potentially several individuals in the pool that are all equally optimal. When comparing different settings we therefore have to compare a whole set of equally optimal DAGs[6] with a set of equally optimal DAGs from another setting. Comparing a single optimal structure from each setting can give a wrong impression.

The DAGs depicted in figure 5.4 are a selection of some equally optimal individuals (-426) from setting 1. Around 45 DAGs were returned that had this quality (some being equivalent), all bearing resemblance to the original network. The upper left DAG seemed to be the 'best' DAG induced. Many of the optimal DAGs agree on edges that can be found back in the original network ($X_1 \rightarrow X_3$ for instance), but on some arcs no general consensus seems to exist.

Figure 5.5 shows a selection of equally optimal DAGs (-510) found in setting 2. Around 35

[6] All equally optimal DAGs that the EA has 'come across' before the quality improved — not necessarily *all* possible DAGs with this quality. The EA will often produce a superior individual before having found all equivalent (or at least seem equivalent to the quality measure) structures with equal quality.

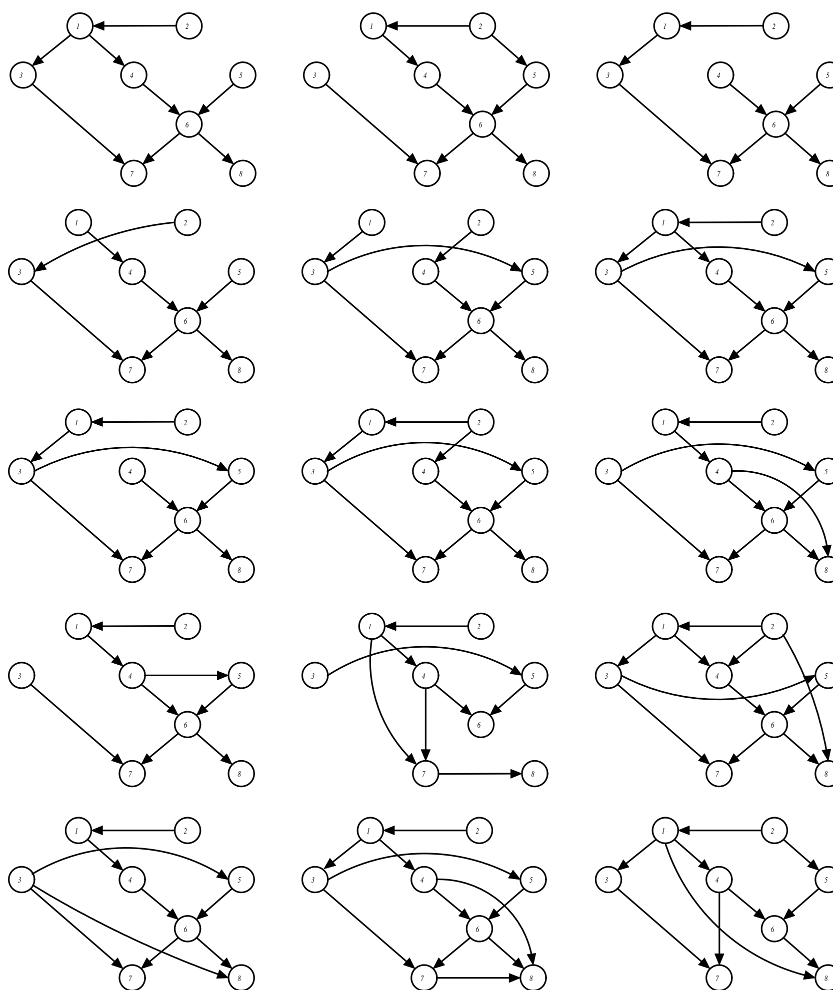


Figure 5.5: Some DAGs from setting 2

networks were returned by the EA all having the same quality. The upper left DAG definitely bears a striking resemblance to the Asia network, only missing an arc from X_2 to X_5 and instead has an arc between X_2 and X_1. There are however DAGs that have too many arcs compared to the Asia network, but all DAGs certainly have a lot in common with the original network.

In figure 5.6 the behaviour of setting 3 is plotted, as well as four optimal DAGs selected from a set of about 20 optimal DAGs. An optimal quality of -478 is reached between 8.000 and 10.000 iterations. We see that the DAGs are quite similar to the Asia network. The plot on the right depicts the number of edges as a function of iterations, i.e. the number of edges of the optimal individual in the pool at a given iteration step. We see that the initial best network has 15 edges, but from iterations 2.000 and onward optimal individuals will have between 8 and 9 (sometimes 10) edges.

Generally we see that supplying prior (structural) knowledge yields structures that have more in common with the original network than when supplying no prior knowledge. In setting 2 and 3 a subset of the DAGs induced look very similar to the Asia network. In setting 1 however we can also discern some structural similarities with the Asia network, especially when we ignore the direction of the arcs. It is however obvious that some post-

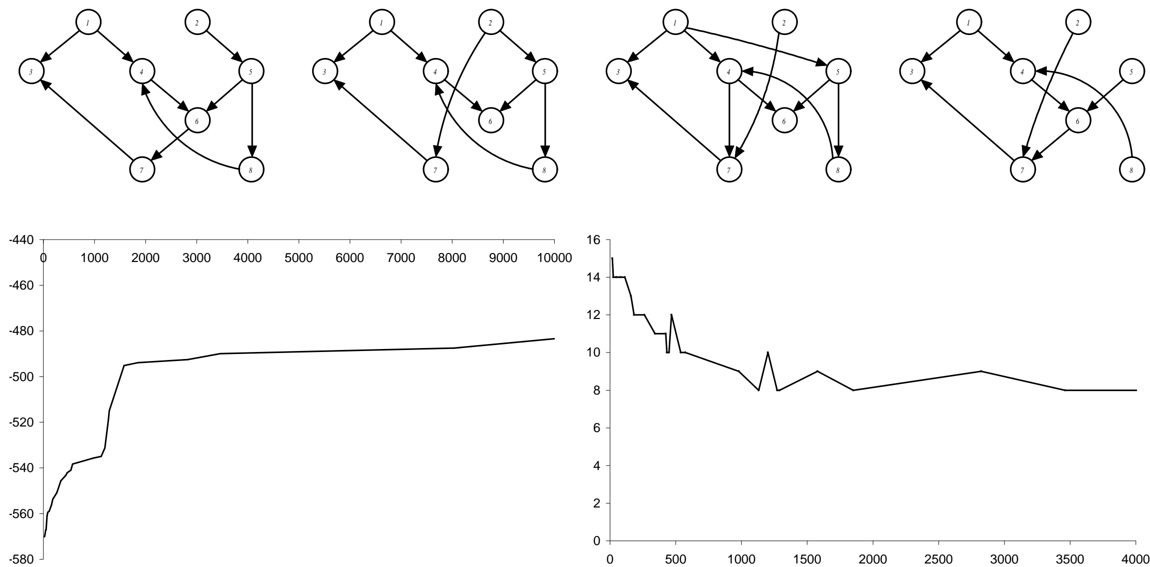


Figure 5.6: Setting 3: some DAGs and quality/no. edges as function of iterations.

processing work of the induced structures is desirable (algorithmic or in collaboration with domain experts) in all settings before we have a total match with the original Asia network.

5.5.2 General remarks

All experiments were run the full 25.000 iterations on a 550 MHz Pentium III based PC. EAs are pretty greedy when it comes to computing time, and it took on average about 12 minutes to complete the 25.000 iterations. The computation of the data quality is time consuming, but the decomposition property of the quality measure makes the process faster.

Yet, there are many individuals that are eventually replaced, and along with them the quality of their nodes, some of which later will be reintroduced and computed again in new individuals. Of course the EAs don't return a single best individual as a result as is the case with incremental search heuristics. Realizing that we are not dealing with partial solutions, but that all individuals correspond to connected DAGs of all the nodes, we are actually presented with several potential solutions differing in quality or/and topology. If we log the fittest individuals in the population pool from a certain point, say from 12.000 iterations and onward, we can later pick the DAGs we think are most appropriate. This is a big advantage compared to traditional methods, especially in the context of expert verification and validation of the structures. We can present the expert with a range of good networks (often different topologies with the same quality as we saw), and the expert can pick the one she is most comfortable with. Yet another possibility is to use model averaging briefly discussed in chapter 3. We are namely dealing with a set of good networks, and combining their inference result may yield a more trustworthy result (and we can even average the networks obtained from different runs).

The search space (landscape) we are exploring consists of connected DAGs (the phenotype at least). We don't allow any disconnected DAGs in the population. This constraint also makes it harder to mutate (and recombine). We could also allow disconnected DAGs (they are allowed within the Bayesian network formalism), but this would make the search space much bigger. Similarly we could allow cyclic DAGs when opting for a 'dumb' recombination

operator, and subsequently somehow 'repair' the offensive chunks (or arcs) before we add the offspring to the population pool (or pass it on to the mutation operator). Alternatively cyclic individuals could be punished somehow and would then have a smaller chance of being selected. It might very well be that cyclic individuals have chunk configurations that indeed are very good, and cyclic individuals should consequently have at least a small chance of being selected for recombination.

A big improvement would be to search the (connected) equivalent DAGs solution space. This would narrow the search space, and would allow for a smaller population pool.

Chapter 6

Conclusion

In this thesis we have concentrated on three topics, namely (i) the theory needed to induce Bayesian networks from data, (ii) how to model structure priors in terms of chunks, and (iii) how to actually induce Bayesian networks with EAs using chunks. Here we will add some final remarks and draw conclusions about the issues investigated in this thesis.

We discussed two ways of inducing Bayesian networks: the frequentist and the Bayesian approach. The frequentist approach is straightforward, whereas the Bayesian approach is a more novel approach founded on nicer theoretical principles. Furthermore the Bayesian approach has a nice way of handling prior knowledge, and should therefore be the natural choice when we model expert knowledge. Unfortunately theory is one thing, and practice another. The virtual counts needed to obtain the prior uncertainty distribution are practically impossible to asses. The sheer number of counts needed makes them impossible to elicit from domain experts, and above that, the counts have also shown to have an excessive influence on the networks found. They are simply too sensitive, which also makes them unsuitable for expert assessments. A non-informative prior can then be used, but the whole point of the Bayesian approach is then lost.

From a cognitive point of view, chunks are plausible building blocks when inducing Bayesian networks. It therefore makes sense to model prior structural knowledge in terms of chunks. In practice the chunking might be governed by *real* knowledge chunks, but can also be used in a more *artificial* manner, and just as a practical way of grouping edges even though the edges don't connect semantically related concepts (variables). We however assume that the chunks are more or less independent of each other (assuming we stay valid), which can be disputed when we are not dealing with closely semantically related knowledge concepts.

Anderson et al. have suggested that there exists standard chunks for different constructs. This idea could maybe be used in knowledge acquisition (acquiring prior structural knowledge), instead of the single relational approach used today when constructing the DAG in collaboration with domain experts.

The probabilities defined on chunk graphs have to be elicited from domain experts somehow. Of course the uniform probability assignment can be used, but the gradual scale 0–1 could maybe be exploited in a better way. It is however a well-known fact that probability assessments in general can be quite problematic [6] — the estimates of the structural priors might be too difficult.

It is evident that when we want to use prior expert knowledge we have to make clear *what* experts can, and what they can't verbalize or otherwise express. Only then we can tailor algorithms and methods that fill in the 'shortcomings' of the expert. More research is needed combining well established cognitive models (such as ACT or ACT*) and ideas from computer

science, if we want to use Bayesian networks for serious decision support.

The EA implementation with chunks seems to work quite well. It is inherent of EA algorithms that the parameters have to be fine tuned, and that we therefore cannot make a 'general' EA that works in all settings. In contrast to search heuristics that incrementally build structures, EA handles full structures varying in quality, is less prone to local maxima (depending on the parameters) and additionally offers a convenient framework for implementing chunks. These points outweight the computational time needed to run EAs. When dealing with big networks (many nodes) this might however be a problem. Global constraints then have to be made, for instance an upper limit on the number of parents of the nodes.

The annoying equivalence property of Bayesian networks is problematic, not only in connection with EAs. Equivalence has been recognized as a major obstacle of efficient induction of Bayesian networks, and is an active area of research.

In conclusion, we have developed a method of specifying *a priori* structure knowledge in terms of chunks, and we have shown that evolutionary algorithms is a suitable framework for inducing the structure of Bayesian networks from databases when using chunks as building blocks. Our experiments show that constraining the induction by supplying prior knowledge in terms of chunks indeed yield more recognizable structures compared to an ignorant setting with no prior knowledge.

Bibliography

[1] J.B. Best, *Cognitive Psychology*, Brooks/Cole Wadsworth, London, 1999.

[2] R. Bouckaert, *Bayesian Belief Networks: from Construction to Inference*, PhD thesis, Faculty of Mathematics and Computer Science, Utrecht University, Utrecht, 1995.

[3] G.F. Cooper, C. Glymoure, *Computation, Causation & Discovery*, MIT Press, London, 1999.

[4] G.F. Cooper, E. Herskovits, *A Bayesian Method for Induction of Probabilitstic Networks from Data*, Machine Learning, 9. 309–347, 1992.

[5] R.G. Cowell, A.P. Dawid, S.L. Lauritzen, D.S. Spiegelhalter, *Probabalistic Networks and Expert Systems*, Springer-Verlag, New York, 1999.

[6] M.J. Druzdzel, *Probabilistic Reasoning in Decision Support Systems: From Computation to Common Sense*, PhD thesis, Dept. of Engineering and Public Policy, Carnegie Mellon University, Pittsburgh, 1993.

[7] P.G. Hoel, S.C. Port, C.J. Stone, *Introduction to Probability Theory*, Houghton Mifflin Company, Boston, 1971.

[8] L.C. van der Gaag, S. Renooij, *Probabalistisch redeneren*, Syllabus, Faculty of Mathematics and Computer Science, Utrecht University, Utrecht, 2001.

[9] D. Heckerman, *Tutorial on learning with Bayesian Networks* in M.I. Jordan, *Learning in Graphical Models*, Kluwer Academic Publishers, Doordrecht, 1998.

[10] D. Heckerman, D. Geiger, D.M. Chickering, *Learning Bayesian Netorks: The Combination of Knowledge and Statistical Data*, Technical Report MSR-TR-94-09, Microsoft Research, 1995. (obtainable from anonymous FTP).

[11] F.V. Jensen, *Bayesian Networks and Decision Graphs*, Springer-Verlag, New York, 2001.

[12] T. Kočka, *Graphical Models: Learning and Applications*, PhD thesis, Faculty of Informatics and Statistics, University of Economics, Prague, 2001.

[13] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Berlin, 1996.

[14] R. Tamassia, I.G. Tollis, *Graph Algorithms and Applications I*, World Scientific Publishing Company, Singapore, 2002.

Appendix A

Implementational issues

In the following subsections we will discuss different issues regarding the implementation of the methods and data structures.

A.1 Edge representation

For the data structure of the edges $\bar{E}$ of $Gr = (\mathbf{X}, \bar{E})$ we use an $|\mathbf{X}| \times |\mathbf{X}|$ adjacency matrix of bits implemented as an $|\mathbf{X}|$ dimensional array of $|\mathbf{X}|$ bit integers. Index a_{ij} of the 'matrix' is set to 1 (true) if there is an arc from node X_i to node X_j. Note that with an integer representation of the rows, we can obtain the first (last) node receiving an arc from node i by taking the integer part of the logarithm base 2 of the integer (row) representing the ith node. The following node receiving an arc from i is obtain by simply XOR masking the integer of node i with the already found nodes. This representation allows for an efficient implementation of the edge set, as we don't have to loop through all indices of the array to find the following node that receives an arc (plus we can implement many of the methods/functions operating on graphs efficiently by exploiting binary operators on bit level).

A.2 Acyclic and connected tests

To check for cycles we use a *depth-first search*[1] strategy. We recursively follow the arcs in the depth, marking all seen nodes, and if we encounter a node we have visited before we are have found a cycle. If we have exhaustively visited all descendant nodes of a node, we mark the node as closed, and recursively backtrack (closing all nodes on our way back) and repeat the process until we return to the common node from where we began. We then restart the search from another common node (if it has not already been closed). Note that we only have to depart from (a subset of) the common nodes to check for a cycle (as discussed in chapter 4). If we have closed all (except one — one node cannot form a cycle) of those nodes we can conclude that none of the common nodes are part of a cycle.

To check if the structure is connected, we simply start at node 1, and follow the *edges* (also against the direction of an arc) marking the nodes we pass until we cannot reach any unmarked nodes anymore. When we backtrack to the 1st node (recursively) we have either marked all nodes or we haven't. If we have indeed marked all nodes (we keep a count of the number of nodes marked), we are dealing with a connected structure, otherwise we are not.

[1] We refer to [14] for further details on general graph algorithms.

A.3 Population pools

The parent pool consists of structures with three attributes:

1. An array of the same dimension as the number of chunks with the current configuration of all the chunks.
2. An $|\mathbf{X}|$ dimensional array containing the data quality of all the nodes of the structure.
3. A variable containing the combined prior probability of the super chunk graph based on the configurations of the chunks *plus* the sum of the array with data quality of the nodes. This is the *fitness field*.

Tournament (kill) selection does not rely on a sorted population pool (we can do with a few comparisons when we have selected the tournament members to pick the best/least fit member), and we need therefore no sorting mechanism to rearrange population members. We implement the pool as an array of structures with the same dimension as the population size. The (kill) tournament members are gathered in a similar array (of size dependent on the tounament (kill) size).

A.4 Mutation points

Each chunk can undergo mutation depending on the mutation factor f_m. Evidently we could loop through all chunks and draw a random number, and mutate if we are under the mutation factor threshold. If we realize [7] that the distance I between two "event points" where the probability of an event occurring at each (discrete) point is f_m has a geometric distribution $I \sim \mathrm{ge}(f_m)$ we can find the next event point, from the current point, by (we round to highest integer less than or equal to) $pos = \frac{\log(1-u)}{\log(1-f_m)} + 1$, where u is drawn uniformly from $[0 \ldots 1[$. The next point to mutate is thus *pos* positions ahead of the current mutation point. This is a more efficient way of finding mutation points compared to looping through all positions where we would have to draw a random number each time.