Vrije Universiteit Brussel

Faculteit Ingenieurswetenschappen

Learning Causal Models of Multivariate Systems

and the Value of it for the Performance Modeling of Computer Programs

Ph.D. Dissertation

Jan Lemeire

Supervisor: Prof. dr. ir. Erik Dirkx

December 11, 2007

VRIJE UNIVERSITEIT BRUSSEL
FACULTY OF ENGINEERING
Department of Electronics and Informatics (ETRO)

Learning Causal Models of Multivariate Systems

and the Value of it for the Performance Modeling of Computer Programs

Thesis submitted in fulfilment of the requirements for the award of the degree of Doctor in de ingenieurswetenschappen (Doctor in Engineering)
by

Jan Lemeire

Supervisor: Prof. dr. ir. Erik Dirkx

Brussels, December 2007

Examining committee

Prof. Jean-Paul Van Bendegem - Vrije Universiteit Brussel - chair
Prof. Rik Pintelon - Vrije Universiteit Brussel - vice-chair
Prof. Kris Steenhaut - Vrije Universiteit Brussel - secretary
Prof. Jacques Tiberghien - Vrije Universiteit Brussel
Prof. Bernard Manderick - Vrije Universiteit Brussel
Prof. Em. Michel Mouchart - Université Catholique de Louvain
Prof. Koen Bertels - TU Delft
Prof. Bertil Folliot - University Pierre et Marie Curie, Paris

Print: Grafikon, Oostkamp

©2007 Uitgeverij VUBPRESS Brussels University Press
VUBPRESS is an imprint of ASP nv (Academic and Scientific Publishers nv)
Ravensteingalerij 28 B-1000 Brussels
Tel. ++32 (0)2 289 26 50
Fax ++32 (0)2 289 26 59
E-mail: info@vubpress.be
www.vubpress.be

ISBN 978 90 5487 457 7
NUR 950 / 980
Legal deposit D/2007/11.161/064

Preface

I RECALL quite well the first time of my life I heard about computers. I must have been around 10 years of age, so that was in the beginning of the 80's. A friend of my neighborhood spoke about computers in mythical terms, he thought a computer was some kind of oracle which could tell things about the world and the future. Things beyond human capacities. I was shocked that a machine could embody such capacities, but at the same time I felt, I hope you believe me, some skepticism about that whole thing. I went to my father to talk about it.

To my great relief, he reassured me that a computer cannot do anything unless learned by humans. If it would be able to predict the future, it was because somebody supplied him with this information. My worldview was saved. As well as my conviction that it is to humans to understand the world and not to some special god-like machines. I recall this anecdote to illustrate that in those times for the great majority of the population - my friend had heard the story from his parents - a computer was still a mysterious machine. A machine whose capacities and limitations were not yet fully understood. In those early years of computer science, the human brain was used as a metaphor for the working of a computer. The metaphor helped to explain a computer. However, if you think this through with today's experience and knowledge, this metaphor doesn't stand a chance. When using computers in everyday practice, we know that computers are not at all like humans. They do not understand what we want. Unless we provide them with the exact sequence of commands. A computer cannot be called intelligent.

The interesting thing about computer automation is that it forces us to make things explicit. Things that are natural to us, such as the notion of causality or intelligence or learning.

On the other hand, nowadays the metaphor is used in the opposite direction; we employ the internal working of the computer to illustrate how the human brain functions. Because, in fact, it is not the computer which is the mythical machine anymore, but the human brain! All major cognitive scientists must admit that we don't understand everything our brain is capable of. Examples are speech recognition (in a room full of noise!), natural language understanding, reasoning with patterns (discussed in the last chapter), ... And here is the point I want to make: it is by making the computer intelligent, in the way humans are, that we will really understand 'intelligence'. Or 'learning from observations'. Or 'true understanding'. I firmly convinced that the brain is a computer, in the sense that all capacities of the brain can be mimicked by a computer[1].

These reflections are the driving forrce of my scientific ambitions. But I have to warn you, my research does not deal exclusively with these questions. It also includes more down-to-earth research on the performance of computer programs, and practical work about the design and implementation of software (the joy of programming!) or theoretical developments. I hope you will enjoy (parts of) my work.

I would like to start with thanking society for giving me the opportunity for doing a PhD. It's a great honor and privilege to get the time and freedom to develop your own ideas: developing yourself scientifically and personally.

I would like to thank and remember Erik for the faith he put in me and for the opportunity he has given me to start an academic career, as well as the appreciation he had for my industrial experience.

Some feel powerless to take the hurdles of life. The ones, who have the capacities of taking them, have the pleasant obligation to enjoy and to help others in lowering the hurdles. Never forget that the attitude towards life greatly determines the height of the hurdles.

I also would like to thank the members of the jury. For studying my work and the interesting private defense.

The rest of my thankword I would like to do in my mother language.

Vooreerst wil ik mijn ouders bedanken! Mijn moeder die al lang weet dat onderzoek mijn ding is. Ze weet als de beste dat je je kinderen moet aanmoedigen in hun creativiteit. Ik mocht afkomen met de gekste ideeën. Mijn vader heeft me het rotsvast en soms koppig geloof in eigen intellect

[1] After all, Turing showed that a Universal Turing Machine can compute each effective function. And mimic any other universal machine

bijgebracht. Hij leerde me te streven naar het echt begrijpen van dingen. Wat meer is dan een hoop formules.
Vera voor het geduldig nalezen van mijn teksten. Ik wil graag hetzelfde doen voor de verwezenlijking van haar boek.
Mijn zus die ideeën omzet in engagement en niet zoals wij in een ivoren toren blijft zitten.
Mijn broer, samen met Frank mijn belangrijkste intellectuele sparring partner. Ik wens hem alle succes en moed toe om enkele van zijn vele ideeën verder te concretiseren.
Mijn vriendin Anke die als de beste weet steun te geven op de moeilijkere momenten. Ik wil voor haar hetzelfde te doen. En het is een hele geruststellende gedachte te weten dat haar nuchtere analyse van moeilijke situaties me voor naderend onheil zal behoeden.
Mijn meter en peter, en de hele familie, hun onvoorwaardelijke steun (Céline!).
De mannen van TELE, de beste collega's ever: Johan (samen hebben we de boel toch maar op poten gezet!), Arnout (je toewijding en loyauteit bleef me verbazen), Frederik (hoe we samen KDE hebben herontdekt!), Joris (waaw, je begrijpt het!), Rob, Walter en David.
De collega's van ETRO, Jacques Tiberghien en zijn vaderlijke steun, Kris Steenhaut en haar moederlijke zorg, Irene en Karin die ons van zoveel beslommeringen ontlasten, de kaarters...
Mijn thesis studenten, hun aandeel in dit werk, John + Andy = EPPA.
Mijn vrienden, zoals Roger aka Sperwer (het denken begon allemaal met politiek), en alle anderen.

Finally, I would like to thank all people who, despite the many erroneous paths I walked throughout my life, believed in me, and by doing so, supported me greatly.

And I still remember that eureka-moment on a sunny day in February. I was sitting in Fien's veranda, overlooking the snowy fields of Merchtem and instantly I got the idea about introducing causal models into the world of performance analysis. It was my former colleague, Sam Maes, who introduced causality to me. Thanks a lot for that Sam!

Hoeilaart, December 6th, 2007
Jan Lemeire

Abstract

The work presented in this thesis - a philosophical, theoretical and practical exploration of causal inference and its benefits for performance modeling - consists of four parts.

First, the theory of graphical causal models is studied in the perspective of the Kolmogorov Minimal Sufficient Statistic (KMSS), according to which inductive inference should be equated to discovering and modeling the regularities of the data. Regularities are properties of data that allow compression. A Bayesian network provides the KMSS of a probability distribution and is faithful to it when based on the minimal factorization and resulting in a random Directed Acyclic Graph (DAG) and random and unrelated Conditional Probability Distributions (CPDs). Faithfulness is violated when a CPD exhibits a strict regularity, such as defining a deterministic relation, or when a regularity among CPDs gives rise to a conditional independence not following from the Markov condition. We demonstrate that the causal interpretation of Bayesian networks, as defined by Pearl's interventions, corresponds to the canonical decomposition of the model into independent CPDs. The concepts of causal model theory, such as d-separation and interventions, rely on this decomposition. However, the assumption that these submodels correspond to the underlying physical mechanisms of the system under study is not always valid, as demonstrated by the many counterexamples that can be found in literature. Fortunately, unfaithfulness or the presence of regularities among the CPDs give indications that the real structure is based on another decomposition.

The second contribution of this work is the definition of augmented causal models that explicitly capture information equivalences. They appear when two sets of variables contain the same information about another variable. Under weak transitivity, faithfulness of the graph is reestablished. Firstly, a generalized version of the *d*-separation criterion, called D_{eq}-separation, has to be used. Secondly, the conditional independencies that are graphically described are limited with the simplicity condition.

Based on this, an extension to the PC learning algorithm is developed which allows for the construction of minimal augmented Bayesian networks from observational data. Among information equivalent edges, the one with the least complexity is chosen to be the direct relation.

Thirdly, the benefits of causal inference to the automation of the performance analysis of computer programs is researched; more specifically, the benefits of the dependency analysis, the Markov property and the CPD decomposition on which causal model theory is based. In combination with other statistical techniques - such as kernel density estimation, regression analysis, outlier detection and probability table compression - experimental performance data, from sequential as well as parallel programs, are analyzed qualitatively. Non-trivial models, providing insight into the role of every performance factor, are learned. Unexpected dependencies and potential explanations for outliers could be detected. Independency assumptions and application characteristics are validated.

Finally, I analyze the feasibility to construct generic models of the performance of programs and systems. I argue that a performance model and the properties determining the performance greatly depend on the regularities of program and system. This thesis is supported by the results for the modeling of the execution time of the delivery of set of messages of a parallel program on a parallel system. The models are constructed by the causal learning algorithms from experimental data retrieved from simulation of the communication on a model of the network. The focus lies on the topology of the network. The runtime of random communication on random topologies gives a simple functional relation between the communication and topology characteristics and the runtime. But when either communication or network topology exhibit patterns, such as a one-to-all broadcast communication on an hypercube network, the models differ qualitatively and quantitatively. Each combination of patterns in communication and network topology may result in specific models relying on specific performance characteristics.

Contents

Abbreviations

AS	Application Signature	*see Section 9.1.1*
CPD	Conditional Probability Distribution	*see Section 3.2*
CPT	Conditional Probability Table	*see Section 7.1.4*
CPI	Cycles Per Instruction	*see Section 2.1*
CS	Communication Scheme	*see Section 9.2*
CSPF	Communication Scheme Performance Factor	*see Section 9.4*
DAG	Directed Acyclic Graph	*see Section 3.2*
EPDA	Experimental Performance Data Analyzer	*see Section 7.1*
EPPA	Experimental Parallel Performance Analysis	*see Section 6.2*
KDE	Kernel Density Estimation	*see Section 2.2.1*
KMSS	Kolmogorov Minimal Sufficient Statistic	*see Section 2.5*
MDL	Minimum Description Length	*see Section 2.4.2*
MPI	Message Passing Interface	*see Section 6.2*
NT	Network Topology	*see Section 9.2*
NTPF	Network Topology Performance Factor	*see Section 9.4*
p	number of processors	*see Section 6.1*
SP	System Profile	*see Section 9.1.1*
TETRAD	Tool for causal analysis	*see Section 3.4*
W	Work size	*see Section 6.1*

A list of figures is given at the end of the text.

Chapter 1

Introduction

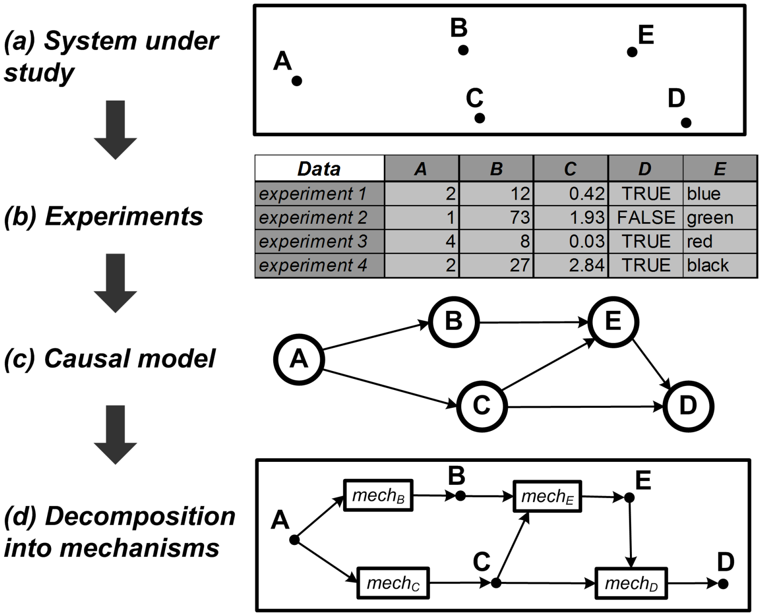
As my grandmother, Grete Drosson, used to say: *"Papier ist geduldig."*

THE primary goal of my work is the injection of causal models and the accompanying learning algorithms into the field of performance analysis. Besides this practical and applied research my work also comprises a theoretical and philosophical study of causal inference.

In this introduction, I will try to give an outline of the text. I start with causal inference and its utility for a performance analysis. I explain how causal models represent qualitative information. The utility of causal models relies on three basic properties. Next, the solutions to overcome the limitations of the current algorithms are discussed. Then, I analyze the validity of causal inference by the theory of Kolmogorov complexity. Finally, the problem of the genericity of models shows that qualitative properties cannot be ignored.

Causal Inference

Causal structure learning algorithms allow the automated learning of causal models from experimental data. A causal model graphically describes the direct causal relations among the variables of interest. It reflects the data-generating mechanisms of a system. The algorithms for causal inference try to understand from observations the underlying structure of a system under study. This learning process is depicted in Fig. 1.1. Variables constitute the observable parts of a system (a). Through experiments, the states of the variables are observed (b). Causal analysis is thus a multivariate statistical analysis. Causal inference returns a graph showing the direct causal relations between the variables (c). I will show that, actually, a causal model corresponds to a decomposition of the system into



Data	A	B	C	D	E
experiment 1	2	12	0.42	TRUE	blue
experiment 2	1	73	1.93	FALSE	green
experiment 3	4	8	0.03	TRUE	red
experiment 4	2	27	2.84	TRUE	black

Figure 1.1: Causal inference tries to reveal the causal structure among the observable variables of a system.

independent mechanisms (d). So Fig. 1.1(c) and Fig. 1.1(d) are different representations of the same information.

Qualitative Information

A performance analysis aims at understanding the execution of a program on a computer system in terms of time and resource utilization. The analysis relies on models providing insight in the variables of program and system responsible for the performance. The contribution to the analysis of causal inference is that it reveals the relational structure of the relevant variables: which and how variables influence the performance.

Causal models represent qualitative information. Qualitative properties play a totally different role than mere quantitative information. They allow for qualitative reasoning, for which precise quantitative information is unnecessary. A representation format for qualitative information was the explicit intention of Judea Pearl when he developed causal model theory [Pearl, 1988, p.79].

Consider the clock frequency of a processor, the number of instructions of a program and the average number of cycles needed for executing an instruction. They are denoted respectively by f_{clock}, $instr$ and CPI (Cycles Per Instruction). The runtime of the application can then be approximated

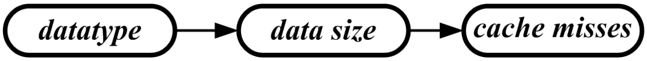


Figure 1.2: Causal model of how data size and datatype influence the cache misses of a program execution.

by the following formula:

$$T_{comp} = instr \times CPI \; / \; f_{clock}. \tag{1.1}$$

This is a quantitative model about performance. The computation time can be calculated from quantitative information about the other three variables.

Qualitative information constitutes another kind of knowledge. Consider the following questions. *Do the cache misses affect the runtime of a program? Which program parameters affect the cache misses? Do the chosen data structures in the program affect the cache misses? Is the knowledge of the size of the data structures sufficient for predicting the cache misses?* These are qualitative questions for which no precise quantitative information is necessary. Answers to these questions can be derived from a causal performance model.

Causal models give away the causal structure and let us understand which variables affect the performance. Every quantity that can be measured or derived from others can be added to the analysis. A causal analysis will reveal the influence of each variable. On top of that, causal inference makes a distinction between direct and indirect influences. Consider a program which implements an algorithm around a main datastructure. Consider that the *datatype* (integer, floating point, double precision, . . .) of the datastructure is a parameter of the program. Causal analysis will reveal that the *datatype* influences the number of *cache misses*. But it will also reveal that this influence happens 'through' variable *data size*, the size of the data structure, as shown in Fig. 1.2. From this model it follows that *datatype* is independent from *cache misses* when conditioned on *data size*. This is written as:

$$datatype \perp\!\!\!\perp cache\,misses \mid data\,size. \tag{1.2}$$

This is called the Markov property: from $A \rightarrow B \rightarrow C$ it follows that $A \perp\!\!\!\perp C \mid B$. Informally speaking, the independence means that once the size of the data is known, the precise datatype doesn't give any additional information about the cache misses. Through this property, a model provides *explanations* for the variables influencing the performance.

Moreover, in the context of learning, the conditional independencies following from the Markov property constitute the key information to differentiate between directly and indirectly related variables. When $A \rightarrow B \rightarrow C$ is the true model, A and C are related, but independency $A \perp\!\!\!\perp C \mid B$ indicates that it is an indirect relation via B.

Finally, causal inference aims at revealing the independent mechanisms of a system. In this context, the mechanisms which determine the overall performance. These basic properties of causal models will proof their utility in a performance analysis.

In Practice

Practical deployment of the current algorithms for causal inference required that three limitations had to be overcome. The learning algorithms work only for data with either linearly related continuous variables or either discrete and categorical variables. But performance data contains a mixture of variable types and contains non-linear relations. Such data cannot be handled by current algorithms. My solution is to employ the general and form-free independency test based on the information-theoretic concept of mutual information. The third limitation is the presence of deterministic relations. They frequently appear in performance data. In the model of Fig. 1.2, the size of the data is completely determined by the datatype. It follows that besides the conditional information of Eq. 1.2, it also follows that $\textit{data size} \perp\!\!\!\perp \textit{cache misses} \mid \textit{datatype}$. This is a violation of the intersection condition, one of the necessary conditions for correct causal inference. The violation is characterized by what I called an *information equivalence*, which means that two variables contain the same information about a third variable. Finding which one of both directly relates to the reference variable cannot be determined by independency information only. The solution I propose is to choose the one having the simplest relation as adjacent to the reference variable. I studied under which assumptions an extension of the definition of causal models lead to consistent models. The learning algorithms could then easily be extended to learn the augmented models.

Meaningful Information

Another part of my work is a philosophical and theoretical study of causal inference based on the principles given by the theory of Kolmogorov complexity. The information content of an individual object is defined by the length of the shortest program that outputs the object, called the Kolmogorov complexity. It offers an objective definition of complexity. The measure is usually expressed in bits. This formal definition of simplicity

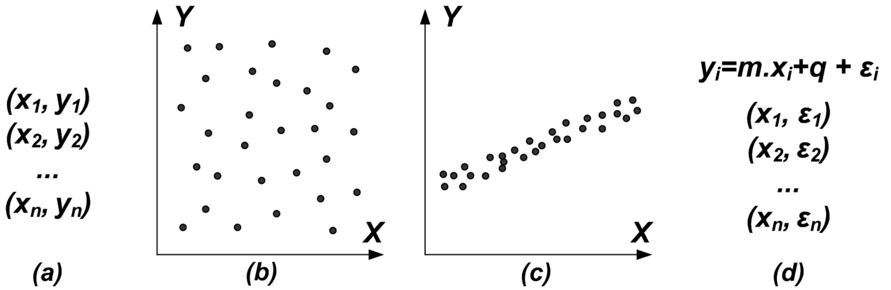


Figure 1.3: Description of two-dimensional data. Literal description of the sample (a). Random (b) and linearly related data (c). The description of the latter can be compressed (d).

enables the use of Occam's razor, "among equivalent models choose the simplest one". Applied to inductive inference, one must select model that describes the data with the minimal number of bits. Learning from observation can thus be viewed as a maximal compression of the observed data.

Consider a sample of n two-dimensional data points. A description is shown in Fig. 1.3(a). If the two dimensions (given by variables X and Y) are unrelated, as in Fig. 1.3(b), the description of the data cannot be compressed. The literal description given by (a) is also the shortest description. If on the other hand, the X and Y dimensions are related, as the linear relation of Fig. 1.3(c), the description of the data can be compressed with the description of the linear function. This is shown in Fig. 1.3(d). The values for Y can be calculated from the X value and an error value which expresses the difference of the actual data point with the linear relation. The errors have smaller values than the values of Y. They can be described shorter. The description (d) is thus more compact than the literal description (a).

Kolmogorov complexity quantifies the information content of an object, but does not make a distinction between random and meaningful information. In the context of learning from observations, we are only interested in the patterns or regularities of the data. We call the properties of data that allow for compression the patterns or *regularities* of the data. One should try to separate the meaningful information - containing the regularities - from the accidental, random information. The Kolmogorov Minimal Sufficient Statistic (KMSS) formally defines the separation. Learning from observation is equated with discovering the patterns in the observed data. One has to look for the KMSS, which is the minimal model that exploits all regularities. For the linearly related data of 1.3(c), the meaningful information is the equation of the straight line, $y_i = m.x_i + q + \varepsilon_i$. The accidental information constitutes the values of X and the error terms of Y.

I will apply the concept of KMSS to causal models. The graph of a causal model describes meaningful information of a probability distribution. In some cases the graph is the only meaningful information, hence it is the KMSS of the distribution. This depends on the presence of regularities not modeled by a causal model. Secondly, I argue that the meaningful information of causal models must be interpreted as a unique decomposition of the system into atomic components, the mechanisms of the system. Both properties, the presence of non-modeled regularities and the correctness of the decomposition, allow us to study the validity of causal inference.

Regularities

Qualitative properties constitute, besides causal inference, another red thread throughout this work. I propose to define qualitative properties as the regularities that lead to the KMSS. To underline the importance of qualitative properties, I devoted the last chapter to a problem in which regularities play a decisive role. A goal of performance modeling is the construction of generic models. Models about the performance of programs running on computer systems ought to be valid for a wide range of combinations of programs and systems. Generic models require the existence of program and system properties that fully characterize the program and system's influence on the performance. I will study a similar problem: the communication time of a set of messages - called the communication scheme - that has to be delivered in a network. I will proof that the patterns of the communication scheme and the network topology influence the performance significantly. This depends on the *match* between the patterns in communication scheme and the patterns in network topology. The existence of generic models is thus limited to certain pattern combinations. Each combination possibly results in a different model. The conclusion is that in a performance analysis the presence of regularities cannot be neglected.

Causal inference and the performance analysis of computer programs are the two central subjects of my dissertation. Causal inference is studied thoroughly; its interpretation and its validity in the first part, its practical deployment and its utility for a performance analysis in the second part.

Scientific Contributions

The scientific contributions of my work consist of:

1. The interpretation of the theory of graphical causal models with the theory of Kolmogorov minimal sufficient statistic (Chapter 4).
2. The extension of causal models and the learning algorithms for the incorporation of information equivalences (Chapter 5).
3. The introduction of the causal learning algorithms in the performance modeling process, the ulitity of causal inference is demonstrated (Chapter 8).
4. The study of the possibility to characterize applications and systems by generic properties. The limitations of generic models is demonstrated for the communication time of the delivery of a set of messages through an interconnection network. It is shown that qualitative and quantitative models become incorrect for specific combinations of regularities in the communication and network topology (Chapter 9).

Outline

The remains of this dissertation are organized in two parts. While the injection of causal models into the world of performance is treated in the second part of this work, the first part is devoted to a closer study of causal inference. Both parts consist of 4 chapters. The first two chapters of each part give introductions to the topics. The third and fourth chapters contain the scientific contributions of my research.

Chapter 2: ***Principles of Inductive Inference.***

Approaches for inductive inference try to address the problem of how one should decide among competing explanations of data given limited observations. The principles of maximal entropy and Occam's Razor are introduced, as well as Kolmogorov Complexity and the Kolmogorov Minimal Sufficient Statistic (KMSS). This last concept formally separates meaningful information - containing the regularities of the data - from random, accidental data.

Chapter 3: ***Graphical Causal Models.***

This chapter gives a comprehensible introduction into graphical causal models and the accompanying learning algorithms. The discussion about what causality really is, is briefly touched upon. The focus lies on the probabilistic account of causality, as developed by Pearl. He established the link between causality and the theory of Bayesian networks. It is based on the effect of a causal structure on the conditional independencies the structure it entails. The learning of causal models from observation is based on the independencies following from Markov chains and v-structures.

Chapter 4: ***The Meaningful Information of Distributions.***

In this chapter causal inference is interpreted as searching for the KMSS of an observed probability distribution. Proof is given that the KMSS is described by the Directed Acyclic Graph (DAG) of a Bayesian network if the graph succeeds in explaining all conditional independencies. The independencies form the regularities of the data. Next, I argue that the causal component corresponds to a decomposition of the system in independent submodels, which are assumed to correspond to the underlying mechanisms of the system under study. The validity of causal models can therefore be studied by the presence of other regularities and the validity of the decomposition.

Chapter 5: ***Information Equivalence.***

In this chapter I present a thorough discussion of information equivalences and the impact on causal models and the learning algorithms. Augmented Bayesian networks are proposed which explicitly capture information equivalences. The complexity of relations is taken as a criterion to select the direct relation among information equivalent ones. Based on this criterion, an extension of the constraint-based PC algorithm is developed which can successfully infer augmented models from data containing deterministic relations.

Chapter 6: ***Performance Analysis of Parallel Processing.***

Part two of this dissertation begins with a chapter about the metrics employed in the performance analysis of parallel applications. It is based on the lost cycle approach of Crovella et al. and the impact of overheads on the speedup. An overview of our performance analysis tool, EPPA, is given. It is used for tracing experiments with parallel applications and storing the data about their performance.

Chapter 7: ***Qualitative Multivariate Analysis.***

This chapter explains how the causal inference algorithms fit into a multivariate analysis. The EPDA tool offers a repository for storing experimental data and a set of statistical techniques for analyzing the data. The different techniques are explained and their role in the modeling process. For applying causal inference on performance data, a form-free independence test based on a kernel density estimation has to be employed. Details about the implementation and calibration are given.

Chapter 8: ***Causal Inference for Performance Analysis.***

This chapter starts with a thorough analysis of the potential benefits of causal models and causal inference to the analysis of the performance of computer programs. Causal performance models are defined and the results of the analysis of experiments with sequential and parallel programs are presented. They demonstrate the utility of integrating causal inference in the performance modeling process.

Chapter 9: ***The Genericity of Performance Models.***

The last chapter addresses the question as to whether generic characteristics of applications and systems exist so that the performance can be accurately estimated for any combination of application and system. A possible explanation for the absence of a generic performance models is revealed by the analysis of the execution time of a communication scheme on a network topology. It is shown that random communication on a random topology gives a statistical value for the runtime. However, when either communication or topology exhibit regularities the delivery can give rise to specific behavior which differs a lot from the random-random combination. A considerable part of the combinations result in very specific qualitative and quantitative models.

Publications

Chapter 2: ***Principles of Inductive Inference.***

- Jan Lemeire, *Complexity-Preserving Functions*, DIMACS Workshop on Complexity and Inference, Rutgers University, Piscataway, NJ, June 2 - 5, 2003.

Chapter 4: ***The Meaningful Information of Distributions.***

- Jan Lemeire and Erik Dirkx, *Causal Models as Minimal Descriptions of Multivariate Systems*, Presentation presented at Workshop Causality and probability in the sciences, organized by Federica Russo and Jon Williamson, Canterbury, UK, June 2006.

Chapter 6: ***Performance Analysis of Parallel Processing.***

- Jan Lemeire, Andy Crijns, John Crijns and Erik Dirkx, *A Refinement Strategy for a User-Oriented Performance Analysis.* In Proc. of the 11th EuroPVM/MPI Conference, Budapest, Hungary, Sept. 19-22, 2004.

- Johan Parent, Katja Verbeeck, Jan Lemeire, Ann Nowé, Kris Steenhaut, Erik Dirkx, *Adaptive Load Balancing of Parallel Applications with Multi-Agent Reinforcement Learning on Heterogeneous Systems*, in Special Issue on Distributed Computations and Applications, Scientific Programming, ed. C. H. Lai, Vol 12, No 2, IOS Press, 2004.

- Jan Lemeire, Erik Dirkx, *Performance Factors in Parallel Discrete Event Simulation*, In Proceedings of the 15th European Simulation Multiconference, Prague, Czech Republic, June 6-9, pp. 623-627, SCS, 2001.

Chapter 7: ***Qualitative Multivariate Analysis.***

- Jan Lemeire and Erik Dirkx, *Causal Models for Parallel Performance Analysis*, Fourth PA3CT-Symposium, Edegem, Belgium, September 2004.

Chapter 8: ***Causal Inference for Performance Analysis.***

- Jan Lemeire, Erik Dirkx and Frederik Verbist, *Causal Analysis for Performance Modeling of Computer Programs.* Scientific Programming, Vol. 15, No 3, pp. 121-136, IOS Press, 2007.

Chapter 9: ***The Genericity of Performance Models.***

- Jan Lemeire, Erik Dirkx, Walter Colitti, *Modeling of the Performance of Communication Schemes on Network Topologies*, Submitted for Parallel Processing Letters.

- Jan Lemeire and Erik Dirkx, *Lookahead Accumulation in Conservative Parallel Discrete Event Simulation*, In Proc. of the High Performance Computing and Simulation (HPCS) Conference, Magdeburg, Germany, June 13 - 16, 2004.

- Jan Lemeire, Erik Dirkx and Bart Smets, *Exploiting Symmetrical Properties for Partitioning of Models in Parallel Discrete Event Simulation.* In Proc. of the 18^{th} Workshop on Parallel and Distributed Simulation (PADS 2004), May 16-19, pp.189-194, IEEE, California, 2004.

Part I

Causal Inference

WHAT is there to study in a *random* world? A world in which each thing we observe doesn't depend on anything; in which the states of all objects are mutually independent and independent of previous states. In such a world, the only thing a scientist can do is to log the state of each object at any moment in time. This would result in a substantial book containing lots of data; but no laws or equations could be derived from it. The historical information would not help us to understand the present state of the world or assist us in predicting future states.

A random world is boring. The interesting part of the world consists of the patterns or *regularities* that are found in observations; from which we can make laws describing how objects depend on each other and how previous states generate future states. This process is called *inductive inference.* Inductive inference is the process of inferring general laws from observations of particular instances. The focus of my research lies on *causal inference.* On how causal models can be learned from experimental data and how they may be beneficial to the performance analysis of computers.

Regularities are defined by their ability to compress data. Hence to serve the quest for minimality as imposed by Occam's Razor. One not only has to look for the minimal model, but for a model solely capturing the regularities. The purpose is to extract from the experimental data the meaningful regularities, to separate them from the accidental random information in the data. This is formalized by the concept of Kolmogorov Minimal Sufficient Statistic (KMSS). The KMSS is the minimal model that captures all regularities of the data. I will use these insights for studying the validity and limitations of causal inference.

Causality occupies a central position in human cognition. It plays an essential role in human decision-making by providing a basis for choosing the action which is most likely to lead to the desired result. Since smoking is one of the main causes of lung cancer, one should stop smoking for a longer healthy life. But stopping would not help if smoking wasn't a cause of cancer. In case the observed correlation between smoking and cancer would be due to a common cause, such as a gene that makes you addicted to smoking and increases the chance of getting lung cancer. Knowledge of the correlations between variables is not enough in decision-making. Deciding on adequate actions must rely on the understanding of causal relations.

Judea Pearl developed a probabilistic account to causality, which culminated in the theory of *graphical causal models.* The theory offers an explanation for the difference between the well-known statistical concepts of correlation or functional relations which are symmetrical, and the asymmetries implied by causality. Causal models do not only describe the functional or probabilistic relations among the variables, but also the mechanisms of the system under study: by which variables the state of each variable is 'produced'. I will defend the thesis that the causal component of graphical causal models is actually a form of *reductionism*.

Causal inference is the learning of causal models from experimental data. The key is the information about the *conditional independencies* that follow from a causal structure. Such conditional independencies constitute the *regularities* which reveal the structure of the system and make causal inference possible.

The conditional independencies following from the causal structure of a system constitute only one type of regularities. Other regularities in the system might occur. Such regularities can produce additional independencies, which may disrupt the correct course of the learning algorithms. *Deterministic relations* are such a kind of regularity. The state of a deterministically related variable is completely determined by a set of variables. Deterministic relations imply non-Markovian independencies. Data about the performance of computer programs contain deterministic variables. Current causal inference algorithms fail when the data contains deterministic variables. To overcome this limitation, I had to develop extensions to causal models and the learning algorithms.

Chapter 2

Principles of Inductive Inference

THE main topic of my research is causal inference, which is a kind of inductive inference. Inductive inference is concerned with the question *"How does one decide among competing explanations of data given limited observations"*. This chapter will review and discuss some of the basic principles that guide learning from observation. The goal is to introduce kernel density estimation (Sec. 2.2.1) and the Kolmogorov Minimal Sufficient Statistic (KMSS) (Sec. 2.5). The former will be used for the independence test in causal inference, the latter to evaluate causal inference.

Firstly, Shannon's information theory is briefly presented. Its concepts will be encountered several times during the rest of this work. Then, the maximum entropy principle of traditional statistics is explored together with kernel density estimation. Next the almost 'holy' scientific principle of Occam's Razor is discussed. To take off with my favorite topic, Kolmogorov complexity, the implications of which are not yet fully understood and can, in my opinion, not be over-estimated. The section introduces the theory of the length of the shortest program and the related Minimum Description Length approach. Even more relevant are the quite new additions about regularities and meaningful information, formally based on the Kolmogorov Minimal Sufficient Statistic.

2.1 Classical Information Theory

Classical information theory is based on a set of fundamental concepts, omnipresent in computer science, relying on a robust and sound mathematical treatment. A more thorough discussion can be found in Cover and

Thomas [1991], the 'bible' of information theory.

Entropy quantifies the amount of *uncertainty* of a stochastic variable. It is a concept of thermodynamics for measuring disorder. It corresponds to the *information content* of a variable. For a discrete random variable X with domain D_X and probability mass function $P(x)$, its entropy is calculated by

$$H(X) = -\sum_{x \in D_X} P(x) \log P(x). \tag{2.1}$$

Note that variables are denoted by upper case letters, values or outcomes of variables by lower case letters. All logs here and in the remainder of text are based 2. By the choice of base 2, the unit for information is *bits.*

Consider that the domain of x (the set of all possible values of x) contains 8 objects. If the probability of an outcome is equal for all, namely 1/8, the uncertainty is maximal, namely 3 bits. If on the other hand, some outcomes are more probable than others, the uncertainty of x decreases. It can even become 0 if the probability is one for one outcome and zero for all others. In that case we are completely certain of the value of x. Entropy is the inverse of knowledge.

To fully understand the impact of the concept, consider the application of designing optimal codes. Entropy represents the average number of bits that are necessary to communicate objects produced by a random source. The definition of entropy must be interpreted as the expectation of $-\log P(X)$; it is the weighted average of $-\log P(x_i)$ over all values of x_i.

Example 2.1 (Optimal Communication Codes).

> Shannon discovered the notion of entropy while studying the capacity of communication channels. Consider a communication line that allows faultless transmission of bits and 8 kind of messages ('a', 'b', ..., 'h') that can be transmitted. We can encode each message with 3 bits ('000', '001', ..., '111'). But if the probability of sending a message is not equal for all messages, we can devise a more optimal prefix code, by attributing shorter codes for messages with high probability and longer codes for messages with low probability. The following table gives an example of a non-uniform probability distribution together with the optimal code calculation.

x	$P(x)$	$-\log P(x)$	old code	new code(x)	$-2^{L(code)}$
a	0.420	1.25	000	0	0.5
b	0.141	2.83	001	100	0.125
c	0.142	2.82	010	101	0.125
d	0.047	4.41	011	11100	0.031
e	0.141	2.83	110	110	0.125
f	0.047	4.41	101	11101	0.031
g	0.046	4.44	110	11110	0.031
h	0.016	5.94	111	11111	0.031

The optimal prefix code utilizes $-\log P(x)$ bits for each message. We need a prefix code, by which no code word is a *prefix* of any other code word, so that we do not need commas for separating the code words. The entropy gives the average code size ($\sum_{x \in A} P(x).L(code(x))$) if the optimal code is used, $L(.)$ is the binary length function. Here, each message needs on average 2.439 bits. The code employed in the example tries to approximate $-\log P(x)$ by integers, and results in an actual average code size of 2.473 bits. This encoding is based on the following property of binary prefix codes, called the Kraft inequality:

$$\sum_{x \in D_X} -2^{L(code(x))} \leq 1 \tag{2.2}$$

This is illustrated by the binary tree in Fig. 2.1. Note the correspondence of the equation with $\sum_{x \in D_X} P(x)$ if the equation becomes an equality by using all leaves of the binary tree. Short code lengths can be formally related to high probabilities.

Two stochastic variables X and Y are **conditionally independent** by conditioning on Z if $P(Y \mid X, Z) = P(Y \mid Z)$. Independency can be interpreted in information-theoretic terms. Two variables contain information about each other if by knowing one variable, the uncertainty - or entropy - of the other is reduced. The information one variable conveys about another can be quantified by the reduction in uncertainty, called **mutual information**, written as $I(X;Y) = H(X) - H(X \mid Y)$. Note that this concept is symmetric: $I(Y;X) = H(Y) - H(Y \mid X) = H(X) + H(Y) - H(X,Y)$, since $H(X,Y) = H(X) + H(Y \mid X)$. The mutual information can also be written as

$$I(X;Y) = \sum_{x \in A} \sum_{y \in B} P(x,y) \log \frac{P(x,y)}{P(x)P(y)} \tag{2.3}$$

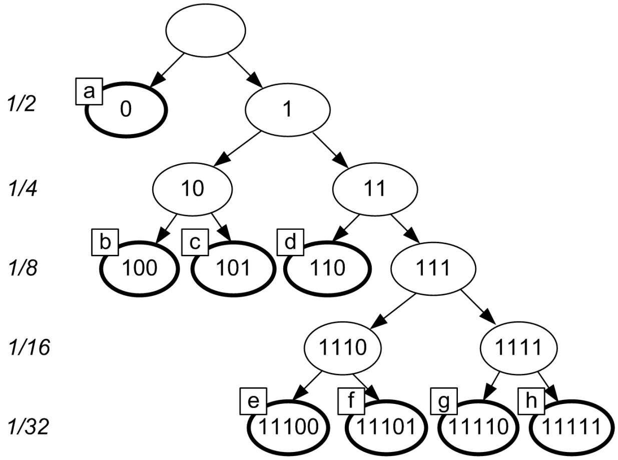


Figure 2.1: Prefix code construction shown by binary trees.

This equation shows clearly that if variables are independent ($P(X, Y) = P(X)P(Y)$) the mutual information gets 0. The equation measures the difference between $P(X, Y)$ and $P(X)P(Y)$ and is always positive.

Definition 2.2. *Random variables X, Y, Z are said to form a Markov chain in that order, denoted by $X - Y - Z$, if the joint probability mass function can be written as*

$$P(X, Y, Z) = P(X)P(Y \mid X)P(Z \mid Y) \tag{2.4}$$

Important consequences are as follows [Cover and Thomas, 1991]:

- X, Y, Z form a Markov Chain if and only if X and Z are conditionally independent given Y. This is called the **Markov property**.

- Information cannot increase along a Markov chain: $I(X;Y) > I(X;Z)$. This is called the *Data Processing Inequality.*

- In literature, a Markov chain is denoted with arrows: $X \Rightarrow Y \Rightarrow Z$. But Markov chain $X \Rightarrow Y \Rightarrow Z$ implies that $Z \Rightarrow Y \Rightarrow X$ is also a Markov chain. The orientation of the relations is not necessary. We will therefore ommit the arrows and write in a Markov chain as $X - Y - Z$. This in order to avoid confusion with the oriented edges of causal models, in which the orientations denote asymmetric causal relations.

2.2 Maximum Entropy Principle

Informally stated, the principle suggests that we should adopt a maximal cautious attitude while learning from observations, or:

The Maximum Entropy Principle

One must opt for the statistical model that explains the observations but keeps maximal uncertainty over all phenomena.

The principle was introduced by Jaynes in 1957 [Jaynes, 1957] but had its roots in the work of Boltzmann and Gibbs on statistical mechanics [Jaynes, 2003]. The **principle of maximum entropy** provides a means to arrive at a probabilistic model in the presence of incomplete and partially unreliable information. Our best guess for P is realized by adopting the P that maximizes Shannon's entropy while remaining consistent with the given information. Given certain constraints on a probability distribution, it constructs a single distribution over all outcomes.

Example 2.3 (Complete ignorance).

The distribution over the domain of a variable without any constraints (except that the sum of all probabilities must be 1) with maximum entropy is given by $P(x_i) = 1/k$ for all $x_i \in D_X$ with $k = |D_X|$. The entropy of the uniform distribution is

$$H(P) = \sum_{i=1}^{k} -P(x_i) \log P(x_i) = -k.\frac{1}{k} \log \frac{1}{k} = \log k \qquad (2.5)$$

Data drawn from any other distribution will, with high probability, be more regular than data drawn according to the uniform distribution. The data points will appear more in certain regions. As the kurtosis (the 'peakedness') of a distribution increases the probabilities drift away from $1/k$ and the entropy decreases. In the limit, $P(x_i)$ becomes 1 for 1 value of X and zero for all others. The entropy then becomes 0.

Example 2.4 (Normal distribution).

The importance and success of the normal (or Gaussian) distribution in statistics is based on the property that it gives the distribution with maximum entropy under the constraints of a given expectancy ($E[X] = \mu$) and standard deviation ($VAR[X] = \sigma^2$). The normal distribution will have mean μ and variance σ^2. In statistical inference, the mean and deviation are estimated from the sample of observed data:

$$\mu \cong \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.6}$$

$$\sigma \cong \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2 \tag{2.7}$$

Example 2.5 (Independence).

Consider two biased coins. Let p and q be the probabilities of having 'heads' with respectively the first and the second coin. The joint distribution with maximal entropy for the combination of the two coins is given by the product of the individual probabilities:

first coin second coin	heads	tails
heads	$p.q$	$(1-p).q$
tails	$p.(1-q)$	$(1-p).(1-q)$

This distribution corresponds to the model of two stochastically *independent* coins, in which the outcome of one does not affect that of the other. Without further information, the principle says we have to assume an independency. Then, the entropy for the combination of two coins equals the sum of the entropies of the individual coins.

The maximum entropy principle could be applied fruitfully in these examples because the information is in terms of statistical functions and averages. But in order to apply the principle to a broader class of information, it should be formally defined what is meant by 'conform the given information'.

2.2.1 Kernel Density Estimation

Problem definition:
Given a sample of data points, estimate the most likely probability distribution that generated the observations.

If the distribution is over discrete variables, it can be estimated by simply counting the number of occurrences of each state and dividing them by the number of data points.

For continuous variables, however, the application of this method would result in a histogram. But this histogram would depend on the chosen intervals. Moreover, the number of intervals is limited, since every interval should contain multiple data points for a correct estimation of its probability. **Kernel density estimation** makes it possible to estimate the distribution from limited sample sizes. The kernel estimate is constructed by centering a scaled kernel at each observation. The value of the estimate at point x is the average of the n kernel values at that point. This corresponds to a convolution of the data points with a well-chosen kernel [Wand and Jones, 1995]:

$$p(x) = \frac{1}{nb} \sum_{i=1}^{n} K(\frac{x - x_i)}{b}) \tag{2.8}$$

with n the sample size and $K(.)$ the multivariate kernel function, which is symmetric and satisfies $\int K(x)dx = 1$. The factor b is the smoothing **bandwidth** and determines the width of the kernel. The estimation of multivariate distributions is based on multidimensional Gaussians and is constructed likewise. Intuitively, a 'probability mass' of size $1/n$ associated with each data point is spread about its neighborhood. Fig. 2.2 shows an example over 1-dimensional data, Fig. 2.3 one over 2-dimensional data. Consult `http://parallel.vub.ac.be` for interactive applets that allow you to experiment with kernel density estimation.

Theoretic analysis and simulation have shown that the choice of the kernel is not crucial. The most-chosen kernel is the Gaussian kernel function. Recall that this is the function with maximum entropy. The bandwidth is the determining factor for good estimates. A too large bandwidth would flatten the distribution; a too small would generate a peak for every data point. A good trade-off smoothens the distribution nicely. Intuitively one can see that the bandwidth for constructing the estimate of Fig. 2.2 gives a good result.

The choice of the kernel bandwidth should be chosen such that the estimate has *maximum entropy* and that nevertheless the sample is still

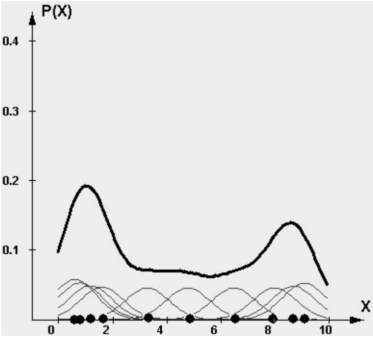


Figure 2.2: Data points (the 10 points on the X-axis) and kernel estimate (thick black line) constructed by Gaussians at each point (thin black lines). The chosen bandwidth is 0.8.

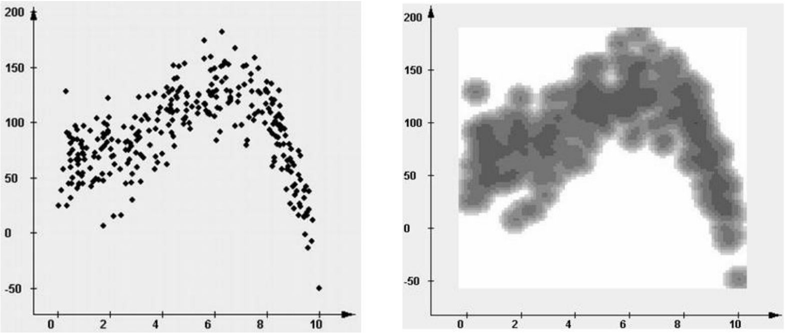


Figure 2.3: Example of a Kernel Density Estimation of Two-Dimensional Data.

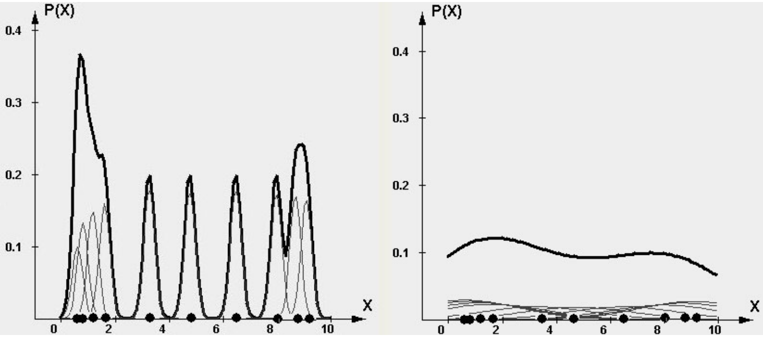


Figure 2.4: Undersmoothed and oversmoothed kernel estimation with bandwidths of respectively 0.25 and 1.5.

Figure 2.5: William of Ockham (1288-1349)

typical for the distribution. A sample is typical for a distribution if it is likely to occur.

Too small bandwidths can give a rough distribution for which the sample is typical, but with low entropy. The estimate is said to be 'undersmoothed'. Increasing the bandwidth maximizes the entropy, but for too large bandwidths the sample is not typical any more. The estimate is said to be 'oversmoothed'. Both cases are depicted in Fig. 2.4.

For applying both principles, the entropy can be calculated, but how measure typicalness? The following section will result in a formal definition of typicalness, but does not give a practical method that can be applied here. The solutions I opted for and the exact calibration of the estimation will be given in Section 7.3.3.

2.3 Occam's Razor

Occam's Razor

"Among the theories that are consistent with the observed phenomena, one should select the simplest theory."

William of Ockham, 1320

Or as it was formulated by Bertrand Russell:

> "It is vain to do with more what can be done with fewer."

Or as Isaac Newton (1642-1727) formulated it in his famous work *'Principia'* [Newton, 1687]:

> "We are to admit no more causes of natural things than such as are both true and sufficient to explain the appearances. To this purpose the philosophers say that Nature does nothing in vain, and more is in vain when less will serve; for Nature is pleased with simplicity, and affects not the pomp of superfluous causes".

The following paradox illustrates the kind of problems that *Occam's Razor* tries to solve.

Example 2.6 (The Grue Emerald Paradox.).

> The paradox deals with the logical problem of inferring general laws by specific observations. For example, as Hume phrased the Problem of Induction in the 18th century (before the discovery of Cygnus stratus in Australia) "All swans we have seen are white, and therefore all swans are white". Nelson Goodman presented the paradox in the article 'The New Problem of Induction' (1966) as follows.
>
> "We seem able to justifiably infer that all emeralds are green from the fact that all previously observed emeralds have been green. This is, of course, not a valid deduction, but it would be a radically skeptical position to deny that this inference confers any likelihood at all on its conclusion. What if we try to follow the same inferential procedure with a different predicate in the place of 'green'? Take Hempel's invented predicate 'grue'. An

> object is grue if 'it is either observed before January 1st 2008, and found to be green, or observed after that date, and found to be blue'. From the fact that all previously observed emeralds have been observed to be grue, which they have, we ought to be able to infer that all emeralds are grue. But this means that all emeralds discovered from next year onwards are likely to be blue. This is patently false, but the challenge is to say how we can uphold the first argument and fault the second, given that they appear to exemplify the same logical machinery."
>
> Why can we rule out the existence of grue emeralds?
>
> Or consider the following claim: "There are diamond pyramids on Pluto". One can claim that, unless we have closely observed Pluto, this statement can not be rejected. The pyramids could be there or could not be there, but until we haven't been there, there is nothing sensible we can say about it.

Occam's Razor gives us an elegant way out of these paradoxes. The simplest world model based on our current knowledge, excludes the existence of grue emeralds or diamond pyramids. To explain these facts our models should be altered enormously. The color of emeralds or the size and shape of diamonds are not mere properties of objects that can simply be altered. The properties come from their physical constitution. Our understanding of the physics of crystals, our insight in color generation and the growth of crystals would fall short in explaining the change of color at a certain time, or pyramid-shaped carbon crystals.

Note that the application of *Occam's Razor* on such problems coincides with our intuition. Dealing with the above paradoxes, looks completely unworldly, irrelevant for outsiders. Thus, consciously or just intuitively, scientists seek simpler and simpler theories that are able to explain our current knowledge.

The more practical problems for which *Occam's Razor* provides an answer, can best be explained by the following example.

Example 2.7 (Curve Fitting).

> Consider the problem of finding the best curve that fits a data sample. Fig. 2.6 shows a sample of 15 data points and three types of functions that are fit on the data. The straight line is the best linear function that minimizes the error, the dashed

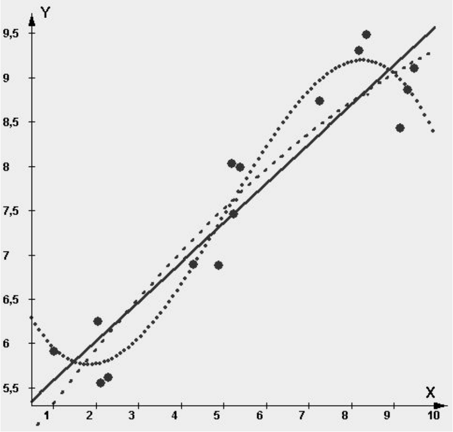


Figure 2.6: Curve fitting on 15 data points with 1^{st}, 2^{nd} and 3^{th} degree polynomials

line is a polynomial of second order and the dotted line is a third-order polynomial. The data is generated from function $Y = 5 + 0.45X + U$ with randomly chosen X-values between 0 and 10, and with U random disturbances between -1 and +1. The following table gives the fitted functions and the sum of squared errors, defined as $\sum_{i=1}^{n}(y_i - f(x_i))^2$ ('o. pol.' stands for 'order polynomial'):

curve type	function	error
linear	$Y = 5.136 + 0.444X$	193
2^{nd} o. pol.	$Y = 4.677 + 0.676X - 0.021X^2$	192
3^{th} o. pol.	$Y = 6.768 - 1.18X + 0.392X^2 - 0.026X^3$	184

Complex functions clearly results in better fits, but lead to *overfitting*. The complexity of the functions should be included into the analysis in order to find a good trade-off between *model complexity* and *goodness-of-fit*.

The main motivations for *Occam's Razor* are:

1. There exist fewer simple models that fit the observed data than complex models. The probability of having a simple, but faulty, hypothesis that coincides with the data, is smaller than having a faulty

complex model. It is quite easy to construct complex models that explain all individual data points, but they follow the individual data points more than the trends in the data. A simpler model is more likely to describe reality.

2. Simple models are more reliable. Even if the true data-generating machinery is very complex, simple models give good predictors [Grünwald et al., 2005]. Consider curves of different complexities fitted on data as shown in Fig. 2.6. Even if the polynomial of third degree would be the true model, the linear curve correctly reflects the main trend of the function. Whereas the third-degree polynomial reflects various trends by its capricious form that are not representative for the true linear function.

3. Simpler theories are more constraining and thus more falsifiable; they provide the scientist with less opportunities to overfit the data "hindsightedly" and therefore command greater credibility if a fit is found [Popper, 1959].

However, the preference for simpler models cannot be theoretically proved to always provide correct models. In practice, the principle will not always lead to correct models. It can not be proved that a simpler model is correct, just as it can not be proved for *any* model learned by induction to be correct. As Grünwald claims in his interesting and readable introduction to MDL [Grünwald et al., 2005, p. 16] or his PhD [Grünwald, 1998], *Occam's Razor* has to be regarded as a *strategy* for scientific research:

> Thus, MDL (and the corresponding form of Occam's razor) is a *strategy* for inferring models from data ("choose simple models at small sample size"), not a statement about how the world works ("simple models are more likely to be true") - indeed, a strategy cannot be true or false; it is "clever" or "stupid".

The aim of modeling is not to find the 'true' model, but a good model.

On the other hand, there is also criticism for *Occam's Razor*. As G. Webb puts it: "What good are simple models of a complex world?". And some researchers report on settings in which the simpler models do not provide good answers. But in my opinion, the absence of a decisive, convincing example, undermine the criticism of these opponents.

2.4 Kolmogorov Complexity

Algorithmic Information Theory, better known by its fundamental concept *Kolmogorov Complexity*, deals with the quantization of information in individual objects. Its concepts are paralleled by those of classical information theory [Grünwald and Vitányi, 2003], where the latter deals with relations between probabilistic ensembles and not individual objects. The mathematical theory of Kolmogorov complexity contains deep and sophisticated mathematics, discussed at length in the fundamental book of Li and Vitányi [1997]. Yet the basic ideas are pretty intuitive, one needs to know only a small amount of these mathematics to apply the notions fruitfully.

2.4.1 Definition

Karl Popper challenged the utility of Occam's razor as long as an objective criterion to measure simplicity is missing. Kolmogorov Complexity provides an answer to it. It was introduced independently by Solomonoff in 1964 [Solomonoff, 1964], Kolmogorov in 1965 [Kolmogorov, 1965] and Chaitin in 1969 [Chaitin, 1969]. Solomonoff used it in his work on deductive inference, Kolmogorov aimed initially to give a satisfactory definition for the problematic notion of random sequence in probability theory, and Chaitin was studying just the program-size complexity of Turing machines.

Definition 2.8. *The **Kolmogorov Complexity** of a binary string x is defined to be the length in bits of the shortest computer program that prints the sequence and then halts [Li and Vitányi, 1997]:*

$$K(x) = \min_{p:\mathcal{U}(p)=x} l(p) \tag{2.9}$$

with $\mathcal{U}$ a universal computer, and $l(.)$ the length of a binary string.

Although the length of the shortest program depends on the chosen computer language, the **Invariance Theorem** guarantees that the description of the same strings by two universal languages differs by no more than by a fixed constant c [Solomonoff, 1964]. **Church's thesis** states that all (sufficiently complex) computational models can compute the same family of effectively computable functions [Church, 1936]. For such functions there is a program that will lead in a finite number of mechanically specified computational steps to the desired computational result. A generic computer, able of computing all effectively computable functions, is called a **Universal Turing Machine**. A Universal Turing Machine can imitate the behavior of any other (Universal) Turing machine. $K(x)$ will therefore only differ by maximal the length of the program that imitates the other Turing Machine, a constant that is independent of x.

Kolmogorov Complexity gives an absolute measure of the information content of an *individual* finite object, whereas Shannon's entropy measures the information content of an object with respect to a set of objects. The entropy of a random variable measures the information of the occurrence of an outcome. It provides the average number of bits of the minimal code necessary to distinguish a specific outcome among the set of possible outcomes. However, it does not say anything about the number of bits needed to communicate any individual message of the set. Consider all binary strings of length 1000000. If the outcome of each string is equally probable, by Shannon's measure, we require 1000000 bits on the average to communicate a string. However, the string consisting of 1000000 1's can be encoded with fewer bits by expressing 1000000 and adding the repeated pattern '1'.

Example 2.9 (bit strings).

Take the following 3 sequences of 1000 bits:

- 01111000011001100111 . . . 00001111100100011101
- 00010001000100010001 . . . 00010001000100010001
- 00100010000110100000 . . . 00001001010000000000

The first string is random, the second repeats "0001", the third is random with the probability of having a '0' 4 times that of having a '1'. $K(x)$ is maximal for the random string, namely 1000 bits. The shortest program literally encodes the string. The second string is described by program `REPEAT 250 TIMES "0001"` and needs about 28 bits (log(250)=8 + 4 + 16 bits for `REPEAT` statement)[1].

The third string can also be compressed. We could for example use the code developed in example 2.1 of the introduction. The probabilities correspond to those for having a substring of 3 bits with probability $\frac{4}{5}$ for having a '0'. The entropy was found to be 2.473 bits. Each message of 3 bits thus needs on average 2.473 bits. The use of this code then results in a description with 2.473*1000/3=823 + 54 (the number of bits in the table,

[1] For not overloading the discussion I omit details about prefix codes.

which is the description of the code) = 877 bits. Note that this is not the shortest program.

Unfortunately, $K(x)$ is *intractable* - there is no algorithm that is able to compute the shortest program for x nor can it be proved that a certain program is the shortest for x. This follows from the *halting problem*: how can we know which programs will eventually terminate and which go on forever without coming to a definite conclusion. This can on its turn can be related to Gödel's famous incompleteness theorem of logic.

2.4.2 Minimum Description Length

Solomonoff's paper, called "A Theory of Inductive Inference" [Solomonoff, 1964], contained the idea that the ultimate model for a sequence of data may be identified with the shortest program that prints the data. His ideas were later extended by several authors, leading to an 'idealized' version of the Minimum Description Length methodology (Solomonoff 1978) [Li and Vitányi, 1997] [Gács et al., 2001].

Minimum Description Length (MDL) [Rissanen, 1978] aims at providing a generic solution to the model selection problem: one has to decide among competing theories of data given limited observations. Because of the intractability of Kolmogorov complexity, the idea is to scale things down in a way that it becomes computable. We should use an arbitrary class of models $\mathcal{M}$ and do the encoding of the data with the help of the model class. This results in a *two-part code*. The first part describing the model and the second part describing the data with the help of the model:

$$description(data) = description(model) \quad + \quad description(data \mid model) \tag{2.10}$$

By minimizing the total length of both descriptions for models of $\mathcal{M}$, this approach inherently protects against overfitting and trades-off goodness-of-fit on the observed data with complexity of the model. The former is quantified by the first term, the latter by the second term.

Minimizing solely the training error (the second term of Eq. 2.10) leads to overfitting. But it also leads to a variance of the hypothesis when trained with finite data sets sampled randomly from the true distribution. When taking different samples of a true linear relation with Gaussian errors, each sample will lead to a different higher-degree polynomial. Since the polynomial is not fitting the trend, but is fitting the random variations which are different from sample to sample. To overcome this, the complexity of the hypothesis should be taken into account. We have to add a penalty term which penalizes model complexity, the first term of Eq. 2.10. The under-overfitting tradeoff corresponds to the *bias-variance tradeoff* which arises in classical estimation theory. Typically, bias comes from not having good hypotheses in the considered class. This generates a systematic error. If the bias is high, the model is underfitting the data. Variance results from the hypothesis class containing too many hypotheses. If the variance is high, the model is overfitting the data. Hence, we are faced with a trade-off: a more expressive class of hypotheses generates a higher variance, a less expressive class generates a higher bias.

Formally written [Grünwald, 1998]:

$$H_{mdl} = arg\ min_{H \in \mathcal{M}} \{L(H) + L(D \mid H)\} \tag{2.11}$$

and $L(.)$ the description length. MDL says to pick out the hypothesis H_{mdl} from model class $\mathcal{M}$ where H_{mdl} is the hypothesis which minimizes the sum of the description length of hypothesis H and of the data D encoded with the help of H. A minimal two-part code contains no redundancy, every bit is information.

Learning can be viewed as data compression: it tells us that, for a given set of hypotheses $\mathcal{H}$ and data set D, we should try to find the hypothesis or combination of hypotheses in $\mathcal{H}$ that compresses D most.

For applying MDL, there is no need to assume anything about the data generation mechanism. Unlike in traditional statistics, it is not needed that the data form a sample from a population with some probability law. The objective is not to estimate an assumed but unknown distribution, but to find good models for the data. Whether good models correspond to the ‘true’ models is a question that is left unanswered.

To formalize our ideas, one needs to decide on a description method, that is, a formal language in which to express the models. The choice of description method should attribute shorter description lengths for simpler functions. Nevertheless, any method is somewhat arbitrary. Just as Kolmogorov complexity, it gives an objective quantization *up to a certain constant* only. Refined MDL, based on the stochastic complexity of a model, tries to counter this problem [Grünwald et al., 2005], but this theory will not be elaborated here. I will stick to so-called *crude MDL* and try to use ‘intuitively reasonable’ codes.

Related Methods

Several other related methods, such as Minimum Message Length (MML) [Wallace and Dowe, 1968] [Wallace, 2005] or the prequential interpretation given by Dawid [1992], also provide generic solutions to the model selection problem. They are based on the same principles. The MDL method can also be given a Bayesian interpretation, which links it with traditional statistical methods, as explained in Grünwald et al. [2005, p. 13, p. 64].

Practical Regression Analysis

The problem of curve fitting, described by Example 2.3, can be handled by MDL. Curve fitting searches for the best trade-off between function complexity and goodness-of-fit. The model class $\mathcal{M}$ is populated with functions appropriate for the system under study. I added the polynomials up to a certain degree, the inverse, power, square root and step function. The description of the hypotheses then contains the values of the function's parameters, each needing d bits, and the function type, for which I count 1 byte for each operation (addition, subtraction, multiplication, division, power, square root and logarithm) in the function. A floating-point value is encoded with d bits, whereas an integer value i requires $\log(i)$ bits.

It is shown that the optimal precision d in bits for the description of each parameter is given by $d = 1/2 \log_2 n + c$, with n the sample size and c some constant [Rissanen, 1989].

Thus

$$L(H) = \#parameters.\frac{\log_2(n)}{2} + 8.\#operations + K \tag{2.12}$$

with K a constant term that does not depend on H. Therefore it does not play any role in finding the minimal description among all functions.

The second part of the description, $L(D \mid H)$, reflects the goodness-of-fit of the curve $Y = f(\mathbf{X})$. By choosing the normal distribution as probability distribution of the errors (the deviances of the data with respect to the curve), $L(D \mid H)$ equals the sum of squared errors:

$$L(D \mid H) = \sum_{i=1}^{n} (y_i - f(x_i))^2 \tag{2.13}$$

The regression analysis has to minimize the sum of Eq. 2.12 and Eq. 2.13.

Example 2.10 (Curve Fitting II).

Reconsider the problem of finding the best curve that fits the data sample, shown in Fig. 2.6. The following table gives the curves and the complexities (in bits, 'o. pol.' stands for 'order polynomial'):

curve type	function	$L(D \mid H)$	$L(H)$	**total**
linear	$Y = 5.136 + 0.444X$	193	17	**210**
2^{nd} *o. pol.*	$Y = 4.677 + 0.676X$ $-0.021X^2$	192	49	**241**
3^{th} *o. pol.*	$Y = 6.768 - 1.18X$ $+0.392X^2 - 0.026X^3$	184	72	**256**

The results show clearly that the linear function is the best trade-off between complexity and goodness-of-fit.

2.5 Kolmogorov Minimal Sufficient Statistic

This concept enables the separation of regularities from accidental, random information.

2.5.1 Regularities as Meaningful Information

Inferring models aims at explaining observed data and predicting future data. Explanation and prediction are based on patterns or **regularities**. What is there to learn from random data? Take the example (2.4.1) of random and regular strings. The random string has the highest complexity, but its bits cannot be regarded as meaningful information. We cannot predict the continuation of the string. The repetitive pattern of '0001' by the second string, on the contrary, provides the necessary knowledge for successful prediction of the continuation of the string.

The meaningful information consists of the regularities that allow the *compression* of the data, i.e. to describe the data using fewer symbols than the number of symbols needed to describe the data literally [Vitányi, 2002]. **Meaningful information** is defined as the bits of a description that are responsible for compression.

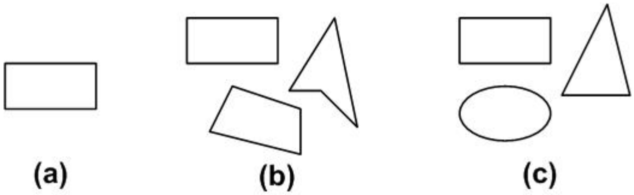


Figure 2.7: 3 examples of an observed sample of figures.

Although regularities represent an intuitive concept, no formal definition exists. A proposal for a definition might be that *regularities* are properties that correspond to a part p' of a description p, such that all objects sharing the property can use p' in a description that is shorter than the literal encoding of the object. Regularities enable compression.

In the context of learning, not all bits of the shortest description of data can be regarded as meaningful information. We will seek a description in 2 parts, one containing the meaningful information, which we put in the model, and one the remaining random 'noise', which put in the data-to-model code. This will result in a two-part code, as explained in the previous section on MDL. The model part should capture all significant regularities present in the data. The more regularities there are, the more the data can be compressed.

Learning has to be viewed as a process of building a model by squeezing out all regularities from the data.

Example 2.11 (Observing Figures).

Consider the three samples containing figures, Fig. 2.7. The questions we would like to get answered comprise the following. *What is the best model for each of the samples? What other objects can we expect to be member of the best model? What are the regularities?*

In the case of polygons, their structure determines the regularities. A rectangle can be described shorter than an irregular quadrilateral. A rectangle has fewer degrees of freedom. The straight edges are the regularities.

On the other hand is an image with a polygon highly regular compared with a random image of $n \times m$ black or white pixels. The latter requires $n \times m$ bits for its description, while the former can be described with about $e \times (\log n + \log m)$ bits.

Where e is the number of edges of the polygon. The coordinates of each vertex require $\log n + \log m$ bits.

2.5.2 Typicality

For sake of simplicity and clarity, I limit this discussion to models that can be related to a finite set of objects. The theory can be generalized to the model class of computable probability distributions [Gács et al., 2001]. An object is called *typical* for a set, when it shares its regularities with the other elements of the set [Li and Vitányi, 1997, Section 1.9]. Atypicalness of an object is measured by its **randomness deficiency** with respect to the set, which is defined as [Vitányi, 2005]

$$\delta(x \mid S) = \log |S| - K(x \mid S^*) \tag{2.14}$$

The lack of typicality of x with respect to a finite set S containing it, is the amount by which the complexity of x as an element of S falls short of the maximal possible complexity ($\log |S|$). Typical elements, having zero randomness deficiency, cannot be described shorter, given the description of the set, then by a code of $\log |S|$ bits, which corresponds to its index in the set.

Example 2.12 (Bit strings II).

> Consider the set of the strings containing 1000 bits. This set contains 2^{1000} elements. The great majority of its elements is random and can not be described with less than 1000 bits. A string 250 times repeating a substring of '0001' is not considered as a random outcome of this set. Although the probability of both strings is equal, the regular one is not perceived as random [Li and Vitányi, 1997]. It is not typical, it can be described with fewer bits. This string is a typical element of the set containing the 250-repetition of 4-bit substrings, containing $2^4 = 16$ substrings. The string is also typical for the set containing the m-repetition of n-bit substring, with n of the same complexity as 4.

By simple counting arguments it follows that most elements of a set are typical. There can only be a few elements with lower complexity. The large majority of the elements of a model set are incompressible, while only a few exhibit additional regularities that allow further compression. When choosing an element at random, the probability of picking a regular string is very low. Most strings are random. So the outcome of a random pick will be a typical element with high probability. Thus, if the observed data is typical, the minimal model corresponds to the correct model. This is a necessary condition for the correctness of the learning methods.

For correct model learning, observed data must be typical for the system under study.

2.5.3 Sufficient Statistics

A statistic is a well-known concept of traditional statistics. Every function $T(D)$ of a data sample D is called a **statistic** of D. A statistic is called a **sufficient statistic** if the information it has about the probability distribution equals the information the data has about the probability distribution[Gács et al., 2001]:

$$I(D; p) = I(T(D); p) \tag{2.15}$$

where $I(.;.)$ is the mutual information (see Introductory Chapter). A sufficient statistic contains all information in D about p, where p indicates the family of probability mass functions that is put into consideration. As Fisher [1922] puts it: “the statistic chosen should summarize the whole of the relevant information supplied by the sample. This may be called the Criterion of Sufficiency.”

Note that for every statistic the Data Processing Inequality holds (defined in the introductory chapter):

$$I(p; D) \geq I(p; T(D)), \tag{2.16}$$

because $p \rightarrow D \rightarrow T(D)$ is a Markov chain.

Example 2.13 (Coin tosses).

Take a biased coin with unknown parameter $P(head) = \theta$. The outcomes of the coin tosses are expected to be independent of each other. Given n coin tosses, it can be proved that the number of 1’s is a sufficient statistic for θ, see [Cover and Thomas, 1991, p. 37].

The **minimal sufficient statistic** is defined as the smallest sufficient statistic [Cover and Thomas, 1991]:

$$p \rightarrow T(D) \rightarrow U(D) \rightarrow D \tag{2.17}$$

It maximally compresses the information about p in the sample. It contains no irrelevant information, it is a function of every other sufficient statistic U.

Example 2.14 (Minimal sufficient statistic of a Normal distribution).

> Consider a random variable X which is normally distributed with parameters μ and σ:
>
> $$P(X) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.18}$$
>
> and a sample x^n (the set of observations $x_1, x_2, \ldots, x_n$) which is drawn independently according to this distribution. p is then the family of Normal distributions. The minimal sufficient statistic of the data is then the mean $\overline{x^n} = 1/n \sum_{i=1}^{n} x_i$ and the variance $Var(x^n) = 1/n \sum_{i=1}^{n} (x_i - \overline{x^n})^2$. It can be verified that the conditional distribution of x^n conditioned on n, $\overline{x^n}$ and $Var(x^n)$ does not depend on μ and σ.

2.5.4 Algorithmic Sufficient Statistics

The corresponding algorithmic variant of minimal sufficient statistic exhibits the interesting properties we are looking for.

The goal is to find a model set S that contains x and the objects that share x's regularities. The **Kolmogorov structure function** $K_k(x \mid n)$ of x is defined as the log-size of the smallest set including x which can be described with no more than k bits [Cover and Thomas, 1991]:

$$K_k(x \mid n) = \min_{\substack{p: l(p) \leq k \\ \mathcal{U}(p,n)=S \\ x \in S}} \log |S| \tag{2.19}$$

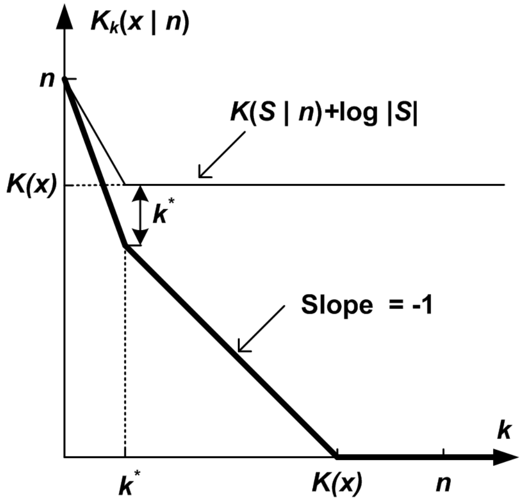


Figure 2.8: Kolmogorov structure function for n-bit string x, k^* is the Kolmogorov minimal sufficient statistic of x.

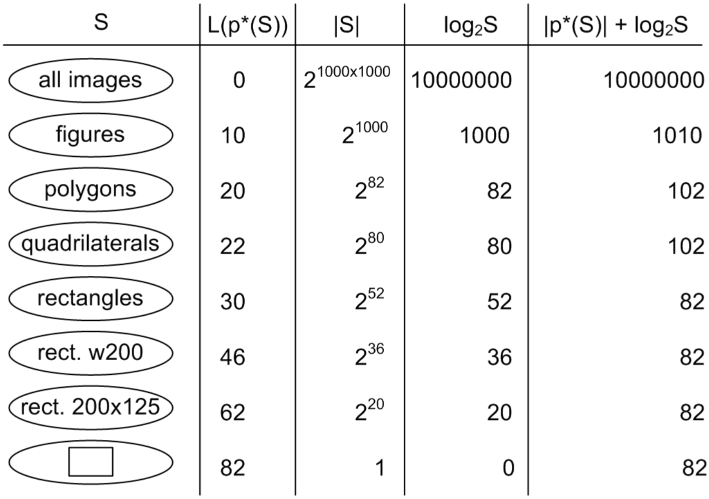
with $|S|$ the size of set S. By $\mathcal{U}(p, n) = S$, we mean that running the program p with data n on the universal computer $\mathcal{U}$[2] will print out all elements of S.

A typical graph of the structure function is illustrated in Figure 2.8. By taking $k = 0$, the only set that can be described is the entire set $\{0, 1\}^n$ containing 2^n elements, so that the corresponding log set size is n. By increasing k, the model can take advantage of the regularities of x in such way that each bit reduces the set's size more than halving it. The slope of the curve is smaller than -1.

When k reaches k^*, all regularities are exploited. There are no more patterns in the data that allow further compression. From then on each additional bit of k reduces the set by half. We proceed along the line of slope -1 until $k = K(x)$ and the smallest set that can be described is the singleton $\{x\}$. The curve $K(S \mid n) + \log|S|$ is also shown on the graph. It represents the descriptive complexity of x by using the two-part code. With $k = k^*$ it reaches its minimal and equals to $K(x)$. When $k < k^*$, S is too general and is not a typical set for x. Only for higher values than k^* is x typical for S.

For random strings the curve starts at $\log|S| = n$ for k=0 and drops with a slope of -1 until reaching the x-axis at $k = n$. Each bit reveals one of the bits of x, and halves the model set.

[2] The concatenated string of p and n is given to the Turing machine.



S	L(p*(S))	|S|	log$_2$S	|p*(S)| + log$_2$S
all images	0	$2^{1000x1000}$	10000000	10000000
figures	10	2^{1000}	1000	1010
polygons	20	2^{82}	82	102
quadrilaterals	22	2^{80}	80	102
rectangles	30	2^{52}	52	82
rect. w200	46	2^{36}	36	82
rect. 200x125	62	2^{20}	20	82
□	82	1	0	82

Figure 2.9: Model sets for the observation of a rectangle.

Example 2.15 (Learning Figures II).

Consider the observation of a rectangle of 200 × 125 in a black and white image of 1000 × 1000 pixels. I will try to draw the Kolmogorov structure function of this observation. Fig. 2.9 shows sets of increasing complexity. The second column gives the approximated Kolmogorov complexity of the sets. I omit the details about the shortest programs. The third column gives the number of elements in the set, the fourth column the minimal size of an index which can enumerate all elements. The last column gives the total description length of the observed rectangle when using the description of the set (second column) and its index in the set (fourth column).

The color of each pixel can be described with one bit. The literal description of the image thus requires 1000×1000 bits. The Kolmogorov structure function is constructed by increasing k and exploiting more and more of the figure's structure. The set's description becomes more complex, but the index length decreases faster so that the total description length decreases. This continues until a more detailed description of the set does not lead anymore to a compression of the total description. Such as the set of all rectangles with width of 200 (indicated as 'rect. w200') or all rectangles with width 200 and height

125 (indicated as 'rect. 200x125') or the singleton of only the observed rectangle. All regularities of the observation are described, the remaining information that is added to the model is random information. In this example, k^* of Fig. 2.8 is 30. The corresponding model - the set of all rectangles - is called the Kolmogorov Minimal Sufficient Statistic.

The **Kolmogorov minimal sufficient statistic** of x is defined as the program p^* that describes the smallest set S^* such that x is a typical element of S^* and the two-stage description of x is as good as the best single-stage description of x [Gács et al., 2001]:

$$p^* = min\{l(p) \ \mid \mathcal{U}(p) = S,\ x \in S,\ K(S^* \mid n) + \log|S^*| \leq K(x)\} \quad (2.20)$$

The descriptive complexity of S^* is then k^*. Program p^* thus describes the meaningful information, and nothing else, present in x.

Example 2.16 (Rain).

Consider that we want to transmit a picture of rain through a channel with limited capacity [Vereshchagin and Vitányi, 2002, p.4]. One can opt for lossy compression as follows: only send the meaningful information and let the receiver choose the random information to construct a movie which has the same regularities. One transmits background, a description of a drop (the characteristics of its form, its average and variance of the size) and the regularities of the falling drops (frequency, direction, . . .). The receiver has all sufficient information to reproduce a picture of rain which has the same characteristics but in which the particular drops are chosen randomly. With a random deficiency of almost zero, the created picture will be indistinguishable from the original.

2.6 Conclusions of Chapter

Is it rational to form inductive concepts?
Does it make sense to generalize from limited experience?
For the simple fact that knowledge conceived from inductive inference cannot be proved to be invariably true. As opposed to deduction in logic which

relies on ‘absolute’ proofs: if B follows from A and C from B, then we know as a fact that C follows from A.

This chapter addressed the problem of inductive inference. Some of the approaches were outlined which try to answer the above questions. The maximum entropy principle states that, among the models conform the given information, the one with maximal uncertainty should be chosen. For broadening the class of problems for which this principle can be employed, the constraint ‘conform the given information’ was translated into the constraint that the observations must form a ‘typical outcome’ of the model.

Occam’s razor or the quest for minimal-sized explanations is formalized through the concept of Kolmogorov complexity. The complexity of an individual object is the length of the shortest program that computes the object and then halts. The success of Kolmogorov complexity for inductive inference is determined by its application in approaches such as Minimum Description Lenght (MDL). The derived concepts of Kolmogorov complexity offer formal definitions for typicality and the separation of regularities from noise by respectively randomness deficiency and Kolmogorov Minimal Sufficient Statistic (KMSS).

After having formally defined typicalness, we can employ the maximum entropy principle:

> In order to learn from observations, choose the model which has the maximal entropy and for which the data is typical.

This result relates to the KMSS. If we assume that each element of the model set is equally probable, the entropy of the model is $\log |S|$. The KMSS then corresponds to the best model given by the maximum entropy principle: the data is typical and the entropy is maximal because the set is of maximal size. The data not typical for a smaller model set, in which case some regularities are left out. A larger model would add accidental information about the data, in which case the data is still typical, but the model is not minimal anymore.

The next chapter will explain causal inference, about how one can understand the causal structure of a system by observations. In the subsequent chapter, the principles and the validity of causal inference will be evaluated with the here introduced concept of KMSS.

Chapter 3

Graphical Causal Models

WHAT value does a concept have that cannot be formally defined? Some people might answer none. Causality is such a concept. We all intuitively understand it, use it in every day language, can immediately indicate what the causes of phenomena are... But scientists have long immersed themselves in it without coming up with a definite conclusion about the formal meaning of causality. Even worse, at the start of modern statistics, with the invention of correlation by Pearson, causality was completely banned from the realm of science. The discussion about the exact definition of causality and its impact on science still continues, but has not yet attained a decisive conclusion. Just as the concept *intelligence*, a colloquial word which is not yet completely understood.

Of what help can computer science be? Intelligent computers are a long-living promise still waiting to be realized. In order to be able to mimic intelligence with computer programs, a complete and thorough understanding of intelligence is required. Computer science thus forces us to get to the bottom of such still 'obscure' concepts as causality and intelligence.

One approach to understand causality is given by the theory of graphical causal models. Graphical causal models describe the relations among entities with the intention to reveal the underlying physical mechanisms that govern the system under study. Together with the learning algorithms they provide an interesting, yet controversial discussion of what we can learn about the world. This chapter presents current theory on graphical causal models or more specifically, *causally interpreted Bayesian networks*, as developed by Pearl et al.

This chapter is organized as follows. After an in depth analysis of conditional independencies, on which causal theory is based, Bayesian networks are introduced as representing probability distributions as well as the independencies of a distribution. In the following section graphical causal models are defined. It begins by touching on the fierce debate of what causality actually is. Next the definition proposed by Pearl, based on interventions, is given. This is followed by the link which is established between graphical causal models and Bayesian networks. The section is concluded with demonstrating why Markov networks fail as a good description method of probability distributions. The last section presents the basic constraint-based causal structure learning algorithm, the PC algorithm, and the assumptions under which the correct model will be learned.

3.1 Independencies

Dawid, Spohn, Pearl, Verma, and others started, in the 1970's, to consider the probabilistic independencies as important properties of a system under study. They started to interpret the graph (DAG) of a Bayesian network as a representation of the conditional independencies of a joint distribution [Pearl, 1988]. This will be the key for the formalization of causal models and the construction of learning algorithms.

Definition 3.1. ***Conditional independence*** *of X and Y given Z, written as $X \perp\!\!\!\perp Y \mid Z$, is defined as*

$$P(X, Y \mid Z) = P(X \mid Z).P(Y \mid Z) \tag{3.1}$$

The joint distribution is simply the multiplication of the distributions of the single variables solely. Unconditional independence of X and Y appears when $P(X, Y) = P(X).P(Y)$. Since in general $P(X, Y \mid Z) = P(X \mid Z).P(Y \mid X, Z)$ holds by the chain rule, an independency implies that

$$X \perp\!\!\!\perp Y \mid Z \quad \Leftrightarrow \quad P(Y \mid X, Z) = P(Y \mid Z) \tag{3.2}$$

Furthermore, by an independency, the conditional distribution can also be rewritten as:

$$U \perp\!\!\!\perp W \mid V \Leftrightarrow P(U \mid w) = \sum_{v \in V} P(U \mid v).P(v \mid w) \quad \text{whenever} \quad P(v, w) > 0 \tag{3.3}$$

The information shared by U and W is also present in V. The knowledge of W does not provide additional information about U once V is known. Note that variables are denoted by upper case letters, values or outcomes of variables by lower case letters.

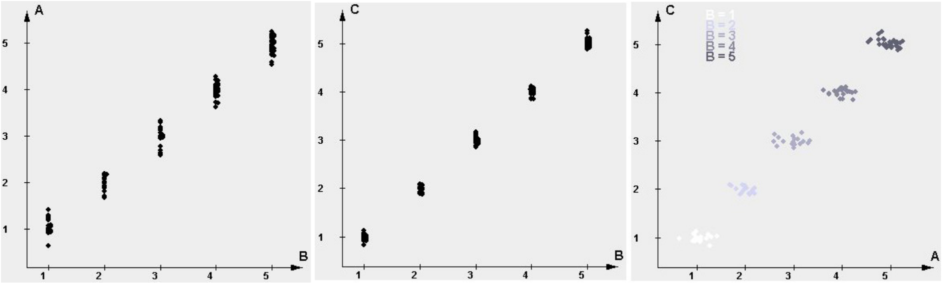


Figure 3.1: Example of Conditional Independence.

Example 3.2 (Conditional Independencies).

The data shown in Fig. 3.1 is an example of a conditional independency, namely $A \perp\!\!\!\perp C \mid B$. B is a discrete variable chosen randomly between 1 and 5. The values of A and C are generated by $A = B + \epsilon$ and $C = B + \epsilon$ respectively, with ϵ a small random disturbance. The last plot of the figure shows that A and C are dependent ($A \not\perp\!\!\!\perp C$), but get independent when conditioned on B ($A \perp\!\!\!\perp C \mid B$). When you fix B to a certain value, the relation between A and C is clearly unrelated.

3.1.1 Correlation

Traditional statistics are founded on the concept of *correlation*, invented by Galton and Pearson. Its 'discovery' created a revolution in the beginning of the 20$^{\text{th}}$ century. It indicates a probabilistic association between random variables. In the narrow sense, correlation, also called **correlation coefficient**, indicates the strength and direction of a linear relationship between two random variables. In the broad sense **correlation** or co-relation refers to the departure of two variables from independence, irrespective of the nature of the relation, be it linear or not. To avoid confusion, one better talks about **association**.

It is quantified by the Pearson product moment correlation coefficient of two variables X and Y, defined as

$$r_{xy} = \frac{\text{cov}(x,y)}{\sqrt{\text{var}(x)\text{var}(y)}} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 . \sum_{i=1}^{n}(y_i - \overline{y})^2}} \tag{3.4}$$

where $\overline{x}$ and $\overline{y}$ indicate the mean value of the x and y values respectively, var(.) the variance of a continuous variable and cov(.,.) the covariance between two variables.

Form-free dependence can be measured with the mutual information $I(X;Y)$, as defined by classical information theory (discussed in Section 2.1). To estimate it from data, I will rely on a Kernel Density Estimation (KDE, see section 2.2.1) to determine the underlying probability distribution on which the definition of mutual information can be applied. The mutual information measures the decrease in uncertainty of one variable when the state of the other is known. If the uncertainty becomes zero, the state of the variable can be predicted from the state of the other. We say that X is determined by Y. Pearson's correlation coefficient does not distinguish the closeness to linearity and the certainty on the dependence, it is a sum of both.

Example 3.3 (Examples of related data).

Fig. 3.2 shows some examples of two-dimensional data. Pearson correlation coefficient and the mutual information is calculated. Pearson gives a value of 1 for perfect linearity. If linearly related, high positive values for $x_i - \overline{x}$ consequently correspond to high, either positive or negative values of $y_i - \overline{y}$, and vice versa. The denominator of the definition scales the values, so that the coefficient lies between 0 and 1, and is maximal for a perfect linear relationship. When not perfect linear, the product of the nominator is not maximal, resulting in lower values. In absence of a positive or negative trend of Y with increasing X values, the coefficient reaches 0. If X and Y behave independently of each other, then large positive deviations of X from its mean, will be just as likely to be paired with large or small, negative or positive, deviations of Y from its mean. These will cancel each other out in the long run and the expectations of these two deviations will be zero.

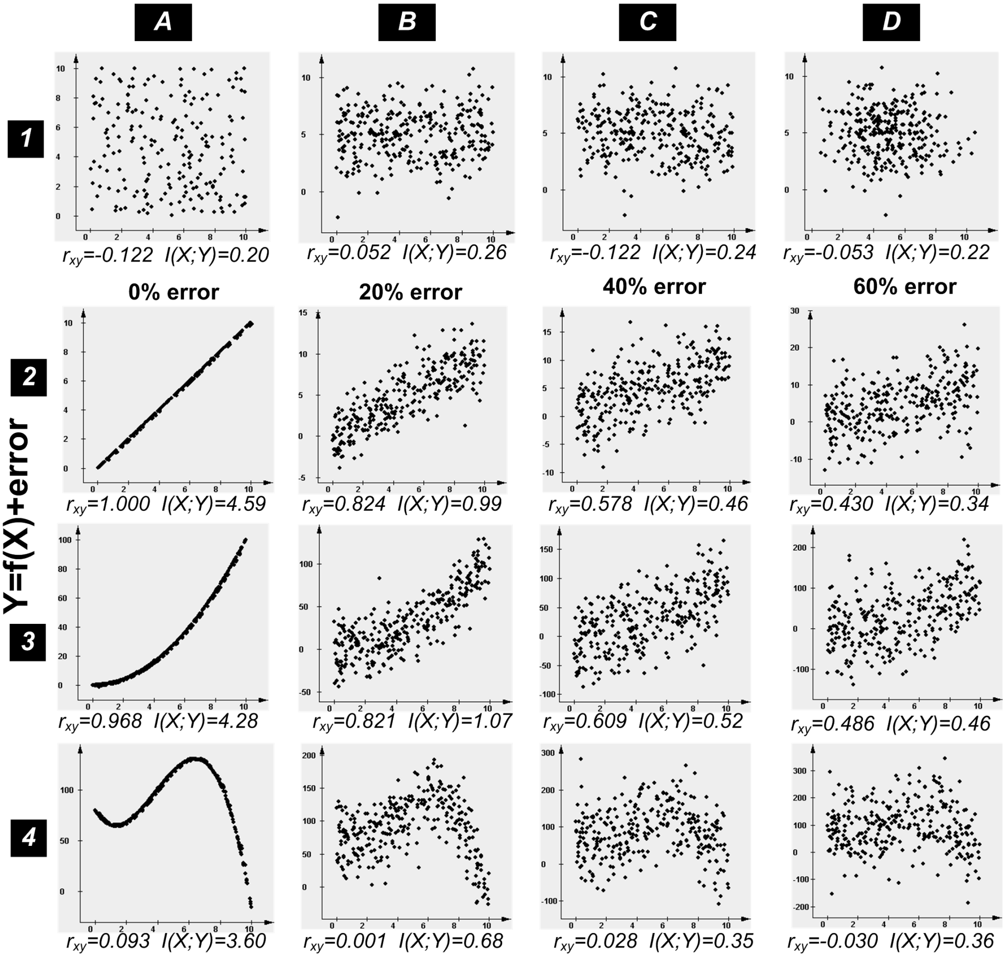


Figure 3.2: Two-dimensional data with Pearson correlation coefficient and mutual information.

> The bottom row data shows an example of non-linear related variables with a nearly 0 correlation coefficient.

3.1.2 Qualitative Property and Qualitative Reasoning

Despite the quantitative definition of conditional independence, independencies are *qualitative properties.* As Pearl puts it in [Pearl, 1988, p.79]:

> “ The traditional definition of independence uses equality of numerical quantities, as in $P(x, y) = P(x).P(y)$, suggesting that one must test whether the joint distribution of X and Y is equal to the product of their marginals in order to determine whether X and Y are independent. By contrast, people can easily and confidently detect dependencies, even though they may not be able to provide precise numerical estimates of the probabilities.”

And he continues further on the same page:

> “ Evidently, the notions of relevance and dependence are far more basic to human reasoning than the numerical values attached to probability judgments. In a commonsense reasoning system, therefore, the language used for representing probabilistic information should allow assertions about dependency relationships to be expressed qualitatively, directly, and explicitly. The verification of dependencies should not require lengthy numerical manipulations but should be accomplished swiftly with a few primitive operations on the salient features of the representation scheme. Once asserted, these dependency relationships should remain a part of the representation scheme, impervious to variations in numerical inputs.”

So, what we are looking for is a representation scheme that allows explicit modeling of conditional independencies, regardless of the probability distribution, which allows us to perform *qualitative reasoning.* When making a diagnosis, a doctor knows which symptoms occur with which diseases, without being able to put precise numerical quantities on the probabilities. At best he uses approximations for the probabilities, he knows that one disease is more probable than another. Most of the diagnosis is based on qualitative information. A doctor knows what effect a disease has on the body and uses this information to reason qualitatively about the possible mechanisms that put the human body in its current state.

Qualitative information also triggers decisions. Take the fact that for certain laws it makes no sense to implement them without public consent or a general information campaign for the public. The knowledge of this guides a government when taking decisions.

3.1.3 The Occurence of Independencies

Recall that $P(X \mid Y, Z) = P(X \mid Z)$ is equivalent to

$$\forall x \in X_{dom}, y \in Y_{dom}, z \in Z_{dom} : \quad P(x \mid y, z) = P(x \mid z) \tag{3.5}$$

The existence of one value of x, one value of y and one value of z for which the above equation is false is sufficient to make X and Y dependent. Most distributions will not exhibit the above *regularity* and result in dependence of X and Y. At least the most elements of the set of all joint distributions over X, Y and Z. These are the *typical elements.*

But consider the generation mechanism $T \to U \to V$, in which U is generated by a mechanism that only depends on the value of T, and V by a mechanism only depending on U. For this $T \perp\!\!\!\perp V \mid U$ holds, *irrespective of the 'nature' of both mechanisms*, irrespective of the exact parameterization of $P(U \mid T)$ and $P(V \mid U)$. The attentive reader remembers the discussion of the previous section, discussing the 'modeling of regularities'. He understands that this is an example that shows the relevance of regularities. Thus:

> Conditional independencies are not to be expected unless the generating mechanisms exhibit a specific structure.

This property will be exploited by the causal structure learning algorithms to reveal the structure of the underlying mechanisms.

Example 3.4 (Examples of related data II).

Unfortunately, making distinctions between dependencies and independencies in experimental data is not a 'black and white' issue. On the data of Fig. 3.2 we see that the estimated correlation coefficient or mutual information is not exactly 0 for independent variables (row 1). We have to choose a threshold that determines dependency. In the independency test of the TETRAD tool for causal analysis, this threshold is set to 0.5 when using Pearson's correlation coefficient. It can be seen that then Pearson fails to detect the non-linear relations of the bottom row. They have a correlation coefficient of almost 0. On the other hand, the mutual information estimated with KDE, give positive values. By calibration of our KDE-based test, the threshold was set to 0.35 (Section 7.3.3). The data of the bottom row is then correctly classified as dependent. The test however fails to recognize the linear relation with 60% errors such as example (2D), for which the estimated mutual information of 0.34 is below the threshold.

3.1.4 Graphoid Properties

The analysis of conditional independencies is based on the Graphoid Properties [Pearl, 1988]. They are given below, the intuitive interpretations added to the equations are taken literally from Pearl's book. For all disjoint subsets $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{Z}$ holds (sets of stochastic variables are represented by boldface capital letters):

1. *Symmetry*

$$\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y} \mid \boldsymbol{Z} \;\Leftrightarrow\; \boldsymbol{Y} \perp\!\!\!\perp \boldsymbol{X} \mid \boldsymbol{Z} \tag{3.6}$$

   In any state of knowledge $\boldsymbol{Z}$, if $\boldsymbol{Y}$ tells us nothing new about $\boldsymbol{X}$, then $\boldsymbol{X}$ tells us nothing new about $\boldsymbol{Y}$.

2. *Contraction, Decomposition and Weak Union*

$$\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y} \mid \boldsymbol{Z} \;\&\; \boldsymbol{X} \perp\!\!\!\perp \boldsymbol{W} \mid \boldsymbol{Z}, \boldsymbol{Y} \;\Leftrightarrow\; \boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y}, \boldsymbol{W} \mid \boldsymbol{Z} \tag{3.7}$$

   The left-to-right arrow tells that if we judge $\boldsymbol{W}$ irrelevant to $\boldsymbol{X}$ after learning some irrelevant information $\boldsymbol{Y}$, then $\boldsymbol{W}$ must have been irrelevant before we learned $\boldsymbol{Y}$. From right-to-left, the property means that irrelevant information should not alter the relevance of other propositions in the system; what is relevant remains relevant, and what is irrelevant remains irrelevant. The right-to-left equation incorporates the decomposition and weak union properties. Decomposition says that if two combined items are judged irrelevant to $\boldsymbol{X}$, then each separate item is irrelevant as well:

$$\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y}, \boldsymbol{W} \mid \boldsymbol{Z} \;\Rightarrow\; \boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y} \mid \boldsymbol{Z} \;\&\; \boldsymbol{X} \perp\!\!\!\perp \boldsymbol{W} \mid \boldsymbol{Z} \tag{3.8}$$

Weak union says that learning irrelevant information $\boldsymbol{W}$ cannot help the irrelevant information $\boldsymbol{Y}$ become relevant to $\boldsymbol{X}$:

$$\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y},\ \boldsymbol{W} \mid \boldsymbol{Z} \ \Rightarrow\ \boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y} \mid \boldsymbol{Z},\ \boldsymbol{W} \tag{3.9}$$

3. *Intersection*
   If P is strictly positive:

$$\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Z} \mid \boldsymbol{Y},\ \boldsymbol{W}\ \ \&\ \ \boldsymbol{Y} \perp\!\!\!\perp \boldsymbol{Z} \mid \boldsymbol{X},\ \boldsymbol{W} \ \Rightarrow\ \boldsymbol{X},\ \boldsymbol{Y} \perp\!\!\!\perp \boldsymbol{Z} \mid \boldsymbol{W} \tag{3.10}$$

   When $\boldsymbol{X}$ does not affect $\boldsymbol{Z}$ when $\boldsymbol{Y}$ is held constant nor $\boldsymbol{Y}$ affects $\boldsymbol{Z}$ when $\boldsymbol{X}$ is held constant, neither $\boldsymbol{X}$ nor $\boldsymbol{Y}$ nor their combination can affect $\boldsymbol{Z}$.

These properties tell us how independencies imply other independencies. For the intersection property, strict positiveness is put forward as a necessary condition. This condition is, however, far too strict, weaker forms exist that guarantee that the intersection property holds [Martín et al., 2005]. The violation of the property will be treated thoroughly in chapter 5. The property is mainly broken by *deterministic relations*. When $Y = f(X)$, $P(Y \mid X)$ is 1 for exactly one value of Y and zero for the others.

3.1.5 Markov networks

Because of the *symmetry* of correlations, an undirected graph is the most intuitive and obvious choice while looking for a structured representation of the relations among the variables, . Such a graph is called a **Markov network** [Pearl, 1988]. Pairs of variables that are directly related are connected by an unoriented edge. Markov networks follow directly from classical information theory (discussed in Section 2.1). Recall that a **Markov chain** is written as $X - Y - Z$, and implies $X \perp\!\!\!\perp Z \mid Y$. Retrieving (in)dependencies from the graph is straight forward. Two variables are dependent if they are connected by at least one unblocked path in the graph. A **path** between two nodes is a sequence of edges connecting both nodes. A path is a **blocked path** as soon as one of the variables along the path is a member of the conditioning set.

Example 3.5 (Markov network).

Fig. 3.3 shows an example Markov network over 6 variables. The independencies that follow from the graph include the following:

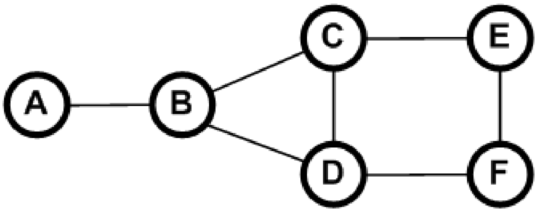


Figure 3.3: Example Markov Network.

- $A \perp\!\!\!\perp C, D, E, F \mid B$; but $A \not\perp\!\!\!\perp F \mid D$
- $B \perp\!\!\!\perp E, F \mid C, D$; but $B \not\perp\!\!\!\perp E, F \mid C$
- $C \perp\!\!\!\perp F \mid D, E$; but $C \not\perp\!\!\!\perp F \mid E$.
- ...

Note that independencies are transitive and that two (conditionally) independent variables never become dependent by conditioning on some additional variables. At first sight, Markov networks correctly represent the relational structure among variables. But this is not true.

Consider the system in which Y is caused by two unrelated, random variables X and Z: $X \rightarrow Y \leftarrow Z$. The arrows have a causal meaning. The interpretation is that X and Z exert influence on Y. The mechanism that generates the state of Y is affected by the values of X and Z. Then, X and Z are marginally independent, but become dependent when conditioned on Y. In general, two marginally independent variables will be correlated if one conditions on any of their common effects.

Example 3.6 (Counter-intuitive dependence).

Consider a student who did not pass an exam. Let us assume that he didn't study hard enough or either he didn't understand the subject matter. This is represented by causal model *hard work* → *pass exam* ← *understanding matter*. Consider that *understanding matter* and *hard work* are not related. Some students can study very hard without ever getting to understand the material properly. But if we know a student worked hard without passing the exam, we can deduce that the failing of the exam is probably due to a bad understanding. Accordingly:

- *understanding matter* $\perp\!\!\!\perp$ *hard work*

- *understanding matter* $\not\perp\!\!\!\perp$ *hard work* | *pass exam*

Causal model $X \rightarrow Y \leftarrow Z$ is called a **v-structure**. V-structures play a major role in causal model theory. They will turn out to be the key for detecting causal relations from observations. Their properties require a representation by directed graphs. In contrast with v-structures, are the Markov chains present in causal models $X \rightarrow Y \rightarrow Z$, $X \leftarrow Y \leftarrow Z$ or $X \leftarrow Y \rightarrow Z$. Here, the Markov property applies, thus:

Markov chain	$X \not\perp\!\!\!\perp Y$	$Y \not\perp\!\!\!\perp Z$	$X \not\perp\!\!\!\perp Z$	$X \perp\!\!\!\perp Z \mid Y$
v-structure	$X \not\perp\!\!\!\perp Y$	$Y \not\perp\!\!\!\perp Z$	$X \perp\!\!\!\perp Z$	$X \not\perp\!\!\!\perp Z \mid Y$

The v-structure is due to the asymmetry of causality. Variable Y is called a **collider** along the path from X to Z. I will write a Markov chain for which X and Z are dependent as $X - Y - Z$.

3.2 Bayesian Networks

Bayesian networks are mainly concerned with offering a dense and manageable representation of probability density distributions, called probability distribution for short[1]. They have become a popular representation for encoding uncertain expert knowledge in expert systems [Cowell et al., 1999]. A Bayesian network records a state of probabilistic knowledge, provides means for updating the knowledge as new information accumulates and facilitates query answering mechanisms for knowledge retrieval. The novelty that Pearl et al. introduced was to see the graph of the network also as a representation of conditional independencies.

3.2.1 Representation of Distributions

A joint probability distribution is defined over a set of stochastic variables $X_1 \dots X_n$ and defines a probability ($P \in [0,1]$) for each possible state $(x_1 \dots x_n) \in X_{1,dom} \times \dots \times X_{n,dom}$, where $X_{i,dom}$ stands for the domain of variable X_i. A joint distribution can be **factorized** relative to a variable

[1] Also it must be noted that I limit the discussion to finite distributions. This avoids some theoretical problems with infinite distributions. In the context of learning, in which we have to rely on limited information and uncertainty, this is a valid assumption. Discretized distributions will be learned.

ordering $(X_1, \ldots, X_n)$ as follows:

$$P(X_1, \ldots, X_n) = P(X_1). \prod_{i=2}^{n} P(X_i \mid X_1, \ldots, X_{i-1}) \tag{3.11}$$

Each variable is represented by a factor, namely its probability distribution given the previous variables in the ordering. But, variables from the conditioning set can be removed from the set if they do not provide information on the variable under consideration. Formally, X_j is obsolete in the factor of X_i if X_j becomes conditionally independent from X_i by conditioning on the rest of the set:

$$P(X_i \mid X_1 \ldots X_{i-1}) = P(X_i \mid X_1 \ldots X_{j-1}, X_{j+1} \ldots X_{i-1}) \tag{3.12}$$

or

$$X_i \perp\!\!\!\perp X_j \mid X_1 \ldots X_{j-1}, X_{j+1} \ldots X_{i-1} \tag{3.13}$$

Conditional independencies of this form can be exploited to reduce the complexity of the factors in the factorization. The more of such independencies, the simpler the description of the joint distribution will be.

The conditioning sets of the factors can be described by a Directed Acyclic Graph (DAG), in which each node represents a variable and has incoming edges from all variables of the conditioning set of its factor. The joint distribution is then described by the DAG and the Conditional Probability Distributions (CPDs) of all variables conditioned on its parents, $P(X_i \mid parents(X_i))$. A **Bayesian network** is a factorization that is *edge-minimal*, in the sense that no edge can be deleted without destroying the correctness of the factorization. It maximally exploits independencies of the form of Eq. 3.13. Accordingly, a Bayesian network defined over a set of variables consists of a DAG in which each node represents a variable and a CPD of each variable, expressing the conditional probability distribution of a variable conditional on its parents in the graph.

Although a Bayesian network is edge-minimal, it depends on the chosen variable ordering. Some orderings lead to the same networks, but others result in different topologies. As shown in the following example.

Example 3.7 (Different factorizations).

> Consider 5 stochastic variables A, B, C, D and E and a joint probability distribution defined over them. Fig. 3.4 shows the graph that was constructed by simplifying the factorization based on variable ordering (A, B, C, D, E) by the three given

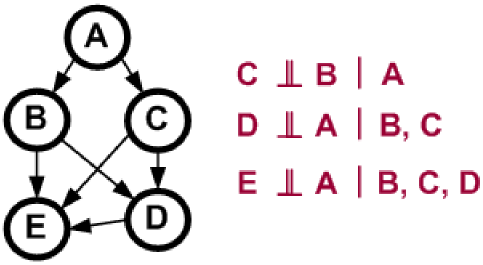


Figure 3.4: Factorization based on variable ordering (A, B, C, D, E) and reduction by three independencies.

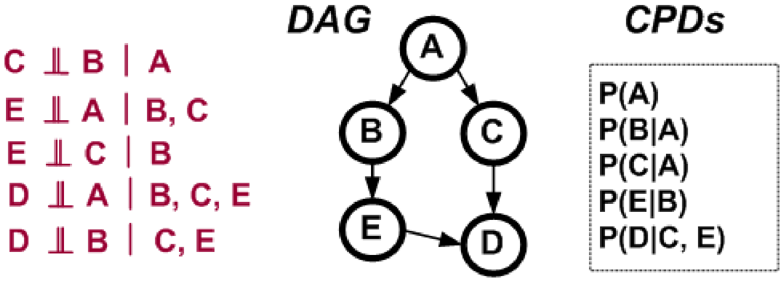


Figure 3.5: Bayesian network based on variable ordering (A, B, C, E, D) and five independencies.

conditional independencies. Assume also that the factorization of the same distribution but based on variable ordering (A, B, C, E, D) leads to the Bayesian network depicted in Fig. 3.5. In that case, both models represent the same distribution, but one contains 2 edges less.

3.2.2 Representation of Independencies

The **Markov condition** gives the conditional independencies that come from the factorization reduction (Eq. 3.12): *any node is independent of all its non-descendants by conditioning on its parents.* Take care not to mix this condition with the Markov *property*. Combinations of the basic independencies imply other independencies, following from the graphoid properties. Pearl and Verma constructed a graphical criterion, called *d*-separation, for retrieving all independencies from the graph of a Bayesian network that follow from the Markov condition.

Definition 3.8. *(d-separation) Let p be a path between a node X and a node Y of a DAG G. (By a path we mean any succession of edges, regardless of their directions.) Path p is called unblocked given subset* $\boldsymbol{Z}$ *of nodes in G if every node w on p satisfies:*

1. *if w has two converging arrows along p, w or any of its descendants is in* $\boldsymbol{Z}$.
2. *and if w has no converging arrows, it is not an element of* $\boldsymbol{Z}$.

X and Y are called d-connected given $\boldsymbol{Z}$, *if there is an unblocked path between them in G. Conversely,* $\boldsymbol{Z}$ *is said to d-separate X from Y in G, denoted $X \perp Y \mid \boldsymbol{Z}$, iff* $\boldsymbol{Z}$ *blocks every path from X to Y.* $\boldsymbol{Z}$ *blocks a path if the above condition is not valid for one of the nodes on the path.*

Example 3.9 (d-separation).

Take the Bayesian network of Fig. 3.5. The d-separation criterion tells us that variable B separates A from E, since B blocks the path $A \rightarrow B \rightarrow E$. On the other hand, the path $A \rightarrow C \rightarrow D \leftarrow E$ is blocked by $C \rightarrow D \leftarrow E$, which is called a $v-structure$. This path gets unblocked given D. Now, with the definition of d-separation, one can verify that the Bayesian network of Fig. 3.4 is indeed edge-minimal when considering the conditional independencies present in the network of Fig. 3.5. In the factorization of Fig. 3.4, A must for example be connected with D because $A \not\perp\!\!\!\perp D \mid C$, unless $A \perp\!\!\!\perp D$. Both independencies can be read from the graph of Fig. 3.5.

A graph is an **Independence Map**, or **I-map** for short, of a joint distribution if every independency found in the graph appears in the distribution. The DAG of a Bayesian network is a minimal I-map, removing an edge from the graph destroys its I-mapness. A graph is called faithful if it is an I-map and the independencies that follow from Markov are the only independencies of the distribution.

Definition 3.10. *(Faithfulness) A DAG is called* ***faithful*** *for a distribution if for all disjoint subsets* $\boldsymbol{A}$, $\boldsymbol{B}$ *and* $\boldsymbol{C}$:

$$\boldsymbol{A} \perp\!\!\!\perp \boldsymbol{B} \mid \boldsymbol{C} \Leftrightarrow \boldsymbol{A} \perp \boldsymbol{B} \mid \boldsymbol{C} \tag{3.14}$$

In other words:

> Faithfulness requires that each conditional independency in the distribution corresponds to a *d*-separation in the graph.

The non-minimal factorization of Fig. 3.4 is not faithful. For example, the graph does not represent independency $C \perp\!\!\!\perp E \mid B$. Although the oriented edges intuitively let us think of causal relations, or at least of an asymmetrical relation, they shouldn't be interpreted as such. Bayesian networks make no statements about causality. The theory of Bayesian networks is not contradicted.

3.3 Graphical Causal Models

This section introduces graphical causal models. The reader may consult references [Pearl, 2000][Spirtes et al., 1993][Tian and Pearl, 2002][Shipley, 2000] for a complete theoretic elaboration. Causal models intend to graphically describe the structure of the underlying *physical mechanisms* governing a system under study. A physical mechanism is a process that determines the state of a variable. All variables that influence the outcome of the process are called **causes** of the outcome variable. An **indirect cause** produces the state of the effect indirectly, through another variable. If there is no intermediate variable among the known variables, the cause is said to be a **direct cause**. A direct cause is a relative concept. It is relative to the set of variables under consideration, the context. A direct cause of A becomes an indirect cause of A if a more direct cause is added. Once the outcomes of the direct causes of A are determined, then whether or not $A = a$ occurs no longer has anything to do with whether the events that are indirect causes of A occur. The direct causes *screen off* the indirect causes from the effect. The direct causes can be regarded as input variables of the effect's generation mechanism. The effect is than the output of the mechanism. In this interpretation, causality deals with 'mechanisms'.

In its most general form, the mechanism of X_i can be described by the Conditional Probability Distribution (CPD) $P(X_i \mid \textit{direct causes}(X_i))$. It mathematically describes the outcome of the stochastic process by which the values of X_i are chosen in response to the values of its direct causes. The mechanism is regarded as a *black box*, it is described as the output versus the input.

The description of the direct causes over a set of variable of interest can be given by a Directed Acyclic Graph (DAG). Each node of the graph

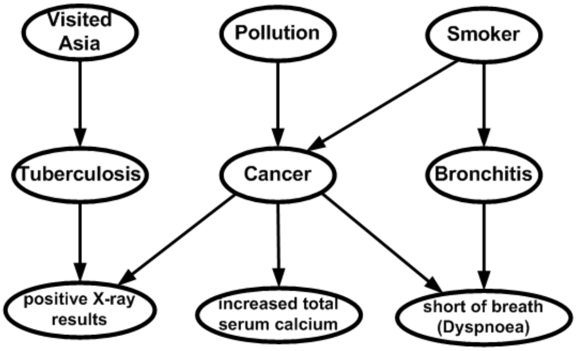


Figure 3.6: Example Causal Model from the field of Medicin.

represents one of the variables and has incoming edges from all its direct causes. Together with the CPDs, the model describes a joint probability distribution:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid parents(X_i)) \tag{3.15}$$

This is the ground for the *probabilistic account* of causality.

Example 3.11 (Medical Causal Model).

Fig. 3.6 shows an example of a causal model in the medical field. *Environmental factors* are shown at the top row, *diseases* at the middle and *symptoms* at the bottom. The causal relations among the variables are shown with directed edges. The occurrence of a disease is influenced by environmental and other factors. A disease disrupts the normal functioning of the human body. A disease will usually not be observed directly, but through observable effects, called symptoms. For instance, having cancer is more likely when living in a polluted area or being a regular smoker. The relation is described by $P(cancer \mid pollution, smoker)$. The probability of having cancer is then calculated as follows:

$$P(cancer) = P(cancer \mid pollution, smoker).P(pollution).P(smoker) \tag{3.16}$$

3.3.1 Defining Causality: The Shadow's Cause

Without formally having defined 'causality', everyone intuitively understands the concept of cause and effect. Also in the previous example, it is clear that environmental factors are direct causes of disease, but only indirect causes of symptoms. Despite our intuitive understanding of causality and our daily usage of the concept, the formalization of causality has given rise to a lot of intellectual contention. Since the beginning of science, all major philosophers, from Aristotle, via Galileo and Newton, to Leibniz and Descartes, and scientists from different fields, such as Physics, Biology, Sociology, Economy to Computer Science, have pondered the open question. A wide diversity of solutions has been given, but until now no satisfactory answer has emerged. Read the interesting appendix of [Pearl, 2000] for an historical overview of the discussions relating to the concept causality. Williamson [2005] gives an overview and dicussion of the different views circulating today: be it mechanistic, probabilistic, counterfactual, agency, manipulability or epistemic.

Not only is there no consensus about the definition of causality, since the invention of the concept of correlation, the scientific validity and value of the concept itself was questioned. Causation has to do with *production* (the effect happens to be produced by the cause), but, as Hume pointed out, it is not empirically verifiable that the cause actually produces or engenders the effect. It is only verifiable that the (experienced) event called 'cause' is invariably associated with or followed by the (experienced) event called 'effect'. We only observe events and the stochastic co-occurrence of events, we do not actually observe which one is responsible for the other. Pearson concluded that causality was an outdated and useless concept. The proper goal of science was to simply measure direct experiences (phenomena) and to economically describe them in the form of mathematical functions; and not try to explain phenomena. If a scientist could predict the likely values of variable Y after observing the values of variable X, then he would have done his job. The 'why' question is irrelevant, only the 'how' question is relevant. Also Bertrand Russell insisted that causality did not belong to modern science:

> "the law of causality ... like much that passes muster among philosophers, is a relic of a bygone age, surviving, like the monarchy, only because it is erroneously supposed to do no harm."

It must admitted that he later retreated from this extreme view, recognizing the fundamental role of causality in physics [Druzdzel and Simon, 1993].

The aim of this work is not to dig deeper into the discussion of what causality really is. But the reader may surely consider me as a believer of the concept of 'causality'. This belief is based on the observations that causality is part of our everyday language, ànd of our scientific language, ànd the fact that there is a assymetric component to association. I will try to understand what causal model theory, as built by Pearl, has to say about causality.

I just want to add some more flavor to this topic, about how Shipley paints the problem. According to him, we are unwitting participants in the nature's *Shadow Play* [Shipley, 2000]. These shadows are cast when the causal processes in nature are intercepted by our measurements. All that can be directly observed are the consequences of these processes in the form of complicated patterns of association and independence in the data. Non-observable causal relations can only be observed through its 'shadow': the correlations it entails. As with shadows, these correlation patterns are incomplete - and potentially ambiguous - projections of the original causal processes. We try to infer the underlying causal processes from their shadow (observational data). Correlations are the way of experimentally validating hypothesized causal relations.

This points out another discussion: *causation clearly implies correlation, but does correlation always imply causation?* To Shipley [2000, p. 3] the answer is invariably yes, barring a few exceptions ("Tuesday follows Monday"). Any simple correlation between two variables implies an unresolved causal structure: one of both variables is the cause of the other or a third, possibly hidden, variable is the common cause of both. Bunge on the other hand, sees causation as only one category of general determinism [Bunge, 1979, p. 17, p. 29].

3.3.2 Interventions

Pearl defines causality formally through *interventions*, which relies on the modularity assumption [Pearl, 2000]. A causal model over n variables represents n different physical mechanisms, each responsible for generating the state of one variable. Hence, a causal model breaks up, for all variables X_i, into independent components of the form $P(X_i \mid parents(X_i))$. Each component corresponds to a specific physical mechanism. 'Physical' implies that the mechanisms are physically separated, besides the variables there is no physical connection between them. Each component goes to the 'heart' of the data generation mechanism. Thus, mechanisms can vary independently of one another, a property which implies *modularity*: 'Each

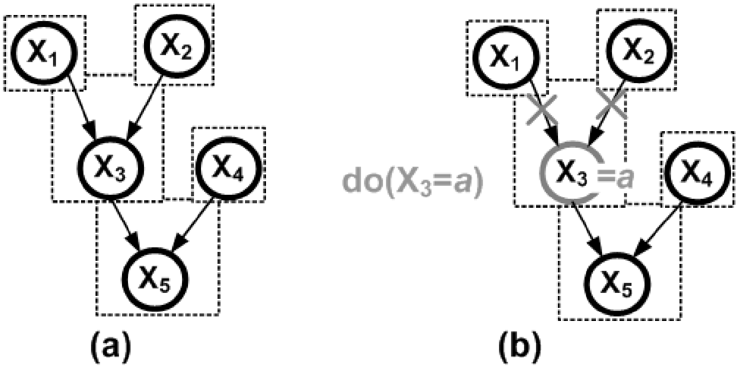


Figure 3.7: Example causal model showing the decomposition (a) and the effect of an intervention (b).

assignment process remains invariant to possible changes in assignment processes that govern other variables in the system'.

Modularity results in *autonomy*: a part of the model can be altered without affecting the rest of the model. Pearl introduced the *do*(.) operator to formally define autonomy. This new mathematical operator describes the effect of interventions to the model. Interventions are defined as specific modifications of some factors in the product of the factorization (Eq. 3.15). An **intervention** forces a variable to a given state and eliminates the corresponding factor from the factorization.

Example 3.12 (Interventions).

> Modularity is shown in Fig. 3.7(a). Fig. 3.7(b) shows how setting X_3 through an intervention results in a new model with a fixed state of X_3.

This results in the following definition [Pearl, 2000, p. 23]:

Definition 3.13 (Graphical Causal Model)**.** *Let $P(\boldsymbol{v})$ be a probability distribution on a set $\boldsymbol{V}$ of variables, and let $P(\boldsymbol{v} \mid do(\boldsymbol{x}))$ denote the distribution resulting from the intervention $do(\boldsymbol{X} = \boldsymbol{x})$ that sets a subset $\boldsymbol{X}$ of variables to constants $\boldsymbol{x}$. A DAG G is said to be a graphical causal model compatible with $P(\boldsymbol{v} \mid do(\boldsymbol{x}))$ if and only if the following three conditions hold for every $\boldsymbol{X} \subseteq \boldsymbol{V}$:*

1. *G is an independence map of $P(\boldsymbol{v} \mid do(\boldsymbol{x}))$;*

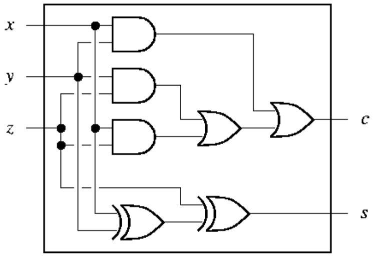


Figure 3.8: Digital Circuit, a causal model used in engineering.

2. *$P(x_i \mid do(\boldsymbol{x}))$=1 for all $X_i \in \boldsymbol{X}$ whenever x_i is consistent with $\boldsymbol{X} = \boldsymbol{x}$;*

3. *$P(v_i \mid do(\boldsymbol{x}), pa_i) = P(x_i \mid pa_i)$ for all $V_i \notin \boldsymbol{X}$ whenever pa_i is consistent with $\boldsymbol{X} = \boldsymbol{x}$.*

A causal model not only enables the prediction of a state of unknown variables (like Bayesian networks), but also the prediction of the effect of local changes made to the system. Causality inherently provides autonomous components that can be reused when parts of the systems are used in different contexts. They are therefore extremely useful in engineering or decision-making, as in politics or economics. In fact, models used in engineering are implicitly based on the properties of causality. This is shown by the following example.

Example 3.14 (Digital Circuit).

Consider the model of a digital circuit shown in Fig. 3.8, the scheme makes it possible to predict the output when the input nodes are set to a certain value. But it also allows reasoning on how parts of the component should be changed to attain a certain desired transfer function.

We can conclude:

Understanding the causal structure enables the prediction of the effects of changes to the system.

3.3.3 Causally Interpreted Bayesian Networks

Pearl links causality with probability theory by showing its correspondence to Bayesian networks. The clue is that a causal model *by its structure* implies conditional independencies, which are *independent of the precise parameterization of the CPDs.* These independencies allow us to learn the causal structure of a system from observation. The **Causal Markov Condition** describes the independencies that follow from a causal structure: each variable is probabilistically independent from all its non-descendants conditional on its direct causes. The condition has two aspects: a probabilistic and a causal one.

The probabilistic aspect of the Causal Markov condition is similar to the Markov condition. A causal model can therefore be regarded a Bayesian network in which all edges are interpreted as representing causal influences between the corresponding variables. It combines components $P(X_i \mid parents(X_i))$ of all variables X_i. A causal model therefore implies a joint distribution:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid parents(X_i)) \tag{3.17}$$

The correspondence to a joint distribution and the validity of the Markov condition results in a close connection between causal and probabilistic dependence [Spohn, 2001]. The theory can thus be regarded as an extension to the theory of Bayesian networks.

The extension is based on the *causal interpretation* of the edges of a Bayesian network. A graphical causal model is therefore also referred to as a 'causally interpreted Bayesian network'. The causal interpretation represents the second aspect of the Causal Markov Condition: *every probabilistic dependence has a causal explanation.* This is known as the *Principle of the Common Cause* [Williamson, 2005].

Difference between Bayesian Networks and Causal Models

The basic difference between both is that Bayesian networks offer dense descriptions of probability distributions, while causal models intend to say something about the system under study. Everything that holds for Bayesian networks holds for graphical causal models. Yet, causal model add the following:

1. A causal model is a unique model describing the mechanisms of the system. Multiple Bayesian networks exists though, modeling the same system, but based on a different variable ordering.

2. Through the knowledge of the underlying mechanisms, one understands how a system is built. This makes it possible to predict the effect of changes to the system.

3. Predicting the probabilities of unknown variables in the light of the available information is possible with both. Bayesian networks correctly model systems quantitatively, but not qualitatively. Causal models describe all conditional independencies that follow from the system's structure.

3.3.4 Invalidity of the Markov Network Representation

As we have seen in Section 3.1.5, Markov networks cannot represent *v*-structures. If we want a model to be an I-map, i.e. represent only conditional independencies that are present in distribution, edges have to be added for any v-structure. Causal model $X \rightarrow Z \leftarrow Y$ implies the dependency of $X \not\perp\!\!\!\perp Y \mid Z$ and $X \perp\!\!\!\perp Y$. Markov network $X - Z - Y$, however, implies $X \perp\!\!\!\perp Y \mid Z$ (and $X \not\perp\!\!\!\perp Y$). To prevent this false independency, we should connect X and Y. The medical model of Fig. 3.6 can only be represented by a Markov network that is an I-map by adding edges, as shown in Fig. 3.9. The causal relations are shown by thicker lines. The problem is that the relations in this model do not correspond to any real direct relation. Consider the edge between $Visited\ Asia$ and $Pollution$. This edge had to be added to prevent independency $Visited\ Asia \perp\!\!\!\perp Pollution \mid Positive\ X - ray\ results$. Thus the relation between both stems from variable $Positive\ X - ray\ results$. The resulted Markov network is then an I-map, but can never be faithful.

The moral is that a wrong modeling framework, which wrongly models the regularities (here: the conditional independencies), results in models far from reality.

3.4 Causal Structure Learning Algorithms

The goal of causal inference algorithms is to learn the causal structure of a system based on observational data. Basically, there are two types: constraint-based and scoring-based. An overview is given by Korb and Nicholson [2003].

Scoring-based algorithms [Korb and Nicholson, 2003] are an optimized search through the set of all possible models, which tries to find the minimal model that best describes the data. Each model is given a score that is a trade-off between model complexity and goodness-of-fit. An objective

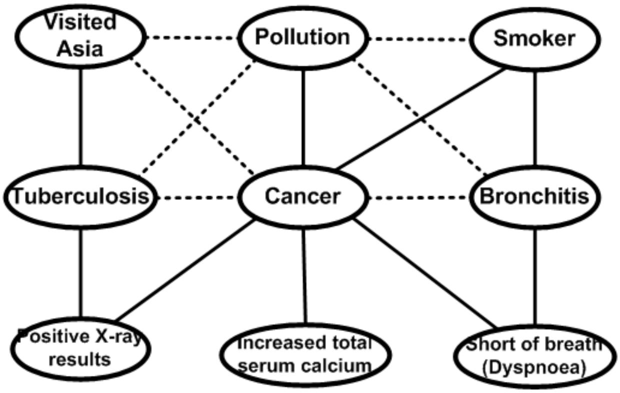


Figure 3.9: Medical model represented by a Markov network that is an I-map.

score is attained by using the MDL or MML approach [Comley and Dowe, 2003][Comley and Dowe, 2005][Lam and Bacchus, 1994].

The constraint-based learning algorithms are based on the pioneering work of Spirtes, Glymour and Scheines [Spirtes et al., 1993]. The standard algorithm is the *PC algorithm.* It is implemented and integrated in the TETRAD tool for causal analysis, developed by the Dept. of Philosophy of Carnegie Mellon University and freely available [Spirtes et al.].

3.4.1 The PC Algorithm

The graph is constructed in two steps. The first step, called *adjacency search*, learns the undirected graph. The second step tries to orient the edges. The result of the algorithm is a set of observationally indistinguishable models, the so-called *Markov-equivalence* class, and is proved to be correct for distributions that are faithful to some directed acyclic graph.

The algorithm takes background knowledge into account, prior belief about whether one variable directly influences another. The user can specify edges that are required or forbidden and can put constraints on orientations. If prior belief forbids adjacency, the algorithm need not bother to test for that adjacency. If prior belief requires that there be a direct influence of one variable on another, the corresponding directed edge is imposed. During the algorithm, prior belief is assumed to override the results of unconstrained search.

The construction of the undirected graph, the first step, is based on the property that two nodes are adjacent if they remain dependent by conditioning on every set of nodes that does not include both nodes. The algorithm

Algorithm 3.1 Steps of the PC causal structure learning algorithm

Input: data set, independence test, background knowledge

Output: partially oriented, acyclic graph

1. Part 1: determining adjacency
    (a) remove edges from knowledge
    (b) correlation search
    (c) conditional independencies: gradually increase the size of the conditioning set
2. Part 2: determining orientation of the edges
    (a) orientation from knowledge
    (b) look for v-structures
    (c) orient for non-v-structures

starts with a completely connected, undirected graph. There is an edge between every pair of nodes. It removes edges for each independency that is found. The number of nodes in the conditioning set is gradually increased up to a certain maximal number, called the *depth* of the search. If for every pair A, B, one should test on all subsets of $V \setminus \{X, Y\}$, this results in an exponential number of tests to perform. One can however limit the number of subsets to be tested. For detecting adjacency, it is sufficient to test for conditional independencies of A and B given subsets of variables adjacent to A and subsets of variables adjacent to B that are on undirected paths between A and B. It relies on the following property [Spirtes et al., 1993].

Lemma 3.15. *If a distribution P over variables V is faithful to a Bayesian network with DAG G, and $X, Y \in V$, then:*
If X and Y are d-separated by any subset of $V \setminus X, Y$, then they are d-separated either by $Parents(X)$ or $Parents(Y)$

The orientation step is based on the identification of v-structures, for which two nodes are independent, but become dependent conditional on a third node. In that case, both nodes are causes of the third node and are oriented towards it. Recall that for all three other possible orienta-

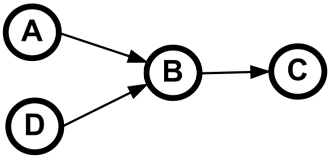


Figure 3.10: A simple causal model.

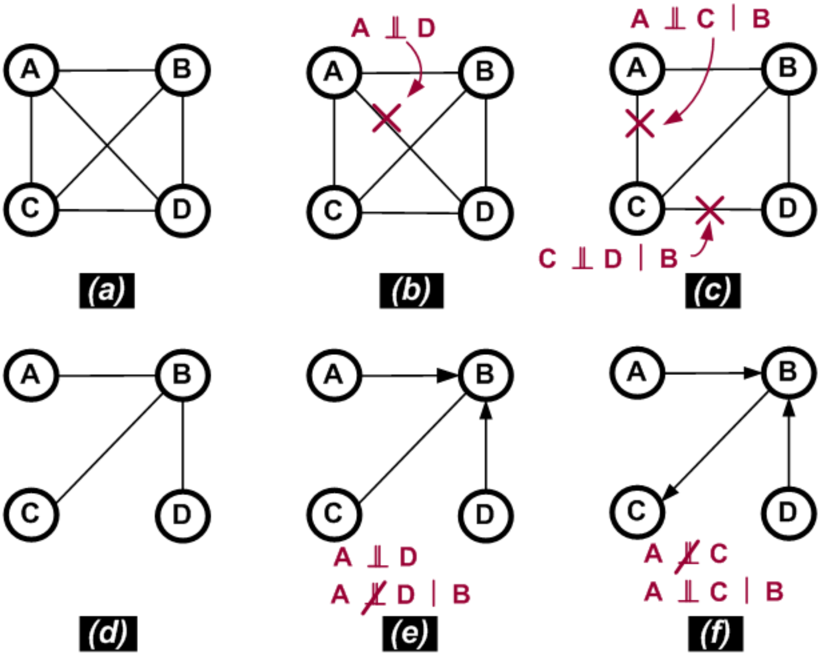


Figure 3.11: Execution of the PC algorithm when learning the model of Fig. 3.10.

tions of the three nodes the opposite is true, the two nodes are initially dependent, but become independent by conditioning on the third node. Next, when all v-structures are identified, some edges may be directed if an opposite orientation would result in a v-structure. In absence of enough v-structures, it might be that there is not enough information to direct all edges. The orientations of these edges are left undecided. This results in a partially oriented, acyclic graph, representing the Markov equivalence class of graphs that are representative for the data.

Example 3.16 (Learning the causal structure).

> Take data that is representative for the model of Fig. 3.10. The results of the different steps while executing the PC algorithm are depicted in Fig. 3.11. The algorithm starts with a completely connected undirected graph *(a)*. At first it will remove edge $A - D$ since both variables appear to be marginally independent *(b)*. In the next step, one conditions on one variable for all pairs of variables that are still adjacent. Two independencies imply the removal of edges $A - C$ and $A - D$ *(c)*. The first part of the algorithms ends with an undirected graph in which all adjacent variables are directly related *(d)*. During the orientation step, v-structure $A \rightarrow B \leftarrow D$ is detected. This results in the orientation of $A - B$ and $A - D$, both towards B *(e)*. Finally, the $B - C$ relation can be oriented as $B \rightarrow C$, since an opposite orientation would lead to v-structure $A \rightarrow B \leftarrow C$ which is not confirmed by the independencies *(f)*. The result corresponds to the expected model.

3.4.2 Assumptions

The validity of the PC algorithm is based on the *minimality principle*, the *causal Markov condition* and the *faithfulness property* [Spirtes et al., 1993]. The first principle guarantees minimality of the model, the second that correlation implies causation and the third that all conditional independencies follow from the causal structure. Spirtes, Glymour and Scheines rely in their work on causal models on an *axiomatization* of these 3 conditions.

The TETRAD manual states that under 7 assumptions the algorithm will find the correct equivalence class of indistinguishable causal models [Scheines et al., 1996] (see also [Spirtes et al., 1993, p. 80]):

1. The correctness of the background knowledge input to the algorithm.
2. Whether the Markov Condition holds. E.g., it is violated in mixtures of populations in which causal connections are in opposite directions for different subpopulations.
3. Whether the Faithfulness Condition holds.
4. Whether the distributional assumptions made by the statistical tests hold. One assumption is that the experiment should be typical, i.e. random. We will choose the input parameters randomly from a uniform distribution.

5. The power of the statistical tests against alternatives.

6. The significance level used in the statistical tests.

7. The PC algorithm requires *causal sufficiency*, ie. that all common causes should be known: variables that are the direct cause of at least two variables. It must be noted that more sophisticated learning algorithms exist that are capable of detecting latent common causes.

TETRAD does not offer a formal mechanism for combining these factors into a score for the *reliability* of the output. It also does not detect whether one of the assumptions, such as faithfulness, is not valid.

3.4.3 Complexity

When analyzing n variables and a depth of two the worst-case complexity of the algorithm is $O(n^{depth})$. In practice, the complexity will be limited to $O(n^3)$. The first step of the algorithm, the construction of the undirected graph is clearly the most computation-intensive part. First, when checking the associations, $n \times (n-1)$ independencies have to be checked. When conditioning on one variable, $n \times e_0 \times 2e_0$ independencies have to checked, where e_0 is the average number of remaining edges of a node. e_0 will in most cases be proportional to n. When conditioning on two variables, $n \times e_1 \times 2(e_1 \times (e_1 - 1))$ independencies have to be checked, where e_1 is the average number of remaining edges after the previous independency tests. This value can be proportional to n, but in practice this will be a constant quasi-independent from n. If we assume that the average number of edges of a node of the faithful graph is independent from n. In that case the independency tests when conditioned on 1 variable will have removed most of the edges between indirectly related nodes. Only the ones remain that are between nodes that are related through at least two different causal paths. The number of such nodes can be assumed to be independent of n in most cases.

The algorithm is thus not very scalable. An increase of n leads to a polynomial increase of the algorithm's runtime. A solution for this problem is the insertion of background knowledge or the use of constraint-based algorithms.

3.5 Summary of Chapter

The intent of this chapter was to offer insight into the principles that form the basis for the graphical causal model theory as conceived by Pearl et al., and the accompanying learning algorithms, developed by Spirtes, Glymour and Scheines. In the second part of this work causal models will be used as a representation scheme for modeling performance and causal inference to learn them from experimental data.

The fundamental idea of the theory is that a system's causal structure implies *conditional independencies* and that these, when detected in experimental data, allow us to infer the causal structure. The learning of causal models is based on the independencies following from Markov chains (through the Markov property) and v-structures. The Markov property tells us how to make the difference between direct and indirect relations, whereas v-structures let us determine the causal orientation of the relations. Causal models are closely related to Bayesian networks, since they offer, besides a dense description of a joint probability distribution, an explicit representation of conditional independencies. The independencies that follow from a Bayesian network are given by the Markov condition. D-separation offers a graphical criterion for retrieving these independencies from the graph of the Bayesian network. Multiple Bayesian networks can exist though, based on a different factorization variable ordering. Only those with the least number of edges can come into consideration for the correct causal model. This will be further explained in the next chapter.

But what does the knowledge of causal relations learn us? What makes the difference between $A \rightarrow B$ and $B \rightarrow A$? Nothing when simply observing both variables. Asymmetrical causal relations result in symmetrical correlations. The answer lies in the effect of *changes* to the system. Pearl defined changes by interventions: forcing variable A to state a is denoted as $do(A = a)$ with the *do* operator. Here the asymmetry comes into play, an intervention on A will only influence the state of B in case that A is a cause of B. These considerations underline Shipley's metaphor of *the shadow's cause*; causality only becomes tangible through its consequences.

As a young, curious, passionate scientist, when getting to know Pearl's work, I became intrigued by the highly debated, yet controversial concept of causality. Especially given the absence of a satisfactory scientific treatment of such an intuitive, deep concept. I was completely swept away by the brightness and beauty of Pearl's theory. Yet it must be admitted, the theory receives a lot of criticism. *Is it all there is to causality? Do causal models really describe mechanisms?* Even more suspicion exists about the possibility of learning causal models from data, about the plausibility of the assumptions. So despite my enormous sympathy for the theory, I should prove my critical attitude. The next chapter is devoted to my analysis of the subject with the principles discussed in the previous chapter.

Chapter 4

The Meaningful Information of Probability Distributions

BEFORE extending the causal learning algorithms and putting them into practice, I will analyze the interpretation and validity of causal inference with the general principles of inductive inference discussed in Chapter 2. Kolmogorov complexity's approach to inference must be understood as finding the regularities of the data. The most common application of Kolmogorov complexity is to look for the minimum model in a set of models, such as employed by MDL and MML (Section 2.4.2). I propose to use a slightly different approach. For analyzing the validity of causal inference I propose to not stick to an a priori chosen model class, but rather to search actively for the regularities appearing in the data. The choice of a model class implies that only a limited number of regularities are put into consideration. *But what if some regularities of the data are overlooked? What if the real minimal model does not belong to the chosen set?*

Causal inference searches the minimal Bayesian network in the set of all possible Bayesian networks. The conditional independencies are the regularities at hand, they constitute the meaningful information. The causal interpretation relies on the fact that the minimal model tells us something about the causal structure of the system under study. The thesis advocated in this chapter is that the existence of other regularities strongly indicates that this interpretation might be incorrect.

The validity of causal inference is fiercely criticized, especially the causal interpretation of the Directed Acyclic Graph (DAG) and the validity of faithfulness [Freedman and Humphreys, 1999, Cartwright, 2001, Williamson, 2005]. It is questioned whether the causal model indeed des-

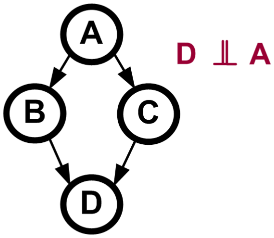


Figure 4.1: Causal model in which A is independent from D.

cribes the mechanisms of the system. And if this can be learned from observational data alone. The counterexamples suggest that we cannot rely on faithfulness. Pearl uses *stability* as the main motivation for the faithfulness of causal models [Pearl, 2000, p. 48]. Consider the model of Fig. 4.1. In general, one expects A to depend upon D. A and D are independent only if the stochastic parameterization is such that the influences via paths $A \to B \to D$ and $A \to C \to D$ cancel out exactly. This system is called unstable because a small change in the parameterization results in a dependency. The unhappy balancing act is a measure zero event, the probability of such a coincidence can therefore be regarded as nearly zero.

This chapter attempts to contribute to the discussion about these objections by applying the theory of *Kolmogorov Minimal Sufficient Statistic* (KMSS), discussed in Chapter 2. The first section shows how conditional independencies compress the description of a probability distribution and that the DAG of a causal model is the KMSS in absence of additional regularities. The graph is then faithful, it explains all independencies. Next, I argue that the causal interpretation of a Bayesian network relies on a decomposition of the model into smaller, independent components, one for each Conditional Probability Distribution (CPD). The validity of causal inference can then be studied by (a) the presence of regularities or independencies not described by the causal model and (b) whether the decomposition of the model corresponds with the mechanisms of the system. Several counterexamples are analyzed based on the answers to both questions. The chapter concludes with a survey of the different aspects of the faithfulness property.

4.1 Minimal Description of Distributions

This section investigates the Kolmogorov Minimal Sufficient Statistic of probability distributions and, more specifically, in which cases the Directed

Acyclic Graph (DAG) of a Bayesian network contains all meaningful information. The conditional independencies are the regularities that allow compression of distributions and the construction of minimal models.

4.1.1 Compression of Distributions

Recall that a joint distribution can be factorized relative to a variable ordering $(X_1, \ldots, X_n)$ as follows:

$$P(X_1, \ldots, X_n) = P(X_1). \prod_{i=2}^{n} P(X_i \mid X_1, \ldots, X_{i-1}) \tag{4.1}$$

From the theory of Bayesian networks (Section 3.2), we know that a factorization is reduced by conditional independencies of the following form:

$$P(X_i \mid X_1 \ldots X_{i-1}) = P(X_i \mid X_1 \ldots X_{j-1}, X_{j+1} \ldots X_{i-1}) \tag{4.2}$$

Formally, X_j is obsolete in the factor of X_i if X_j becomes conditionally independent from X_i by conditioning on the rest of the set. The factorization leads to $P(X_1 \ldots X_n) = \prod CPD_i$, with CPD_i the conditional probability distribution of variable X_i. The descriptive size of the CPDs is determined by the number of variables in the conditioning sets. The total number of conditioning variables thus defines the shortest factorization. A two-part description is then:

$$descr(P(X_1 \ldots X_n)) = \{parents(X_1) \ldots parents(X_n)\} + \{CPD_1 \ldots CPD_n\} \tag{4.3}$$

Note that the parents' lists can be described very compact and correspond to the description of a DAG. The description of a random DAG defined over n nodes requires $n/2.(n-1)$ bits.

The following theorems show that the first part offers the minimal model if the CPDs are random and unrelated.

Theorem 4.1. *The parents' lists, $\{parents(X_1) \ldots parents(X_n)\}$, in the two-part code given by Eq. 4.3 contain meaningful information of a probability distribution.*

Proof. The description of a non-reduced CPD_i requires $(|X_{i,dom}| - 1). |X_{1,dom}| . |X_{i-1,dom}| .d$ bits with $|X_{k,dom}|$ the size of the domain of X_k and d the precision in bits to which each probability is described[1]. Every variable X_j that can be eliminated from the conditioning set

[1] The proof is given for a uniform prior over the parameterization of the distribution.

of X_i results in a reduction of the descriptive complexity by

$$\begin{aligned}(|X_{i,dom}| - 1).\, |X_{1,dom}| \ldots |X_{j-1,dom}| \\ .(|X_{j,dom}| - 1).\, |X_{j+1,dom}| \ldots |X_{i-1,dom}|\, .d\end{aligned} \tag{4.4}$$

The information that variable X_j can be eliminated takes no more than $\log n$ bits. This is almost always lower than the above complexity reduction, except when d is taken absurdly small. It is therefore useful to describe the variables taking part in the CPDs, the parents' lists. Every bit of the parents' lists reduces the descriptive complexity by more than one bit and, hence, contains meaningful information. □

Theorem 4.2. *If the two-part code description of a probability distribution, given by Eq. 4.3, results in an incompressible string, the first part is the Kolmogorov minimal sufficient statistic.*

Proof. If a more compact description of the distribution would exist, the two-part description would contain redundant bits. Theorem 4.1 showed that the first part contains meaningful information. By hypothesizing incompressibility of the whole description, the second part of the description, consisting of the CPDs, is assumed to be incompressible. The CPDs thus do not contain meaningful information. Hence, the first part, described minimally, is the Kolmogorov minimal sufficient statistic. □

The distribution decomposes quasi-uniquely[2] and minimally into the CPDs, which, in absence of further knowledge, may be assumed to be atomic and independent. The decomposition thus offers a *canonical representation.* The system under study is decomposed into independent components that are only connected via the variables.

4.1.2 Minimality of Causal Models

The following two theorems show that the causal model corresponding to the minimal factorization is the KMSS and faithful if its DAG and CPDs are random and unrelated. Then, minimality, faithfulness and the Markov Condition are fulfilled. They make up the three conditions for causal inference (Section 3.4.2), except for the causal interpretation of the Causal Markov Condition, which is discussed in the next section.

[2] There can be multiple minimal factorizations, which are closely related though. I come back to this in the next section.

Theorem 4.3. *If a faithful Bayesian network exists for a distribution, it is the minimal factorization.*

Proof. Oliver and Smith defined the conditions for sound transformations of Bayesian networks, where sound means that the transformation does not introduce extraneous independencies [Oliver and Smith, 1990]. No edge removal is permitted, only reorientation and addition of edges. Additionally, if a reorientation destroys a v-structure or creates a new one, an edge should be added connecting the common parents in the former or in the newly created v-structure. Such transformations however eliminate some independencies represented by the original graph. Assume the existence of a Bayesian network A based on a different variable ordering that has fewer edges than the faithful network. It must be possible to transform A into the faithful one. A has fewer edges, so edges must be added by the transformation, and this destroys independencies. But network A cannot represent more independencies, because the faithful network represents all independencies. The assumption leads to a contradiction. □

Note that the inverse is not true, the minimal factorization is not always faithful. The constraint-based learning algorithms assume the existence of a faithful Bayesian network. The theorem showed that this is then the minimal factorization. So, the learning algorithms are assumed to return the minimal Bayesian networks.

Theorem 4.4. *A Bayesian network is faithful if the set of conditional probability distributions (CPDs) is incompressible (the CPDs are random and unrelated).*

Proof. Recall that a Bayesian network is a factorization that is edge-minimal. This means that for each parent $pa_{i,j}$ of variable X_i:

$$\begin{aligned} &P(X_i \mid pa_{i,1}, \dots pa_{i,j}, \dots pa_{i,k}) \\ \neq &P(X_i \mid pa_{i,1}, \dots pa_{i,j-1}, pa_{i,j+1}, \dots pa_{i,k}) \end{aligned} \tag{4.5}$$

Variables cannot be eliminated from the CPDs of a Bayesian network. The proof will show that any two variables that are d-connected are dependent, unless the probabilities of the CPDs are related. We have to consider the following possibilities. The two variables can be adjacent (a), related by a Markov chain (b) [3], a v-structure (c), a combination of both or connected by multiple paths (d).

[3] Recall that a Markov chain is a path not containing v-structures.

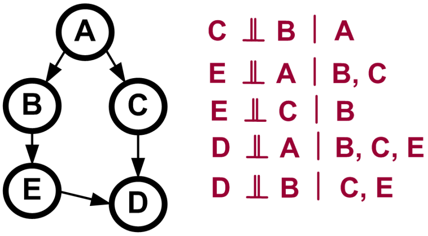


Figure 4.2: Bayesian network based on variable ordering (A, B, C, E, D) and five independencies.

First I prove that a variable marginally depends on each of its adjacent variables (a). Consider nodes D and E of the Bayesian network of Fig. 4.2. To avoid overloading the proof, I will demonstrate that $P(D \mid E) \neq P(D)$, but the proof can easily be generalized. The first term can be written as:

$$\begin{aligned} P(D \mid E) = &\, P(D \mid E, c_1).P(c_1) \\ &+ P(D \mid E, c_2).P(c_2) + \ldots \end{aligned} \tag{4.6}$$

with c_1 and $c_2 \in C_{dom}$. C is also a parent of D, thus, by Eq. 4.5, there are at least two values of C_{dom} for which $P(D \mid E, c_i) \neq P(D \mid E)$ [4]. Take c_1 and c_2 being such values, thus $P(D \mid E, c_1) \neq P(D \mid E, c_2)$. There are also at least 2 such values of E_{dom}, take e_1 and e_2. Eq. 4.6 should hold for all values of E and equal to $P(D)$ to get an independency. This results in the following relation among the probabilities:

$$\begin{aligned} & P(D \mid e_1, c_1).P(c_1) + P(D \mid e_1, c_2).P(c_2) \\ = \; & P(D \mid e_2, c_1).P(c_1) + P(D \mid e_2, c_2).P(c_2) \end{aligned} \tag{4.7}$$

Note that the equation can not be reduced, the conditional probabilities are not equal to $P(D)$ nor to each other.

Next, by the same arguments it can be proven that variables connected by a Markov chain are by default dependent (b). Take $A \rightarrow B \rightarrow E$ in Fig. 4.2, independence of A and E requires that

$$P(E \mid a) = \sum_{b \in B} P(E \mid b).P(b \mid a) = P(E) \quad \forall a \in A. \tag{4.8}$$

and this also results in a regularity among the CPDs.

[4] $P(D \mid E)$ is a weighted average of $P(D \mid E, C)$. If one probability $P(D \mid E, c_1)$ is different than this average, let's say higher, than there must be at least one value lower than the average, thus different.

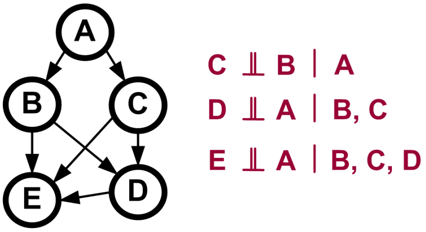


Figure 4.3: Factorization based on variable ordering (A, B, C, D, E) and reduction by three independencies.

In a v-structure, both causes are dependent when conditioned on their common effect (c), for $C \to D \leftarrow E$, $P(D \mid C, E) \neq P(D \mid E)$ is true by Eq. 4.5. Finally, if there are multiple unblocked paths connecting two variables, then independence of both variables implies a regularity, too (d). Take A and D in Fig. 4.2:

$$\begin{aligned} &P(D \mid A) \\ &= \sum_{b \in B} \sum_{c \in C} \sum_{e \in E} P(D \mid c, e).P(c \mid A).P(e \mid b).P(b \mid A). \end{aligned} \tag{4.9}$$

Note that $P(c, e \mid A) = P(c \mid A).P(e \mid A)$ follows from the independence of C and E given A. All factors from the equation satisfy Eq. 4.5, so that, again, the equation only equals to $P(D)$ if there is a relation among the CPDs. □

Table 4.1 gives an example parameterization of $P(D \mid E, C)$ in the model of Fig. 4.2 for which D and E become independent, assuming that $P(C = 0) = P(C = 1) = 0.5$. The regularity of Eq. 4.7 applies for the distribution.

E	C	$P(D \mid C, E)$
0	0	0.4
0	1	0.3
1	0	0.2
1	1	0.5

Table 4.1: Example of a CPD for which $P(D \mid E) = P(D)$, assuming that $P(C = 0) = P(C = 1) = 0.5$.

A distribution can be described by several distinct Bayesian networks, which are based on different variable orderings (see subsection 3.2.2). The networks build with a non-optimal variable ordering are however not minimal, the CPDs are related. Take the distribution described by the Bayesian network of Fig. 4.2, $E \perp\!\!\!\perp C | B$ holds. But this independency does not follow from the graph of Fig. 4.3. The graph is constructed by simplifying the factorization based on variable ordering (A, B, C, D, E) by the three given conditional independencies. It follows from the exact cancellation of the influences through the paths $C \rightarrow E$ and $C \rightarrow D \rightarrow E$, given by equation:

$$P(E \mid B, C) = \sum_{d \in D} P(E \mid B, C, d).P(d \mid B, C) = P(E \mid B). \tag{4.10}$$

Bayesian networks not based on a minimal factorization are always compressible, by the regularities among the CPDs that follow from the independencies not represented by the graph.

The existence of a unique faithful Bayesian network is not always certain, though. Multiple faithful models can exist for a distribution. These models represent the same set of independencies and are therefore statistically indistinguishable. They define a *Markov-equivalence class*. It is proved that they share the same v-structures and only differ in the orientation of some edges [Pearl, 2000]. The corresponding factorizations have the same number of conditioning variables, thus all have the same complexity. The observations do not give us information to decide on the correct model, but we have demarcated a set of closely related models which contains the correct model. The class can be described by a Partially Directed Acyclic Graph (PDAG), in which some edges are left unoriented. The constraint-based learning algorithms are able to learn this equivalence class, which can be represented by a graph in which some edges are not oriented. Thus, although the model is not unique, we know exactly which parts of the model are undecided.

On the other hand, we want to know the conditions under which a joint probability distribution can be represented by a faithful model. In [Pearl, 1988, p. 128], Pearl developed a set of necessary, but not sufficient, conditions. But he doubts that there exists an exhaustive list of conditions that can guarantee faithfulness [Pearl, 1988, p. 131]. The non-existence of such a list is approved by the theorem. Any dependency among non-adjacent variables that follows from the Markov condition can be turned into an independency by properly chosen parameterization the CPDs.

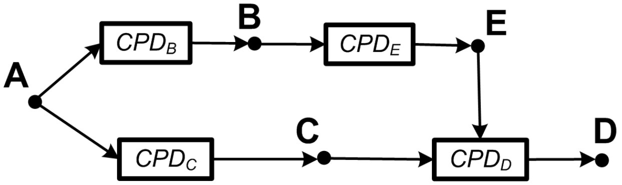


Figure 4.4: Decomposition of the causal model of Fig. 4.2 into independent components.

4.2 Correspondence of Causal Component to Decomposition

The causal interpretation of Pearl is defined by the capacity to predict from the causal model the effect of changes to the system. Where changes are defined as *interventions*, atomic operations that remove the CPD of a specific variable and set its state to a given value. The correctness of the mutilated model relies on the modularity assumption, a subsystem can be replaced by another without affecting the rest of the system. Theorem 4.2 showed that if the minimal factorization results in an incompressible description, the description corresponds to a unique decomposition of the distribution into independent components, the CPDs. Fig. 4.4 shows the decomposition for the model of Fig. 4.2. Each variable gets a CPD attributed, except input variable A. The canonical decomposition suggests modularity and makes it plausible that mechanisms may be replaced. Then - in absence of other regularities, background knowledge, experiments with interventions or other information - the most obvious hypothesis is that each CPD represents an independent part of reality, a physical mechanism.

From this we may conclude that the causal interpretation of the model is correct as long as we define causality in terms of interventions. The modularity of the decomposition captures Pearl's interventions. An intervention, which Pearl considers an atomic operation, can be seen as replacing one specific CPD with a CPD that allows perfect control over the variable (for setting it to a certain state).

The claim of this dissertation does not suggest that decomposition is all there is to causality. The hypothesis I advocate is that it reflects the causal component of graphical causal models. The consequences of causality for causal models rely on this decomposition. Just as the interventions do not give a definition of causality, but describe the benefits of understanding the structure of a system's underlying mechanisms.

4.2.1 A Distributed d-separation Algorithm

To prove the hypothesis, it should be verified that all aspects of causal model theory are conform the CPD decomposition. I will demonstrate that independencies following from the Markov condition (given by the d-separation criterion defined in section 3.2.2) can be determined by a distributed algorithm that runs on each CPD component independently, without global synchronization. The components only communicate through the variables.

The subtlety of d-separation lies in the v-structures. Knowing something about the state of one cause of X tells us nothing about the state of the other causes, unless X is known or something is known about X via its effects. This happens 'within' a CPD. This also explains why, contrary to what a graph suggests, the edges are not the atomic elements of the systems. A decomposition of the model along its edges would is not able to explain the behavior of a v-structure.

Algorithm 4.1 Distributed algorithm for the conditional information

```
Input:
Output: To be executed by each component:
    1. If the effect node is in conditioning set, set all cause nodes to true.
    2. If effect node is set to true:
        • set $infoThruEffects$ to true, and
        • set all cause nodes to true.
```

The distributed algorithm for determining $X \not\perp\!\!\!\perp Y \mid \boldsymbol{Z}$ will run on each component independently. Each component only has access to the cause nodes and the single effect node. Each node has one boolean that can be read or set by the components, and is initially set to false. Each component keeps also track of one boolean, $infoThruEffects$, initially set to false. One first runs the preparatory step defined by Algorithm 4.1. It spreads the necessary information about the conditional variables. Since they represent the a priori information, this step may be executed in advance and can be expected to be completed before running the actual algorithm. After that, variable X is set to true and Algorithm 4.2 is executed. When finished, all nodes that became true depend on X. By the acyclicity of the graph, both algorithms stop.

Other algorithms, like for inference or identifiability, rely on d-separation.

Algorithm 4.2 Distributed algorithm for determining dependence on X

Input:

Output: To be executed by each component:

1. If the effect node is X, set effect node and all cause nodes to true.
2. If the effect node is set to true by another component and is not in conditioning set: set all cause nodes to true.
3. If one of the cause nodes is set to true by another component:
    - if the effect node is not in conditioning set, set effect node to true; and
    - if the effect node is in conditioning set or $infoThruEffects$ is true: set all other cause nodes to true.

One can thus expect that they can also be written as distributed algorithms.

The causal decomposition can be said to rely on a form of *reductionism*.

4.2.2 Reductionism

According to the reductionist view, the world can be studied in parts. Parts can be isolated and analyzed independently to the rest of the world. Understanding of the parts helps us to understand the whole.

The antipole is holism, a principle which was concisely summarized by *Aristotle* in the *Metaphysics*:

> "The whole is more than the sum of its part".

Or, as Wikipedia puts it [5], **Holism** is the idea that all the properties of a given system (biological, chemical, social, economic, mental, linguistic, etc.) cannot be determined or explained by the sum of its component parts alone. Instead, the system as a whole determines in an important way how the parts behave.

Example 4.5 (Traffic Jam).

> Consider the typical accelerating and decelerating behavior of traffic during traffic jams, also called the accordion effect. A

[5] http://en.wikipedia.org/wiki/Holism

> traffic jam, for example, contains patterns of behavior which cannot be reduced to the behavior of an individual car. This indeed contradicts *strong reductionism*, according to which a system can be understood by understanding its constituent parts solely. A simple addition of the parts' behavior does not give the overall system's behavior. Weaker forms of reductionism admit that the interactions between the parts should be taken into account too. That 'the whole is more than the sum of its parts' is due to the interactions of the parts. In order to understand the behavior of the whole, a synthesis of the parts together with their interactions should be made. Interactions of subsystems can still be studied independently of the rest. For understanding the accordion effect of traffic jams, one should consider how a car reacts to the behavior of the car in front of him, making him to break or accelerate. Still, this can be studied independently of the specific street or country in which the traffic jam occurs. A relative simple model of an individual car can be designed which includes its interaction with other cars. Using this model results in traffic simulations that come close to reality.

In the weaker form, reductionism holds, subsystems and their interactions can be studied independently, abstractions can be made. This does not exclude that complex situations exist in which it is extremely difficult to know all parts and interactions that participate, such as in social sciences.

Ackoff [1974, p. 8] gives a viewpoint that goes even further. According to him

> "reductionism is a doctrine that maintains that all objects and events, their properties, and our experience and knowledge of them are made up of ultimate elements, indivisible parts."

It is not only that the world can be studied in parts, but, even more, the world exists of unique, independent parts. By this principle, one should endeavor in finding the ultimate elements when studying systems or phenomena. Systems or situations can be understood by combining the independent mechanisms. The minimal, unique and independent mechanisms form a *canonical decomposition* of the system under study. Intuitively I'm willing to go along with this point of view. Either way, it is difficult to imagine how a 'holist' world, in which mechanisms cannot stand independently, would look like, or how it needs to be studied. Opponents of reductionism do not come up with strong answers or convincing examples.

4.3 Validity of Causal Inference

I demonstrated that the minimal Bayesian network is (a) the KMSS in absence of regularities in or among the CPDs and (b) the causal interpretation relies on the CPD decomposition. This section intends to contribute to the discussion about the validity of the causal inference algorithms by analyzing counterexamples with respect to both statements:

- (a) The minimality of a causal model is challenged by the presence of regularities not incorporated into the model. Conditional independencies are a specific kind of such regularities. Unfaithfulness is an indication of non-modeled regularities. The question is then whether these regularities imply that the causal interpretation is incorrect.

- (b) The correctness of the decomposition is challenged by physical mechanisms that are responsible for producing the state of multiple variables or the presence of global mechanisms controlling other mechanisms. The question is then whether one can infer the right mechanisms from the regularities not represented by the minimal Bayesian network.

	conform reality	global mechanism	wrong decomposition
minimal model	(A)	(B)	(C)
unfaithful model	(3) (6)	(2) (7)	(5) (8)
non-modeled regularities	(1)		(4)

Table 4.2: Classification of counterexamples along 2 axes: whether there are independencies or other regularities not captured by the model and whether the CPDs correspond to real physical mechanisms.

Table 2 classifies the counterexamples according to both criteria. Case (A) happens when the inferred causal model is minimal and conform reality. While in cases (B) and (C) minimality does not imply that the model is correct. This happens when reality is more complex than suggested by

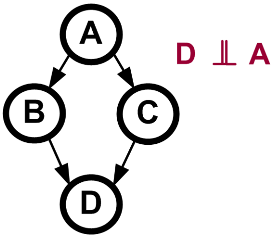


Figure 4.5: Causal model in which A is independent from D.

the complexity of the observations. Occam's razor can never guarantee correctness, but must be regarded as the most effective *strategy* for inductive inference.

Counterexample 1. The system exhibits additional regularities, but remains faithful and the decomposition is correct. A well-known example is when the description of individual CPDs can be compressed. This regularity is called *local structure* [Boutilier et al., 1996] and appears 'inside a component'.

Counterexample 2. The most-known example of unfaithfulness is when in the model of Fig. 4.5 A and D appear to be independent [Spirtes et al., 1993]. This happens when the influences along the paths $A \rightarrow B \rightarrow D$ and $A \rightarrow C \rightarrow D$ exactly balance, so that they cancel each other out and the net effect results in an independence. The independence of A and D is, however, not 'expected' by the causal model. This balancing act can give an indication of a global mechanism, such as evolution, controlling the mechanisms such that the parameters are calibrated until they neutralize [Korb and Nyberg, 2006].

Counterexample 3. Distributions with deterministic relations can not be represented by a faithful graph [Spirtes et al., 1993]. They will be thoroughly discussed in the next chapter. I show that it is related to the violation of the *intersection condition*, one of the graphoid properties that Pearl imposes on a distribution in the elaboration of causal theory and its algorithms [Pearl, 1988](Section 3.1.4). The solution I will propose is to incorporate the information about deterministic relations in an *augmented causal model.* So that the model again incorporates all regularities from the data. Next I will characterize the conditional independencies following from deterministic relations. The *d*-separation criterion is extended so that it can be used to retrieve all conditional independencies from the model. In such way, the faithfulness of the model can be reestablished. Deterministic relations correspond to non-random CPDs, but appear in nature. The CPD decomposition is still correct here.

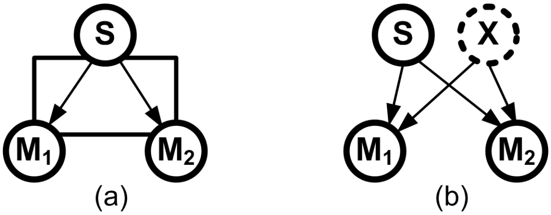


Figure 4.6: Particle with state S decays into 2 parts with momenta M_1 and M_2. (a) Represented as one mechanism and by a (b) faithful model.

Counterexample 4. For some systems, the CPDs do not represent independent mechanisms. Take the example of particle decay, one of the counterexamples of the Causal Markov Condition reported by Williamson [2005, p. 55], taken from Fraassen [1980, p. 29]:

> Suppose that a particle decays into 2 parts, that conservation of total momentum obtains, and that it is not determined by the prior state of the particle what the momentum of each part will be after the decay. By conservation, the momentum of one part will be determined by the momentum of the other part. By indeterminism, the prior state of the particle will not determine what the momenta of each part will be after the decay. Thus there is no prior screener off.

The prior state S fails to screen off the momenta. But by symmetry, neither of the two parts' momenta M_1 and M_2 can be considered as the cause of the other. This system cannot be represented by a causal model. The generation of M_1 and M_2 by S should be considered as one (causal) mechanism, as shown in Fig. 4.6 (a). The system can, however, be represented faithfully by adding a hidden common cause X, as shown in Fig. 4.6 (b). However, the CPDs of this model do not correspond to the real mechanisms. Note that the system exhibits an additional regularity, namely the perfect symmetry of the two parts. Other counterexamples of the Causal Markov Condition given in Williamson [2005, p. 52] are similar.

Counterexample 5. Take the set of strings of n bits for which m consecutive bits are 1 and the others are 0. For $n = 8$ and $m = 2$, "01100000" and "00001100" represent valid strings. Every bit can be regarded as a discrete variable. By picking valid strings randomly, the joint distribution is observed. All bits are correlated, but each pair becomes independent by conditioning on some other bits. The simplest model for this pattern contains a latent variable S, denoting the start position of the non-zero

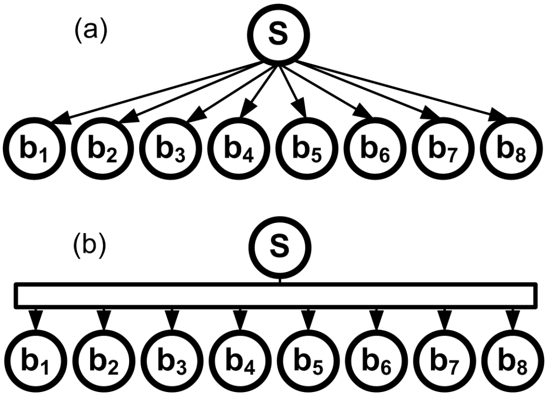


Figure 4.7: Two models for a pattern in a 8-bit string.

bit sequence. The causal model, shown in Fig. 4.7(a), however, considers each edge as a separate mechanism. But the mechanisms cannot be regarded as unrelated, the decomposition is therefore invalid. The system is highly regular and contains conditional independencies not following from Markov. The model of Fig. 4.7(b) is more accurate, it indicates that there is one mechanism generating the states of all bits.

Counterexample 6. Variables in *pseudo-independent models* are pairwise independent but collectively dependent [Xiang et al., 1996]. Consider three variables, X_1, X_2 and X_3, that are pairwise independent, but become dependent by conditioning on the third variable. For example, if the variables are binary and they are related by an exclusive or: $X_3 = X_1$ `EXOR` X_2. Such distributions exhibit strict regularities. Yet, pseudo-independent models fit in the reductionist approach of causal models. We still can try to find out which variables are the causes and which the effects. Possibly, the model $X_1 \rightarrow X_3 \leftarrow X_2$ generates a pseudo-independent distribution if only the knowledge of X_1 and X_2 together says something about X_3. Note that this is examined by case (a) in the proof of theorem 4.4.

Counterexample 7. Consider the coder-decoder example, taken from Spirtes et al. [1993, Fig. 3.23], shown in Fig. 4.8. Variable Y encodes the values of both R and X, and Z decodes Y to match the value of X. This is possible because it is the first bit of Y that corresponds to the value of X. The model is not faithful. Z is independent from R, but connected by an unblocked path in the model. Also, the v-structure $X \rightarrow Y \leftarrow R$ suggests that X and R are dependent when conditioned on Y, but this is not. The coder-decoder system is designed to exhibit the specific behavior that Z equals to X. The CPD components are part of a greater mechanism

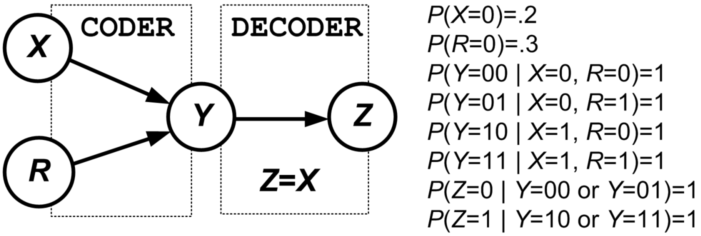


Figure 4.8: Coder-decoder system, taken from [Spirtes et al., 1993, Fig. 3.23], in which Z equals to X.

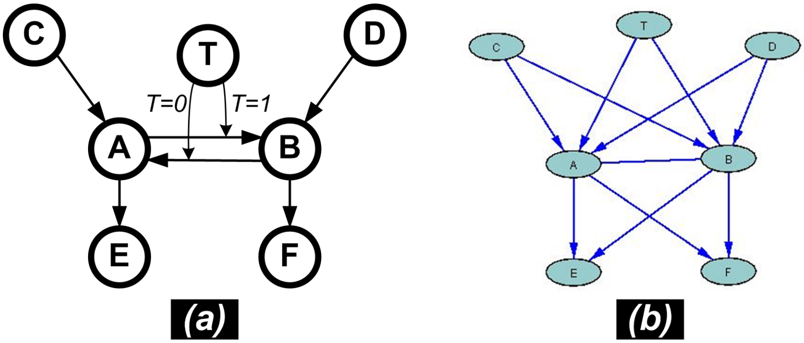


Figure 4.9: System with contextual relation (a) and the learned partially-directed acyclic graph (b).

and are engineered to match each other in such a way that the desired functionality is realized.

Counterexample 9. Another problem for causal inference algorithms is a *mixture of populations*, in which causal connections are in opposite directions for subpopulations. Some people drink too much because they are depressed, while other people are depressed because they drink too much. A mixture of units in which the causal connections go in opposite directions does not in general satisfy the Markov Condition [Scheines et al., 1996, Ch. 2]. Such systems can be regarded as containing *contextual causal relations.* This happens if a causal relation is only valid in a certain *context.* Consider the model depicted in Fig. 4.9(a). Discrete variable T with domain $\{0, 1\}$ sets the context for the causal relation between A and B. A is a cause of B if $T = 1$, while it is an effect of B if $T = 0$. Contextual causal relations were studied by Borms [2006]. The application of the PC algorithm on a sample of this system gives a too complex result. The model learned, Fig. 4.9(b), is a partially-directed acyclic graph. The relation between A and

B is left unoriented since the independencies provide no information to decide upon the orientation. The contextual relation also makes that other causes of A or B do not form a v-structure with respectively B or A. Such a v-structure would determine the orientation of the relation $A - B$. The partially directed acyclic graph represents the Markov-equivalence class containing all directed graphs that cannot be distinguished by the conditional independencies. It consists of two graphs, one with $A \rightarrow B$ and one with $A \leftarrow B$. But none of both is faithful. C is d-connected with E conditional on A (via unblocked path $C \rightarrow B \rightarrow E$): $C \not\perp\!\!\!\perp E \mid A$. But independency $C \perp\!\!\!\perp E \mid A$ holds. This follows from Fig. 4.9(a), the independency holds in both contexts. There is no faithful model and the model also does not correspond to the real causal mechanisms. Contextual relations cannot be represented by standard causal models. The question arises how they can be recognized.

4.4 Faithfulness

Faithfulness of a causal model is put forward as a cornerstone of causal model theory and the accompanying learning algorithms. Its importance is often misunderstood and its validity criticized. Based on the above analysis, we can conclude the following about faithfulness:

1. Faithfulness holds when the system is indeed composed of independent and unrelated mechanisms. The probability of an exact correspondence of probabilities so that causally-related variables become independent is quite small, as expressed by the stability property used by Pearl (see start of Chapter). The typical elements of the set of all Bayesian networks built from a given DAG are faithful (Theorem 4.4).

2. Conditional independencies are the basis for causal inference: we do not consider them as coincidences, but as meaningful information that tell us something about the system under study. Unfaithfulness gives an indication that other mechanisms might be at play.

3. Pearl hypothesizes that there is no unbounded set of conditions on a distribution that would ensure the existence of a faithful graph [Pearl, 1988, p. 131]. Indeed, as shown by theorem 4.4, 'unexpected' conditional independencies can always appear. Every dependence can be turned into an independence by a balanced parametrization of some CPDs.

4. Faithfulness ensures the capacity of deriving answers to qualitative questions from the graph only. This was one of the goals that Pearl put forward while building up causal model theory [Pearl, 1988, p. 79] (Section 3.1.2). All Bayesian networks representing the same joint distribution (built starting from different variable orderings) can be used for predicting unknown variables quantitatively. But they do not describe the same independencies. They are not suitable for *qualitative reasoning*: for answering questions about conditional independencies, such as 'does A affect B when C is known'. A non-faithful model can only answer these questions correctly by quantitative calculation of the probabilities. The model of Fig. 4.3 suggest that B, C and E should be known to have maximal information about D. But from the graph of Fig. 4.2 we know that B has no additional information about D once we know C and E. Qualitative reasoning based on the model only demands that the model contains all meaningful information, that all qualitative properties can be inferred from it without needing precise quantitative information. This is exactly what the faithfulness property stands for.

In the light of meaningful information, faithfulness can be interpreted in a broader sense:

> Faithfulness means that a model explicitly captures all regularities of the data.

Consider a Bayesian network that is not based on the minimal factorization, such as the one depicted in Fig. 4.3. Its DAG is unfaithful and can not explain some of the conditional independencies. The DAG together with the CPDs incorporates all independencies, as they can be calculated from the joint distribution, but not from the meaningful information, the DAG, only. Following the principles according to which modeling should be based on the regularities, faithfulness motivates the ability to learn causal models from observations:

> If a unique, minimal model has the power to foresee all qualitative consequences, it must come close to reality.

4.5 Summary of Chapter

This chapter addressed the two major criticisms towards causal model theory and causal inference - the validity of faithfulness and the causal interpretation of the models - by interpreting them with the principles of *minimality*, *regularity extraction* and *meaningful information*.

The conditional independencies are the regularities at hand that allow compression of a distribution and the learning of the minimal model from the data. Minimality relies on the incompressibility of the minimal Bayesian network. The DAG of the minimal factorization is the Kolmogorov Minimal Sufficient Statistic (KMSS) and is faithful if the CPDs are random and unrelated. Non-minimality implies the presence of non-modeled regularities. Regularities in an individual conditional probability distribution (CPD) or among several CPDs can lead to conditional independencies not expected by the Markov condition; independencies that do not follow from the system's causal structure. Causal model theory thus describes what can 'normally' be expected from a causal structure, without taking into consideration the precise parameterization of the CPDs. The algorithms for causal inference rely on the assumption that no other independencies arise.

I argued that the causal component of causal model theory corresponds to the decomposition of the model into unique and independent components, the CPDs. The *d*-separation criterion can be expressed by a distributed algorithm that runs independently on all components, which only communicate through the variables. What causal models tell us about reality is how the system is composed of independent mechanisms. Each mechanism is responsible for setting the state of one variable. Decomposition corresponds to a reductionist approach: we have chopped up the system into simpler components, which constitute the 'atomic' elements of the system. They give the canonical decomposition of the system.

The validity of causal inference can then be analyzed by the validity of faithfulness, the absence of non-modeled regularities and the correctness of the decomposition. Counterexamples of causal model theory were analyzed by these three statements. The analysis showed that the invalidity of the decomposition could be related to the presence of non-modeled regularities or non-Markovian independencies. Concluding, in case of faithfulness and absence of additional regularities, a causal model offers a plausible hypothesis about reality.

This chapter concludes the philosophical discussion about inductive and causal inference. The next chapter will clear the way for introducing causal inference into the world of performance analysis.

Chapter 5

Information Equivalence

THE theory of information equivalence was developed to overcome limitations of the current causal learning algorithms due to the violation of faithfulness by the presence of *deterministic relations*. Performance data contain a lot of deterministic relations. The PC algorithm fails when they appear in the data. Consider the model of Fig. 5.1. It indicates how the *datatype* of the main data structure (integer, floating point, double-precision, ...) used in an algorithm affects the cache misses when executed. The model shows that it is in essence the size of the datatype which determines the cache misses and not its specific type. The relation between *datatype* and *data size* is however a function: $data\ size = f(datatype)$. A variable that is determined by some other variables is depicted with double-bordered circles, as *data size* in Fig. 5.1. The model implies, by the Markov property, that

$$datatype \perp\!\!\!\perp cache\,misses \mid data\ size. \tag{5.1}$$

But from the function it also follows that

$$data\ size \perp\!\!\!\perp cache\,misses \mid datatype, \tag{5.2}$$

for, by the functional relation, variable *datatype* contains all information about variable *data size*. *datatype* contains the same information about the *cachemisses*. However, both independencies cannot be represented by a faithful graph and pose problems for constraint-based algorithms. The

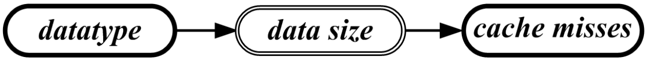


Figure 5.1: Example causal model.

adjacency step of the PC algorithm (see Sec. 3.4.1) fails since the two independencies result in the removal of edges '*data size – cache misses*' and '*datatype – cache misses*'. The algorithm relies on the property that directly related variables do not become independent by conditioning on any set of other variables. By testing this property on two dependent variables, the algorithm discriminates directly from indirectly related variables. This does not hold in the presence of deterministic variables. The current approach to overcome this problem is to remove deterministically related variables from the data [Scheines et al., 1996]. I will, however, argue that deterministically related variables contain valuable information and should not be eliminated from the data.

The problem of deterministic variables is characterized by a violation of the **intersection condition**, one of the necessary conditions for faithfulness [Pearl, 1988]:

$$\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Z} \mid \boldsymbol{W}, \boldsymbol{Y} \quad \& \quad \boldsymbol{Y} \perp\!\!\!\perp \boldsymbol{Z} \mid \boldsymbol{W}, \boldsymbol{X} \quad \Rightarrow \quad \boldsymbol{X}, \boldsymbol{Y} \perp\!\!\!\perp \boldsymbol{Z} \mid \boldsymbol{W} \tag{5.3}$$

The condition states that if two variables render the other irrelevant with respect to a third variable, neither of both can depend on that variable. This condition is violated when two variables contain the same information about a third variable, Z. They are called *information equivalent* with respect to Z. Although both are marginally dependent on Z, either becomes conditionally independent from Z by conditioning on the other variable. Hence, the intersection condition is broken.

If X and Y are information equivalent for Z, they contain the same information about Z. This must be incorporated by the model. It is not clear which of the two should be connected to Z. Connecting both to Z would represent redundant information and in this manner disrupt the minimality condition. Connecting one suffices. I propose to connect the one having the simplest relation with Z. This choice enables the construction of useful models. The purpose is to reestablish the faithfulness of Bayesian networks as representation of conditional independencies by characterizing information equivalences and integrating them into an augmented model. This results in an augmented model that correctly describes the qualitative properties of the system under study, as expressed by the faithfulness property. These models can be learnt from observational data by the extended PC algorithm.

The first section gives an overview of related work. In the following section information equivalences are defined. It is shown that they appear when a distribution results in an equivalent partitioning. Some useful properties of information equivalences are proved. Augmented Bayesian networks, incorporating the knowledge about deterministic relations and basic information equivalences, are defined in Section 3. Section 4 discusses how minimality can lead to a selection criterion for information equivalent relations and section 5 shows how faithfulness can be reestablished for the augmented models. Finally, section 6 extends the PC algorithm so that it can learn augmented models and section 7 reports on experimental verification of the extended learning algorithm.

5.1 Related Work

Recent research has developed methods for performing inferences in Bayesian networks with functional dependencies [Cowell et al., 1999, Cobb and Shenoy, 2004]. Dechter and Mateescu introduced mixed networks to express probabilistic and deterministic information in the form of constraints [Dechter and Mateescu, 2004], whereas I view deterministic relations as proper causal relations. Geiger [1990] and Spirtes et al. [1993] extended the d-separation criterion for retrieving the dependencies entailed by deterministic relations, which they called *D-separation*.

Current constraint-based learning algorithms fail for data containing functionally determined variables [Spirtes et al., 1993, p. 57], they require that such variables are eliminated from the input data [Scheines et al., 1996]. The argument is that such variables are not essential to the model since they contain redundant information. In section 5.2.1 I show, however, that such variables provide insight in the underlying mechanisms and often reduce the complexity of the model. Moreover, determinism is not always known a priori.

For the faithfulness of graphical models, many conditions should hold [Pearl, 1988]. Therefore, other representation schemes of independency information were developed, such as the imsets of Studeny [2001], which can model any conditional independence structure. My approach claims that if violations of faithfulness come from local properties, these properties should be integrated into the causal modeling framework.

5.2 Information equivalences

A necessary condition for the existence of a faithful graph is the intersection condition. This condition is violated by information equivalences.

5.2.1 Definition

Definition 5.1 (information equivalence)**.** ***X*** *and* ***Y*** *are called* ***information equivalent*** *with respect to Z, the reference variable, if*

- $\boldsymbol{X} \not\perp\!\!\!\perp Z$ *and* $\boldsymbol{Y} \not\perp\!\!\!\perp Z$
- $\boldsymbol{Y} \perp\!\!\!\perp Z \mid \boldsymbol{X}$
- $\boldsymbol{X} \perp\!\!\!\perp Z \mid \boldsymbol{Y}$

Note that single stochastic variables are denoted by capital letters, sets of variables by boldface capital letters. Knowledge of either $\boldsymbol{X}$ or $\boldsymbol{Y}$ is completely equivalent from the viewpoint of Z. It follows that $I(Y;Z) = I(X;Z)$, hence X and Y contain the same *amount* and the same *kind* of information about Z.

A variable Y is **determined** by a set of variables $\boldsymbol{X}$ if the relation between Y and the set $\boldsymbol{X}$ is a function, written as $Y = f(\boldsymbol{X})$. A functional relation implies that the variables $\boldsymbol{X}$ contain all information about Y. If Y is correlated to a third variable Z, all information from Y about Z is also present in $\boldsymbol{X}$. If additionally $\boldsymbol{X} \perp\!\!\!\perp Z \mid Y$ holds, meaning that $\boldsymbol{X}$ contains no additional information about Z, then Y and $\boldsymbol{X}$ are information equivalent with respect to Z. For a bijection, when $Y = f(X)$ and $X = f^{-1}(Y)$ are functions, *each* variable dependent on X or Y implies an information equivalence.

If, besides the independencies of Definition 5.1, additionally $X \perp\!\!\!\perp Y \mid Z$ holds, then all three variables contain the same information about each other and are information equivalent. I call this a **multi-node equivalence**.

Example 5.2 (Causal Performance Model).

Fig.5.2 represents a causal model of performance related data of a quicksort algorithm. The overall performance is measured by the computation time (T_{comp}). $\#op$ represents the number of basic compare-swap operations[1], which is affected by the *array size*. The computation time depends on $\#op$, the number of processor cycles for one operation (C_{op}) and the processor's clock frequency (f_{clock}). The number of cycles consists of C_{instr}, the cycles spent executing the instructions of one operation ($\#instr_{op}$), and C_{mem}, the cycles spent waiting due

[1] Recall that the quicksort algorithm mainly consists of comparisons of elements, alternated with swapping elements.

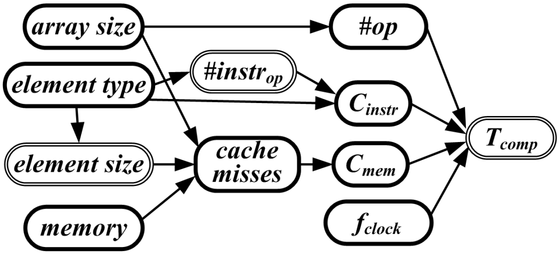


Figure 5.2: Causal model of the performance of the quicksort algorithm.

to memory accesses, which are triggered by the *cache misses*. These are determined by the *memory* capacity, the *array size* and *element size*, which is the size in bytes of the elements of the array. Finally, the data type of the elements (variable *data type*) affects C_{op}, and determines $\#instr_{op}$ and the *element size*. The causal interpretation of the edges should be read as "a change of the state of A causes a change of the state of B".

Deterministic variables are depicted with double-bordered circles, they are determined by their parents in the graph. Even the other non-input variables are quasi-determined by their parents, since a serial computer is deterministic. One can argue that these deterministic variables can be omitted and that the computation time can directly be expressed in function of the parameters. The intermediate variables, however, are of great explanatory importance and extend the predictive power of the model. The cache misses, for example, are only determined by the size of the data, not by the data type. This knowledge enables the prediction of the cache misses for new data types. Moreover, *element size* is a countable variable, whose domain is an ordered set. The relation with the cache misses can be expressed by a continuous function, which makes it possible to predict the cache misses for yet unknown sizes. The relation of the discrete variable *element type* and the *cache misses* is a table without predictive capacities.

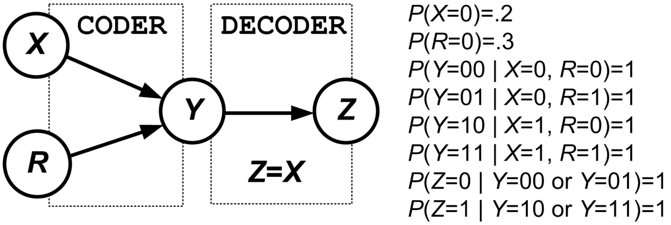


Figure 5.3: Causal model of SGS (Fig. 3.23) in which Z equals X

Example 5.3 (Coder-decoder).

Consider the coder-decoder example, taken from [Spirtes et al., 1993, Fig. 3.23], shown in Fig. 5.3. Variable Y encodes the values of both R and X, and Z decodes Y to match the value of X. This is possible because it is the first bit of Y that corresponds to the value of X. X is therefore deterministically related to Z, though not adjacent in the graph. Both X and Y are information equivalent with respect to Z.

5.2.2 Equivalent Partition

Information equivalences follow from a broader class of relations than just deterministic ones. Therefore I have to introduce the notion of equivalent partition.

If two variables X and Z are dependent, implied by $P(Z \mid X) \neq P(Z)$, the conditional distribution of one variable differs for at least two values of the conditioning variable:

$$\exists\ x_1, x_2 \in X_{dom} : P(Z \mid x_1) \neq P(Z \mid x_2). \tag{5.4}$$

The information a variable contains about another lies into the differences in the conditional distributions. Values for which this distribution is the same contain the same information.

Definition 5.4. *The domain of X, denoted by X_{dom}, can be partitioned into disjoint subsets X^k_{dom} for which $P(Z \mid x)$ is the same for all $x \in X^k_{dom}$. We call this the $Z-partition$ of X_{dom}. We define $\kappa_Z(X)$ as the index of the subset.*

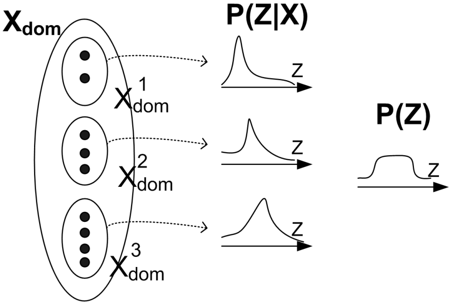


Figure 5.4: Z-partition of the domain of X

Accordingly, the conditional distribution depends solely on the index of the Z-partition:

$$P(Z \mid X) = P(Z \mid \kappa_Z(X)) \tag{5.5}$$

Fig. 5.4 shows the Z-partition of Z_{dom} and the related conditional distributions of Z. $P(Z)$ is also shown, it is the weighted sum of the conditional distributions: $P(Z) = \sum_x P(Z \mid x).P(x)$.

A **binary relation** between two variables associates elements of the domain of X with that of Y. This can be shown graphically by representing both domains with a two sets and connect the elements that are related by the relation. For the **Cartesian product** of X and Y, $X \times Y$, each element of X_{dom} is associated with each element of Y_{dom}.

Definition 5.5. *A relation $\Re \subset X \times Y$ defines an* ***equivalent partition*** *in Y_{dom} to a partition of X_{dom} if:*

1. *$\forall x_1$ and $x_2 \in X_{dom}$ that do not belong to the same partition: $\forall y_1 \in Y_{dom}$ with $x_1 \Re y_1$, it must be that $\neg(x_2 \Re y_1)$.*

2. *For all subsets X^k_{dom} of the partition: $\exists x_1 \in X^k_{dom}, \exists y_1 \in Y_{dom}$: $x_1 \Re y_1$.*

For an equivalent partition, every partition X^k_{dom} corresponds to a partition Y^l_{dom}. If an element of Y_{dom} is related to an element of a partition of X_{dom}, it is not related to an element of another partition, and each partition of X_{dom} has at least one element that is related to a partition of Y_{dom}.

Fig. 5.5 shows an example of an equivalent partition. No y is related to X-values belonging to different partitions and for every partition, there is

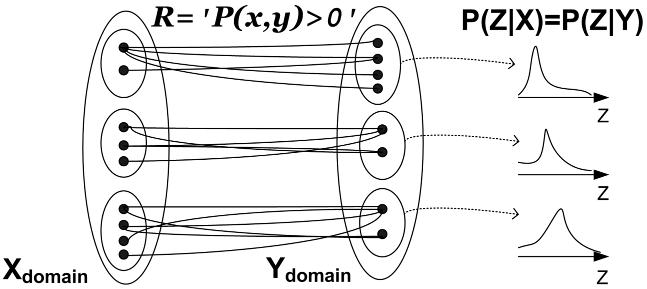


Figure 5.5: Variables X and Y are information equivalent for Z. $P(x, y)$ is only strictly positive for values that affect $P(Z)$ similarly. These values are related by relation R.

at least one related Y-value. Note that a function, for which every x-value has a related Y-value, defines an equivalent partition on Y_{dom} for every partition of X.

This concept will be used to exactly define when the intersection condition is violated. The relation $\Re$ defined by $P(x, y) > 0$ is considered. Two values $x \in X_{dom}$ and $y \in Y_{dom}$ are related by $\Re$ if there is a non-zero probability that both appear together.

Recall that independence of $\boldsymbol{U}$ and $\boldsymbol{W}$ conditional on $\boldsymbol{V}$ ($\boldsymbol{U} \perp\!\!\!\perp \boldsymbol{W} \mid \boldsymbol{V}$) is defined by

$$P(\boldsymbol{U} \mid \boldsymbol{v}, \boldsymbol{w}) = P(\boldsymbol{U} \mid \boldsymbol{v}) \;\; \textit{whenever} \;\; P(\boldsymbol{v}, \boldsymbol{w}) > 0 \tag{5.6}$$

And that:

$$U \perp\!\!\!\perp W \mid V \Leftrightarrow P(U \mid w) = \sum_{v \in V} P(U \mid v).P(v \mid w) \;\; \text{whenever} \;\; P(v, w) > 0 \tag{5.7}$$

Theorem 5.6. *If $X \not\perp\!\!\!\perp Z$ and $Y \perp\!\!\!\perp Z \mid X$, then*

$X \perp\!\!\!\perp Z \mid Y \Leftrightarrow$ the relation $x \Re y$ defined by $P(x, y) > 0$, with $x \in X_{dom}$ and $y \in Y_{dom}$, defines an equivalent partition in Y_{dom} to the Z-partition of X_{dom}.

Proof $\Leftarrow$
I have to prove that $P(Z \mid Y, X) = P(Z \mid Y)$. The left hand side leads to $P(Z \mid Y, X) = P(Z \mid X) = P(Z \mid \kappa_Z(X))$, with $\kappa_Z(X)$ the index of X in the Z-partition of X_{dom}. I will prove that $P(Z \mid Y)$ leads to the same

expression.
By the independence $Y \perp\!\!\!\perp Z \mid X$, I can write that

$$P(Z \mid Y) = \sum_{x \in X_{dom}} P(Z \mid X = x).P(X = x \mid Y) \tag{5.8}$$

The last factor only differs from zero if both X and Y belong to the subsets that correspond to each other by the equivalent partition. By proper numbering of the subsets, I make the indices $\kappa_Z(X)$ and $\kappa_Z(Y)$ correspond. It follows that

$$= \sum_{x \in X_{dom}:\kappa_Z(x)=\kappa_Z(y)} P(Z \mid X = x).P(X = x \mid Y) \tag{5.9}$$

$$= P(Z \mid \kappa_Z(x)) \sum_{x \in X_{dom}:\kappa_Z(x)=\kappa_Z(y)} P(X = x \mid Y) \tag{5.10}$$

Since the conditional distribution of Z is constant in each subset of the Z-partitioning by definition. The sum is 1, since $P(X = x \mid Y)$ is zero everywhere else.

Proof $\Rightarrow$
I have to show that $\forall x_1, x_2 \in X_{dom}$ for which $P(Z \mid x_1) \neq P(Z \mid x_2)$, there exists a $y_1 \in Y_{dom}$ for which $P(x_1, y_1) > 0$ and that for all such y_1 values $P(x_2, y_1) = 0$.
Since $P(x_1) > 0$, there must be at least one value y_1 for which $P(x_1, y_1) > 0$, otherwise $P(X, Y)$ is not a valid distribution. Next, both given conditional independencies, $Y \perp\!\!\!\perp Z \mid X$ and $X \perp\!\!\!\perp Z \mid Y$, imply that (Eq. 5.6)

$$P(Z \mid x_1, y_1) = P(Z \mid x_1) = P(Z \mid y_1). \tag{5.11}$$

Assume that $P(x_2, y_1) \neq 0$, then likewise

$$P(Z \mid x_2, y_1) = P(Z \mid x_2) = P(Z \mid y_1) \tag{5.12}$$

Combining the right hand sides of both equations leads to the contradiction that $P(Z \mid x_2) = P(Z \mid x_1)$. □

By applying Equation 5.6 on both conditional independencies of equivalence $\boldsymbol{X}$ and $\boldsymbol{Y}$ for Z it follows that

$$P(Z \mid \boldsymbol{x}) = P(Z \mid \boldsymbol{y}) \;\; \textit{whenever} \;\; p(\boldsymbol{x}, \boldsymbol{y}) > 0 \tag{5.13}$$

This summarizes the results of the theorem.

5.2.3 Assumptions

In analyzing the relation between independencies in distribution and the DAG (of a valid BN), I make no assumption about faithfulness. A Bayesian network represents the conditional independencies that follow from the Markov condition. Besides those, other independencies can occur. The only thing we can say about the relation between a Bayesian network and the distribution it represents is that:

> For any graph of a Bayesian network that correctly represents a probability distribution, it follows that
>
> - *d*-separation in graph ⇒ (conditional) independency in distribution.
> - (conditional) dependence in distribution ⇒ there is an unblocked path in graph.

In order to be able to investigate the effect of an information equivalence on the conditional independencies between other variables, I will assume a condition that expresses a kind of transitivity.

Assumption 5.7. *Weak Transitivity*

$$\boldsymbol{T}\perp\!\!\!\perp\boldsymbol{V}\mid\boldsymbol{W} \;\;\&\;\; \boldsymbol{T}\perp\!\!\!\perp\boldsymbol{V}\mid\boldsymbol{W},\,\boldsymbol{U} \;\;\Rightarrow\;\; \boldsymbol{T}\perp\!\!\!\perp\boldsymbol{U}\mid\boldsymbol{W} \;\;\textit{or}\;\; \boldsymbol{U}\perp\!\!\!\perp\boldsymbol{V}\mid\boldsymbol{W} \tag{5.14}$$

Or, put the other way around:

$$\boldsymbol{T}\not\perp\!\!\!\perp\boldsymbol{U}\mid\boldsymbol{W} \;\;\&\;\; \boldsymbol{U}\not\perp\!\!\!\perp\boldsymbol{V}\mid\boldsymbol{W} \;\;\Rightarrow\;\; \boldsymbol{T}\not\perp\!\!\!\perp\boldsymbol{V}\mid\boldsymbol{W} \;\;\textit{or}\;\; \boldsymbol{T}\not\perp\!\!\!\perp\boldsymbol{V}\mid\boldsymbol{W},\,\boldsymbol{U} \tag{5.15}$$

It is one of the necessary conditions for the existence of a faithful graph [Pearl, 1988]. Eq. 5.15 says that if T depends on U and U depends on V, it implies that either T depends on V (e.g. as in model $T \rightarrow U \rightarrow V$) or becomes dependent by conditioning on U (e.g. as by v-structure $T \rightarrow U \leftarrow V$)

Take again the coder-decoder example shown in Fig. 5.3. X determines the first bit of Y and R the second. The decoding of Z is determined by the first bit of Y. This model, however, violates the weak transitivity condition. Y depends on X, R and Z; but R is independent from X and Z, also after conditioning on Y. The values of variable Y reflect two separate quantities, one that is determined by X and one by R. Each value

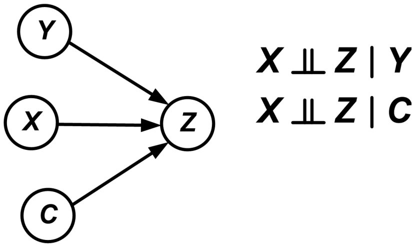


Figure 5.6: Bayesian network which is a counter example of Assumption 5.8.

of Y combines both quantities. The coder-decoder system is designed to exhibit this specific behavior.

The following assumptions are introduced to simplify the discussion. They exclude some exotic cases in which the probability distributions exhibit very specific regularities. These assumptions hold for the experiments presented in section 5.7.

Assumption 5.8.

$$\boldsymbol{X \not\perp\!\!\!\perp Z} \;\;\&\;\; \boldsymbol{X \perp\!\!\!\perp Z \mid Y} \;\;\&\;\; \boldsymbol{X \perp\!\!\!\perp Z \mid C} \;\;\Rightarrow\;\; \boldsymbol{X \perp\!\!\!\perp Z \mid Y,\ C} \tag{5.16}$$

Example 5.9 (Counterexample of Assumption 5.8).

The construction of a counterexample of Assumption 5.8 shows that the violation of the assumption only happens when the probabilities of the CPDs of the Bayesian network give an exact match. $X \not\perp\!\!\!\perp Z$ implies that X and Z are related in the graph, $X \not\perp\!\!\!\perp Z \mid Y, C$ implies that there is a path between X and Z that is not blocked by Y and/or C. On the other hand, independencies $X \perp\!\!\!\perp Z \mid Y$ and $X \perp\!\!\!\perp Z \mid C$ mean that the correlation of X and Z is neutralized when conditioned on Y or C separately.

Fig. 5.6 shows a Bayesian network which can lead to the violation of the assumption by a well-balanced parameterization of the CPDs. The network corresponds to factorization $P(Z) = P(Z \mid C, X, Y).P(C).P(X).P(Y)$. For $X \perp\!\!\!\perp Z \mid Y$ we get the following conditions on the distribution:

$$\forall\ x, y, z : P(z \mid x, y) = P(z \mid y) \tag{5.17}$$

$$\Leftrightarrow \forall\ x, y, z : \sum_c P(z \mid x, y, c).P(c) = \sum_c \sum_x P(z \mid x, y, c).P(c).P(x) \tag{5.18}$$

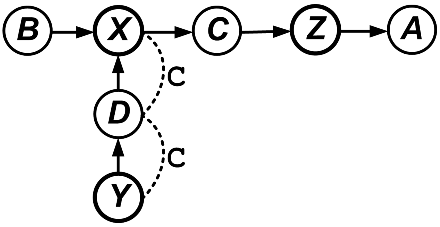


Figure 5.7: Model with X and Y information equivalent for Z and some additional nodes.

The left-hand side and the right-hand side of Eq. 5.17 lead to respectively the left-hand side and the right-hand side of Eq. 5.17. Similarly, independence $X \perp\!\!\!\perp Z \mid C$ imposes an additional set of conditions on the distribution:

$$\forall c, x, z : \sum_y P(z \mid c, x, y).P(y) = \sum_x \sum_y P(z \mid c, x, y).P(x).P(y) \tag{5.19}$$

Assumption 5.10. *If X and Y are information equivalent with respect to a variable Z, it follows that*

$$X \perp\!\!\!\perp Y \mid D \Rightarrow X, Y \perp\!\!\!\perp Z \mid D \tag{5.20}$$

If D contains all information shared by X and Y, D also contains the information that X and Y have about Z. A counterexample is given later.

5.2.4 Properties of Information Equivalences

The following properties prove that information equivalences can be reduced to a set of fundamental equivalences. An example model is depicted in Fig. 5.7. It provides an example for each of the properties. The variables that appear in the properties correspond to those of the model. The first property shows that an information equivalence remains for all variables related to the information equivalent variables via the reference variable.

Property 5.11. *If X and Y are information equivalent with respect to a variable Z and for variable A it holds that $A \not\perp\!\!\!\perp Z$ and Z screens off X and Y from A ($X \perp\!\!\!\perp A \mid Z$ and $Y \perp\!\!\!\perp A \mid Z$), then X and Y are information equivalent with respect to A.*

Proof. From $X \perp\!\!\!\perp A \mid Z$ it follows that (using Eq. 5.7)

$$\begin{aligned} P(A \mid X) &= \sum_{z \in Z} P(A \mid z).P(z \mid X) \\ &= \sum_{z \in Z} P(A \mid z).P(z \mid Y) = P(A \mid Y) \end{aligned} \tag{5.21}$$

The last step is true by $Y \perp\!\!\!\perp A \mid Z$. □

The next property says something about the conditional independencies implied by information equivalences. If among information equivalent variables, one contains all information that a variable has about the reference variable, the other equivalent variables also contain this information.

Property 5.12. *If X and Y are information equivalent with respect to a variable Z, it follows that*

$$Z \perp\!\!\!\perp B \mid X \Leftrightarrow Z \perp\!\!\!\perp B \mid Y \tag{5.22}$$

Proof. Let the domain of X be partitioned into subsets X^k_{dom} so that $P(Z \mid x)$ is the same in each subset, namely $P(Z \mid k)$, but different for elements of different subsets. There are at least two such subsets, since $P(Z \mid x) \neq P(Z)$.

$$P(Z \mid B) = \sum_{x \in X_{dom}} P(Z \mid x).P(x \mid B) = \sum_k \sum_{x \in X^k_{dom}} P(Z \mid x).P(x \mid B) \tag{5.23}$$

$$= \sum_k P(Z \mid k) \sum_{x \in X^k_{dom}} P(x \mid B) \tag{5.24}$$

$$= \sum_k P(Z \mid k) \sum_{x \in X^k_{dom}} \sum_{y \in Y_{dom}} P(x \mid y, B).P(y \mid B) \tag{5.25}$$

Each subset X^k_{dom} maps to a subset Y^l_{dom} for which $P(Z \mid k) = P(Z \mid l)$. By Eq. 5.13, $P(x \mid y, B)$ is only positive whenever $x \in X^k_{dom}$ and $y \in Y^l_{dom}$, thus:

$$P(Z \mid B) = \sum_k P(Z \mid k) \sum_{y \in Y^l_{dom}} P(y \mid B) \sum_{x \in X^k_{dom}} P(x \mid y, B) \tag{5.26}$$

The equivalence also implies that

$$\sum_{x \in X_{dom}^{k}} P(x \mid y, B) = 1 \text{ if } y \in Y_{dom}^{l}, \tag{5.27}$$

hence:

$$P(Z \mid B) = \sum_{l} P(Z \mid l) \sum_{y \in Y_{dom}^{l}} P(y \mid B) \tag{5.28}$$

$$= \sum_{y \in Y_{dom}} P(Z \mid y).P(y \mid B) \quad \Leftrightarrow \quad Z \perp\!\!\!\perp B \mid Y \tag{5.29}$$

□

The following property proves that an information equivalence can be reduced to another if there is a variable which makes the reference variable independent from the information equivalent variables.

Property 5.13. *If X and Y are information equivalent with respect to a variable Z and assumptions (5.7), (5.8) and (5.10) hold, it follows that*

$$X \perp\!\!\!\perp Z \mid C \Leftrightarrow Y \perp\!\!\!\perp Z \mid C \tag{5.30}$$

If, together with Eq. 5.30, additionally $C \not\perp\!\!\!\perp Z \mid X$ holds, then X and Y are also information equivalent for C. Otherwise ($C \perp\!\!\!\perp Z \mid X$), C is together with X and Y information equivalent for Z.

Proof. By assumption 5.8, $X \perp\!\!\!\perp Z \mid C, Y$ holds. Weak transitivity (Eq. 5.14) then demands that $X \perp\!\!\!\perp Y \mid C$ or $Y \perp\!\!\!\perp Z \mid C$ holds. This proves the second independency, because if the first is true, the second follows from assumption 5.10. Then, by assumption 5.8 again, $Y \perp\!\!\!\perp Z \mid C, X$ holds, which means that the information equivalence remains under conditioning on C.

For proving the equivalence, I have to show that (a) $Y \perp\!\!\!\perp C \mid X$ and (b) $X \perp\!\!\!\perp C \mid Y$. From $Y \perp\!\!\!\perp Z \mid X$ and $Y \perp\!\!\!\perp Z \mid C, X$; weak transitivity demands that $Y \perp\!\!\!\perp C \mid X$ or $C \perp\!\!\!\perp Z \mid X$. The second independence is false, which proves the first independence and thus (a). Independence (b) is proved with the same arguments. $C \not\perp\!\!\!\perp Z \mid Y$ holds, because $C \perp\!\!\!\perp Z \mid Y$ would mean that C and Y are information equivalent with respect to Z. But then, by transitivity of information equivalences (follows directly from Eq. 5.13), C and X would be information equivalent for Z, contradicting the given $C \not\perp\!\!\!\perp Z \mid X$.

Finally, if $C \perp\!\!\!\perp Z \mid X$, then - by transitivity - X, C and Y are equivalent for Z. □

Example 5.14 (Information Equivalence Under Conditioning).

> Information equivalences do not always hold under conditioning. In the causal performance model of Fig. 5.2, *elementType* and *elementSize* are equivalent for the processor cycles spent on memory access, C_{mem}. But after conditioning on C_{op}, this equivalence is not true any more. Then, a path between *elementType* and C_{mem} is unblocked and thus: $elementType \not\perp\!\!\!\perp C_{mem} \mid elementSize, C_{op}$. The knowledge of *elementType* enables the prediction of the cycles spent on computation, C_instr. Together with the total number of cycles (C_{op}), an estimation of the cycles lost due to cache misses can be made. Thus, *elementType* gives more information on C_{mem} then the knowledge of *elementSize* alone.

An information equivalence is called a **basic information equivalence** if no variable exists that contains more information about the reference variable than the equivalent variables.

Property 5.15. *Under assumptions (5.7), (5.8) and (5.10), basic information equivalent variables are directly related and at least one of the variables is directly related to the reference variable of a basic information equivalence.*

Proof. Take X and Y information equivalent for Z. X and Y are dependent, so there is a path connecting both. If there is another variable on the path, for example D, making X and Y independent: $X \perp\!\!\!\perp Y \mid D$. By assumption 5.10, $X \perp\!\!\!\perp Z \mid D$ follows. Then, by property 5.13, either D is also information equivalent for Z (when $D \perp\!\!\!\perp Z \mid X$) or X and Y are information equivalent with respect to D (when $D \not\perp\!\!\!\perp Z \mid X$). With the given $X \perp\!\!\!\perp Y \mid D$, the three variables form a multi-node equivalence.

By the dependency of the equivalent variables with the reference variable, there must be a path connecting them. If this path is blocked by a variable C, by property 5.13, variable C is also equivalent for Z; or X and Y form a basic information equivalence for C, while X and Y do not for Z. □

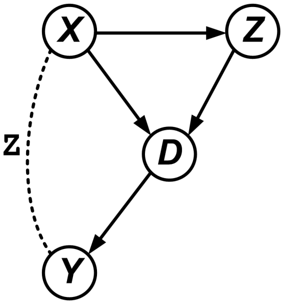


Figure 5.8: Bayesian network which is a counter example of Property 5.15.

Example 5.16 (Counterexample of Property 5.15).

Assumption 5.10 was introduced to exclude a special case in which the last property about adjacency does not hold. Consider the equivalence of X and Y for Z and independence $X \perp\!\!\!\perp Y \mid D$. Then it is possible that:

- D is not information equivalent together with X and Y for Z: this means that $Z \not\perp\!\!\!\perp D \mid X$ or $Z \not\perp\!\!\!\perp D \mid Y$.
- X and Y are not information equivalent for D: this means that $X \not\perp\!\!\!\perp D \mid Y$ or $Y \not\perp\!\!\!\perp D \mid X$.

Fig. 5.8 shows a Bayesian network for which this is possible, for example when $Z \not\perp\!\!\!\perp D \mid Y$ and $X \not\perp\!\!\!\perp D \mid Y$. This requires a very specific parameterization of the probabilities. D has exclusive information about X and about Z. On top of that, D participates in the equivalent partition of X and Y. The relation defined by $P(D, X) > 0$ defines an equivalent partition in D_{dom} to the Z-partition of X_{dom}. Assumption 5.10 excludes this particular case.

5.3 Augmented Bayesian Network

The previous section showed that the basic information equivalences suffice to augment the model. Other equivalences can easily be derived from it. Deterministically related variables, however, possibly generate multiple equivalences. Since for $Y = f(\boldsymbol{X})$, $\boldsymbol{X}$ is equivalent for all variables related

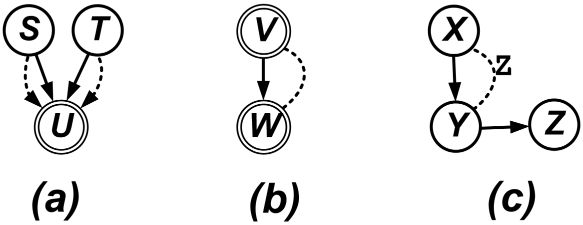


Figure 5.9: Augmented Causal Model with a Functional Relation (a), a Bijection (b) and an Information Equivalence (c).

to Y. A deterministic relation is thus more fundamental and is added to the model instead of all equivalences that follow from it.

Definition 5.17. *An information equivalence augmented Bayesian network consists of a DAG G over variables* $\boldsymbol{V}$*, the conditional probability distributions* $P(V_i \mid parents(V_i))$*, the deterministic relations Deterministic(*$\boldsymbol{V}$*) and the information equivalences Equivalences(*$\boldsymbol{V}$*). Deterministic(*$\boldsymbol{V}$*) is the set of ordered tuples of variables in* $\boldsymbol{V}$*, where for each tuple* $\langle V_1, \ldots, V_n \rangle$*,* V_n *is a deterministic function of* $V_1, \ldots, V_{n-1}$ *and is not a deterministic function of any subset of* $V_1, \ldots, V_{n-1}$*. Equivalences(*$\boldsymbol{V}$*) is the set of ordered tuples of sets of variables in* $\boldsymbol{V}$*, where for each tuple* $\langle \boldsymbol{W}_1, \ldots, \boldsymbol{W}_n \rangle$*, the sets* $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_{n-1}$ *are information equivalent with respect to* $\boldsymbol{W}_n$*.*

I propose the following notation. Deterministic nodes are depicted with double-bordered circles with dashed edges coming from the determining variables, as shown in Fig.5.9 (a). If the parents comprise all the determining variables, the dashed edges may be omitted. Two variables related by a bijection are linked with an unoriented dashed edge (Fig.5.9 (b)). Information equivalent variables are connected by a dashed edge annotated with the reference variable (Fig.5.9 (c)). I do not provide a notation for equivalences of sets of variables, this would require hyper-edges. Such equivalences can be added as text to the graph. Besides, they rarely occur in practice.

5.4 The Complexity Criterion

Modeling is based on the minimality criterion, i.e. we should seek, in spirit of Occam's Razor, the simplest model that is able to describe the data. In the context of causal models, dependent variables are connected with an edge, whereas variables that become independent when conditioned on others are not directly related. For a basic information equivalence, X and

Y for Z, there is no other variable that makes X and Y independent from Z. This implies that they should be related. This was proved by Property 5.15. It is therefore reasonable to call the relations $X - Z$ and $Y - Z$ **information equivalent relations**. From the viewpoint of information, both relations are equivalent.

For a faithful representation, the two independencies, $Y \perp\!\!\!\perp Z \mid X$ and $X \perp\!\!\!\perp Z \mid Y$, suggest that neither X or Y is adjacent to Z. On the other hand, including both edges would disrupt the minimality condition, since both variables have the same information about the reference variable. Relating one of both with Z suffices to model the information they contain about Z. This section will propose a selection criterion.

5.4.1 Complexity of Relations

The relations of information equivalent variables with the reference variable represent the same 'information transfer'. We therefore need criteria, different from conditional independencies, to select among information equivalent relations. In absence of background knowledge, the only objective criterion is the *complexity* of the relations, according to which simpler relations should be preferred over complex ones. The choice between two equivalent variables X and Y for being adjacent to the reference variable Z is decided upon which relation, $Z - X$ or $Z - Y$, is the simplest.

Shannon's mutual information, defined by the decrease in entropy (uncertainty) of a variable due to knowledge of another, measures the information one variable conveys about the other. But it does not take the complexity of the relation into account. Therefore I will rely on the analogous **algorithmic mutual information**, denoted as $I_A(x : y)$, defined as the decrease in Kolmogorov complexity [Grünwald and Vitányi, 2003]:

$$I_A(x : y) = K(x) - K(x \mid y) \tag{5.31}$$

It reflects the additional compression of x thanks to the knowledge of y, where simpler relations lead to higher values. This measure is proved to be symmetric [Grünwald and Vitányi, 2003]. The complexity of an individual object x is measured by its Kolmogorov complexity $K(x)$, defined as the length of the shortest program that prints the object and then halts. The conditional Kolmogorov complexity $K(x \mid y)$ of x given y is the length of the shortest program that given y as input prints x and then halts. The complexity of relation $X - Y$ can then be quantified by estimating $I_A(x^n : y^n)$, where x^n and y^n are the vectors of the observed data with sample size n.

If the complexities of the relations match, we have to decide upon other criteria. I will then connect the reference variable to the variable(s) which is/are cause(s) of the other equivalent variable(s). If, for example, X and Y are linearly and deterministically related, the relation of X and Y with *any other variable* will be completely similar, qualitatively and quantitatively. They contain equivalent information about any other variable, so - in the absence of background knowledge - they represent equivalent quantities. Both variables are indistinguishable from the perspective of the system under study. Handling information equivalences thus only makes sense when non-linear relations appear in the data.

For a relation among continuous variables, a regression analysis is used for estimating $K(x^n \mid y^n)$. It seeks the most appropriate function that fits the data, such that the function minimizes

$$f_{best} = arg\ min_{f \in \mathcal{F}} \{K(f) + K(e^n)\} \tag{5.32}$$

with $\mathcal{F}$ the set of admissible functions and e^n the error vector defined as $e_i = x_i - f(y_i)$ with i from 1 to n. This corresponds to Minimum Description Length (MDL) approach applied to regression analysis 2.4.2. In this approach, minimizes the sum of $L(H) + L(D \mid H)$. $K(f)$ can be approximated with $L(H)$, and $K(e^n)$ with $L(D \mid H)$. Practical complexity measurement was discussed in Section 2.4.2. For calculating $I_A(x^n : y^n)$, I assume that x_i is randomly drawn from a uniform distributions in range $[x_{min}, x_{max}]$, so that $K(x^n) = n.(x_{max} - x_{min})/p$.

For discrete variables, the conditional distributions $P(x_i \mid parents(x_i))$ are described by discrete distributions. The number of probabilities (written with precision d) in the probability table determine its complexity.

I will assume that if one of two information equivalent sets has fewer elements, the relation with the reference variable is simpler.

5.4.2 The Increase of Complexity

The complexity criterion makes sense by making the following assumption:

Assumption 5.18. *The Complexity Increase assumption:*

$$\boldsymbol{A} \not\perp\!\!\!\perp \boldsymbol{C} \ \ \& \ \ \boldsymbol{A} \perp\!\!\!\perp \boldsymbol{C} \mid \boldsymbol{B}$$
$$\Rightarrow \ \ I_A(\boldsymbol{A} : \boldsymbol{C}) \leq I_A(\boldsymbol{A} : \boldsymbol{B}) \ \ \& \ \ I_A(\boldsymbol{A} : \boldsymbol{C}) \leq I_A(\boldsymbol{B} : \boldsymbol{C}) \tag{5.33}$$

$$\boldsymbol{A} \not\perp\!\!\!\perp \boldsymbol{D} \ \ \& \ \ \boldsymbol{A} \perp\!\!\!\perp \boldsymbol{D} \mid \boldsymbol{B} \ \ \& \ \ \boldsymbol{C} \not\perp\!\!\!\perp \boldsymbol{D} \ \ \& \ \ \boldsymbol{C} \perp\!\!\!\perp \boldsymbol{D} \mid \boldsymbol{B} :$$
$$I_A(\boldsymbol{A} : \boldsymbol{B}) < I_A(\boldsymbol{B} : \boldsymbol{C}) \ \ \Leftrightarrow \ \ I_A(\boldsymbol{A} : \boldsymbol{D}) < I_A(\boldsymbol{C} : \boldsymbol{D}) \tag{5.34}$$

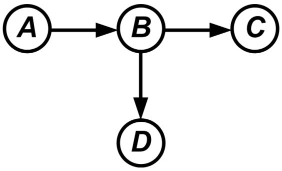


Figure 5.10: Example causal model with $I_A(A:B) < I_A(B:C)$.

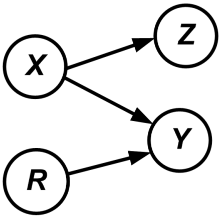


Figure 5.11: Simplest, but incorrect model for the system of Fig. 5.3.

Fig. 5.10 illustrates both cases of the assumption. The complexities of the relations do not decrease when variables are more distant in a causal model. This would happen by correspondences of the relations and neutralization of its complexities. Note the similarity with the *data processing inequality*, which states that, if $A \perp\!\!\!\perp C \mid B$, the mutual information of A and C cannot be higher than that of A and B. The assumption implies that if $X \rightarrow Y \rightarrow Z$ is the true model and X and Y are information equivalent for Z, connecting the nodes having the simplest relation gives the correct model.

In case of independent causal mechanisms, complexity increase is what 'normally' can be expected. Only rarely will it lead to cancellation of complexities, such as in specifically-designed systems like the coder-decoder model of Fig. 5.3. In which X and Y are equivalent for Z, but the relation $X - Z$ is simpler than $Y - Z$. The complexity increase assumption is violated, due to a complete dependence of the decoding relation $Y \rightarrow Z$ on both $X \rightarrow Y$ and $R \rightarrow Y$. Hence, a learning algorithm would consider the $X-Z$ relation as a direct one and not the more complex $Y-Z$ relation, as shown in Fig. 5.11. In the context of learning, choosing the simplest model is the best strategy [Grünwald et al., 2005]. It overcomes overfitting and even if the learned model deviates from the true model, it will give good predictions about the behavior of the system. The model of Fig. 5.11 correctly predicts the behavior of the coder-decoder and represents the purpose of the system.

5.5 Faithfulness

Faithful models provide a compact representation of all independencies of a distribution. To capture the independencies that follow from information equivalences, causal models are augmented.

5.5.1 Conditional Independence and Simplicity

An information equivalence cannot be modeled faithfully in the original setting. Therefore we restrict the conditional independencies that are shown graphically with the requirement that the conditioning set should provide a simpler relation in case of information equivalence.

Definition 5.19. *(Conditional independence and simplicity)* ***Conditional independence and simplicity*** *between two sets of variables* $\boldsymbol{X}$*,* $\boldsymbol{Y}$ *and a conditioning set* $\boldsymbol{Z}$*, written as* $\boldsymbol{X} \perp\!\!\!\perp_S \boldsymbol{Y} \mid \boldsymbol{Z}$*, occurs when*

- $\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y} \mid \boldsymbol{Z}$*, and*
- $I_A(\boldsymbol{Z} : \boldsymbol{Y}) > I_A(\boldsymbol{X} : \boldsymbol{Y})$ *if* $\boldsymbol{Z} \perp\!\!\!\perp \boldsymbol{Y} \mid \boldsymbol{X}$ *(*$\boldsymbol{Z}$ *and* $\boldsymbol{X}$ *are information equivalent regarding* $\boldsymbol{Y}$*), and*
- $I_A(\boldsymbol{Z} : \boldsymbol{X}) > I_A(\boldsymbol{X} : \boldsymbol{Y})$ *if* $\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Z} \mid \boldsymbol{Y}$ *(*$\boldsymbol{Z}$ *and* $\boldsymbol{Y}$ *are information equivalent regarding* $\boldsymbol{X}$*).*

5.5.2 D_{eq}-separation

When there are deterministic relationships among variables, there are conditional independencies that are not entailed by the Markov condition alone. Spirtes et al. [1993], based on the work of Geiger [1990], enlarged the concept of d-separation to create a graphical condition for retrieving all conditional independencies from a graph and a set of deterministic relations. They called it D-separation. I enlarge the criterion to also capture independencies following from information equivalences.

Definition 5.20. *(*D_{eq}*-separation) Let* p *be a path between a node* U *and a node* W *of a DAG* G*. Path* p *is called blocked given subset* $\boldsymbol{V}$ *of nodes in* G *and a set of deterministic relations and information equivalences if there is a node* v *on* p *satisfying one of the following conditions:*

1. v *has converging arrows (along* p*) and neither* v *nor any of its descendants are in* $\boldsymbol{V}$*, or*
2. v *does not have converging arrows (along* p*) and* v *is in* $\boldsymbol{V}$ *or is determined by* $\boldsymbol{V}$*.*

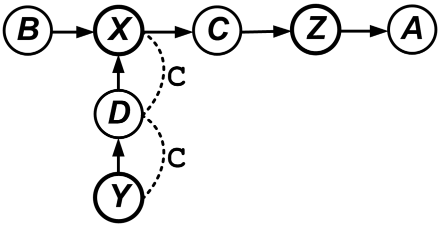


Figure 5.12: Model with X and Y information equivalent for Z and additional nodes depicting the possible consequences (same model as in Fig. 5.7).

*$\boldsymbol{V}$ and the set of deterministic relations and information equivalences is said to D_{eq}-**separate** U from W in G, denoted $U \bot_{eq} W \mid \boldsymbol{V}$, iff members of $\boldsymbol{V}$ block every path from U to W or there is an equivalence of $\boldsymbol{X}$ and $\boldsymbol{Y}$ with respect to Z such that*

1. *$\boldsymbol{Y} \subset \boldsymbol{V}$, and*
2. *the members of $(\boldsymbol{V} \setminus \boldsymbol{Y}) \cup \boldsymbol{X}$ block every path from U to W, and*
3. *the members of $(\boldsymbol{V} \setminus \boldsymbol{Y}) \cup \{Z\}$ block every path from U to W, and*
4. *members of $(\boldsymbol{V} \setminus \boldsymbol{Y})$ do not unblock a path between U and W that is not blocked by $\boldsymbol{X}$.*

Consider the model of Fig. 5.12, A and B are d-separated by X, but not by Y. If, however, X and Y are information equivalent with respect of C, A and B are D_{eq}-separated by conditioning on Y.

5.5.3 Faithfulness Revisited

Given the additional independencies that information equivalences entail, the definition of faithfulness is reconsidered. In cases of information equivalences, the independencies depicted graphically are restricted by the definition of conditional independency and simplicity ($\perp\!\!\!\perp_s$). On the other hand, the extended d-separation criterion ($\bot_{eq}$) makes it possible to retrieve the independencies following from information equivalences.

Definition 5.21. *A causal model is called **faithful**$_{eq}$ to a probability distribution containing information equivalences if*

$$\boldsymbol{X} \bot_{eq} \boldsymbol{Y} \mid \boldsymbol{Z} \;\Leftrightarrow\; \boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y} \mid \boldsymbol{Z} \tag{5.35}$$

$$\boldsymbol{X} \bot \boldsymbol{Y} \mid \boldsymbol{Z} \;\Leftrightarrow\; \boldsymbol{X} \perp\!\!\!\perp_s \boldsymbol{Y} \mid \boldsymbol{Z} \tag{5.36}$$

I will show that the definition makes sense by proving that the consequences of combinations of an information equivalence and conditional independencies that follow from the Markov condition can be captured by a model that is $faithful_{eq}$. Take X and Y equivalent for Z and $X - Z$ the simplest relation, $I_A(X : Z) > I_A(Y : Z)$. Consider conditional independence statements containing at least two of the three variables of the information equivalence. There are ten possible combinations:

1. $X \not\perp\!\!\!\perp A$ and $X \perp\!\!\!\perp A \mid Z$:

   - If also $Y \perp\!\!\!\perp A \mid Z$, by property 5.11 it follows that X are Y equivalent for A. The second part of the Complexity Increase Assumption assures that $I_A(X : A) > I_A(Y : A)$, thus $Y \perp\!\!\!\perp_S A \mid X$, but $X \not\perp\!\!\!\perp_S A \mid Y$. An example is shown in Fig. 5.12.
   - On the other hand, if $Y \not\perp\!\!\!\perp A \mid Z$, then Y is connected to A via an alternative path and has more information about A than X.

2. $Z \not\perp\!\!\!\perp B$ and $Z \perp\!\!\!\perp B \mid X$:
   Independency $Z \perp\!\!\!\perp B \mid Y$ follows property 5.12. Then, there are two possibilities:

   - If B has less information about Z ($Z \not\perp\!\!\!\perp X \mid B$), it is related to Z via X, as shown in Fig. 5.12. By D_{eq}-separation the conditional independency $Z \perp\!\!\!\perp B \mid Y$ can be retrieved from the graph.
   - If on the contrary variable B contains as much information about Z as X, all three nodes are equivalent for Z. This is shown by node D in Fig. 5.12. The node having the simplest relation with Z is related to Z, which is X in the figure.

3. $Z \not\perp\!\!\!\perp C$ and $X \perp\!\!\!\perp Z \mid C$:
   By property 5.13, Y also gets independent, $Y \perp\!\!\!\perp Z \mid C$. Then, there are two possible cases:

   - If $C \perp\!\!\!\perp Z \mid X$, then C is also information equivalent with respect to Z, which is discussed in the previous case.
   - If $C \not\perp\!\!\!\perp Z \mid X$, then C has more information about Z. Property 5.13 proves that X or Y are information equivalent for C as well. This case is shown by node C in Fig. 5.12. By the second part of the Complexity Increase Assumption, $X - C$ must be simpler than $Y - C$, thus $Y \perp\!\!\!\perp_S C \mid X$.

4. $X \perp\!\!\!\perp Y \mid D$:
   By assumption 5.10, it follows that $X \perp\!\!\!\perp Z \mid D$, which is discussed by case 3.

5. Independency $X \perp\!\!\!\perp E \mid Y$ only interferes with the equivalence if there is an independence with Z. This is discussed by the previous cases.

The 5 remaining cases, $Y \perp\!\!\!\perp A \mid Z$, $Z \perp\!\!\!\perp B \mid Y$, $X \perp\!\!\!\perp Z \mid C$, $Y \perp\!\!\!\perp Z \mid D$ and $Y \perp\!\!\!\perp D \mid X$, are equivalent, by symmetry, to respectively cases 1, 2, 3, 4 and 5.

5.6 Constraint-based Learning Algorithms

Recall the PC algorithm, discussed in section 3.4.1. The graph is constructed in two steps. The first step, called *adjacency search*, learns the undirected graph and the second step tries to orient the edges.

The construction of the undirected graph is based on the property that two nodes are adjacent if they remain dependent by conditioning on every set of nodes that does not include both nodes. The algorithm starts with a complete undirected graph and removes edges for each independency that is found. The number of nodes in the conditioning set is gradually increased up to a certain maximal number, called the *depth* of the search. The orientation step is based on the identification of v-structures of the form $X \rightarrow Y \leftarrow Z$, for which X and Z are independent, but become dependent conditional on Y. Recall that for all three other possible orientations of $X - Y - Z$ the opposite is true, X and Z are initially dependent, but become independent by conditioning on Y. During both steps, the following extensions should be considered.

5.6.1 Equivalence Detection

Information equivalences pose a problem for the constraint-based algorithms. Take X and Y equivalent for Z, by $Y \perp\!\!\!\perp Z \mid X$ the algorithm would remove the $Y - Z$ edge and $X \perp\!\!\!\perp Z \mid Y$ deletes the $X - Z$ edge. Information equivalences should therefore be detected during the construction of the undirected graph. For each conditional independency that is found, it should be tested whether an equivalence can be found by swapping variables of the conditioning set with one of both arguments. Furthermore, equivalences imply independencies. For equivalence X and Y for Z, any independency $Y \perp\!\!\!\perp Z \mid X, U$ would be a consequence of the information equivalence. Such tests can thus be skipped in the procedure. This results in Algorithm 5.1.

Algorithm 5.1 Information equivalence detection during adjacency search of PC algorithm

Input:

Output: For each test $U \perp\!\!\!\perp V \mid \boldsymbol{W}$ during the adjacency search:

1. Skip test if an equivalence $\boldsymbol{U}^{+}$ and $\boldsymbol{W}^{-}$ for V, or $\boldsymbol{V}^{+}$ and $\boldsymbol{W}^{-}$ for U has been found previously. $\boldsymbol{U}^{+}$ is defined as a set containing U and some other nodes and $\boldsymbol{W}^{-}$ denotes a subset of $\boldsymbol{W}$.

2. If the independence test turns out positive, check for equivalences $\boldsymbol{U}^{*}$ and $\boldsymbol{W}$ for V, and $\boldsymbol{V}^{*}$ and $\boldsymbol{W}$ for U, with $\boldsymbol{U}^{*}$ and $\boldsymbol{V}^{*}$ sets containing U and V respectively and some nodes adjacent to V and U respectively such that they have the same number of elements as $\boldsymbol{W}$.

3. If an equivalence is found, it is added to the model. Unless there was already an equivalence found of one of both equivalent nodes sets with for the same reference variable, then the other set is added to that equivalence.

4. If both equivalences hold, $\boldsymbol{U}^{*}$, $\boldsymbol{V}^{*}$ and $\boldsymbol{W}$ form a multi-node equivalence.

5. Edge $U - V$ is not removed from the graph.

5.6.2 Equivalence selection

The second step of the extended PC algorithm alternates selection among equivalent relations, given by Algorithm 5.2, with the original orientation step until no more equivalences or undirected edges can be resolved. For orientation, the original orientation rules can be applied on the graph. If for an information equivalence, relation $X - Z$ is considered simpler than relation $Y - Z$, node Y has to be regarded as separated from Z by X, while X is not separated from Z by Y. This d-separation information is used by the orientation rules. For the equivalences that could not be resolved with the complexity criterion, the equivalent node set that are causes of the other equivalent node set are chosen as adjacent to the reference variable.

Algorithm 5.2 Edge selection among information equivalences by the complexity criterion

Input:

Output: To evaluate information equivalences of $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_n$ with respect to Z, compare the complexities of the following functions f_i with i from 1 to n, only if the nodes $W_{i,j} \in \boldsymbol{W}_i$ are still connected to Z.

1. if all edges connecting the equivalent nodes and the reference variable are oriented:

   (a) if the edges are oriented toward the reference variable, consider the following functions: $Z = f_i(\boldsymbol{W}_i,$ *other nodes with edges oriented to reference node* $Z)$.

   (b) if the edges are oriented from the reference variable, in order to evaluate $\boldsymbol{W}_i$, count up the complexities of the functions $W_{i,j} = f_i(Z,$ *other nodes with incoming edges to* $W_{i,j})$, for all $W_{i,j} \in \boldsymbol{W}_i$.

2. if some edges are not oriented, consider the following functions: $Z = f_i(\boldsymbol{W}_i)$.

The complexity of the functions is estimated as explained in section 5.4.1. If the complexity of the simplest function differs by at least 8 bits (this threshold corresponds to 1 operation) with the complexities of the other functions, the corresponding equivalent nodes can be related with the reference variable, the edges of the other equivalent nodes with the reference variable are removed. For *multi-node equivalences*, the simplest edges should remain in the graph such that all nodes are connected.

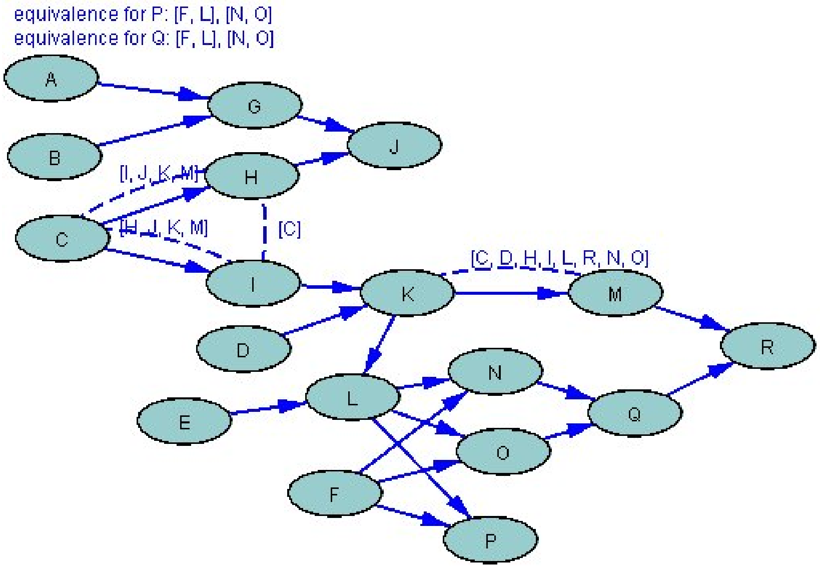


Figure 5.13: Model learned from random data generated by a set of structural equations.

5.7 Experiments

This section reports on learning models based on generated data. 150 data points were generated using the following structural equations:

$G = A + B$	$K = D + 0.4I + err$	$O = 10 + 1.3L - F$
$H = 4C^2$	$L = 1.5 + 0.35K + 0.35E + err$	$P = L.F + err$
$I = 0.4C^3$	$M = 1 + 0.04K^2$	$Q = 10N/O + err$
$J = G + 0.4H$	$N = 10 + L + F$	$R = M + 0.4Q + err$

Variables A, B, C, D, E and F are randomly chosen between 0 and 10 and err is an error term, reflecting a random disturbance with maximal size 1/8 of the variable's range (the difference between its maximal and minimal value). There are 7 deterministic variables (G, H, I, J, M, N and O). The system incorporates the different cases discussed in this paper: a bijective relation between K and M; G is determined by A and B; multi-node equivalence C, H and I; and equivalence of (L, F) and (N, O) for P and Q.

The extended PC algorithm with default options is applied onto the data. To handle data with non-linear relations, an independence test is used based on the conditional mutual information and kernel density estimation, which will be explained in Section 7.3. The augmented Bayesian network learned by the extended PC algorithm, depicted in Fig. 5.13, provides a correct model. Remark that the algorithm does not check for deterministic relations, only for information equivalences. This strategy was followed to verify that the algorithm works.

I also verified the correctness of the assumptions. Assumptions 5.8 and 5.10 hold for the generated data. Weak transitivity, however, was violated in 31 out of the 1022 cases (in which Eq. 5.14 applies). The cases are characterized by a failure of the independence test in detecting dependencies between 'distant' variables. The influence of a variable on another variable happens via multiple variables so that the random disturbances get to dominate the influence. For example, the test classified A and M as independent, while weak transitivity expects dependence (or $A \not\perp\!\!\!\perp M \mid J$, which is also not true), since $A \not\perp\!\!\!\perp J$ and $J \not\perp\!\!\!\perp M$. The test also returned $E \perp\!\!\!\perp K$ and $E \perp\!\!\!\perp K \mid N$, but $E \not\perp\!\!\!\perp N$ and $K \not\perp\!\!\!\perp N$ demand a dependence.

The learning module together with the experimental data can be found on the web. They can be accessed at `http://parallel.vub.ac.be`.

5.8 Summary of Chapter

Deterministic relations and, more generally, information equivalences pose problems for the current constraint-based learning algorithms. This limitation had to be overcome in order to be able to apply the algorithms on performance data. Information equivalences give rise to violations of the intersection condition and the inexistence of a faithful graph. They generate independencies that do not follow from the Markov condition. Information equivalences appear when two sets of variables in some sense have the same information about another variable. The probability distribution is then characterized by an equivalent partitioning.

In order to capture information equivalences, this chapter provided extensions to the definition of causal models and to the PC algorithm according. The extensions were developed according to the same philosophy as the original theory:

- *The conditional independencies provide the required information.* Information equivalences are detected by a violation of the intersection condition.

- *The aim is a faithful model, one that describes all conditional independencies.* To incorporate information equivalences, an augmented

Bayesian network was defined and faithfulness was redefined. Under weak transitivity and two other mild assumptions, information equivalences can be characterized by basic information equivalences, which are added to the augmented Bayesian network. To retrieve the conditional independencies that follow from information equivalences and the Markov condition from the graph, the d-separation criterion was enlarged. To ensure minimality of the model, the complexity of the relations was introduced to determine adjacency among information equivalent relations. Faithfulness can then be reestablished by enlarging the definition of conditional independency with the requirement of simplicity.

- Causal models aim at describing the qualitative properties of a system. A deterministic relation is clearly a regularity. The description of $P(Y \mid X)$ can be compressed considerably when $y = f(x)$.

- *Constraint-based learning algorithms rely on evidence.* The complexity of relations was introduced as a criterion to determine adjacency among information equivalent relations. Complexity is the evidence which allows for the construction of correct models. At least under the assumption that the complexity of the relations increases for more distant variables, which I called the Complexity Increase Assumption. The PC algorithm could easily be extended for learning the augmented models. Experiments with generated data show that the assumptions hold and correct models are learned.

It must be added that the results are also conform the regularity-philosophy unfolded in the previous chapter. Information equivalences are a kind of regularities that were not described by Bayesian networks so far. They are characterized by an equivalent partitioning. Note that the assumptions made for the here developed extensions are based on strict regularities too. An example of the violation of weak transitivity was discussed in the previous chapter, counterexample 7 of Section 4.3. The invalidity of Assumptions 5.8 and 5.10 also happens when the probability distribution exhibits strict regularities. Finally, the violation of the Complexity Increase Assumption happens when complexities along a causal path neutralize each other, so that the net effect is a simpler complexity for more distant variables. The correspondence of the relations has to be viewed as yet a new regularity. Not only should this regularity be added to the model, this regularity also provides useful information. If $A \rightarrow B$ is known to be quite complex and $B \rightarrow C$ too, then $A \rightarrow C$ will be expected to be even more complex. If this is not the case, it is an interesting fact about the system.

This chapter concludes the first part, the theoretical and philosophical study of causal inference.

Part II

Performance Modeling

AN overview of the actors appearing in the second part of my work must start with *computers*, these pieces of hardware allowing for calculations and information processing. The speed of the evolution of computer technology is astonishingly swift, such as the ever-lasting increase of processing power. But despite the doubling every 18 months of the processing power (Moore's Law), it cannot catch up with the growing demand for computational resources. This is triggered by, for example, the continuing release of new applications and the introduction of scientific computing in new fields, such as bio-informatics. *Performance* remains a constant problem for computer application and system developers. Consider also the present need for small, cheap and energy-efficient systems. However, performance becomes an increasingly difficult matter to master, since the growth of available processing power is paralleled with an almost similar expansion of the complexity of today's systems.

A solution for responding to the demand of processing power lies in *parallel processing.* The idea is to connect (simple) computing units so that the accumulated resources become available at once. A program is rewritten in a parallel version, which will perform the same calculations in parallel on the available computing units. Performance is the 'raison d'être' of parallelization. Understanding performance is therefore essential. The increased complexity of interactions among the intertwined processes of computation and communication makes it even more challenging. Software and hardware developers must require insight in the aspects consuming most of the resources: knowledge, at a high level, of the program sections that are responsible for a bad efficiency; knowledge, at a low level, of the bottlenecks in the system. This is nowadays supported by advanced performance monitoring and analysis tools. The challenge for computer scientists is to provide the application and system developer with a clear and understandable *performance analysis.* The track which I chose to pursue in the so-diverse world of the study of computers' performance is to automate the construction of models about performance. Models which are learned by the *statistical analysis* of experimental data. Models about performance serve two purposes: prediction and understanding. Quantitative models are required for estimating the performance for a given configuration. Qualitative models uncover which and how parameters and variables affect the overall performance. I will focus on the qualitative models learned by the algorithms for causal inference.

The overall performance depends on the match of program and system. A certain kind of programs might run faster on a certain type of systems, while another kind might be more efficient on other systems. The question arises to what extent *generic models* exist which are valid for combinations of applications and systems. Whether it is possible to characterize an application at once, so that its performance can be predicted on any system. The worst case scenario is that any combination of application and system results in a different model. This problem is addressed in the last chapter. I will show that regularities play a decisive role in the match of program and system.

Chapter 6

Performance Analysis of Parallel Processing

THE second part of my work studies the modeling of the performance of sequential and parallel programs. This section will introduce parallel processing and the analysis of its performance.

Parallel processing is the only answer to the ever-increasing demand for more computational power. Nowadays, the big giants in hardware and software, like Intel and Microsoft, are increasingly aware of it and have pounced onto the market. But unlike sequential programs running on the Van Neumann computer, the parallelization of programs is not trivial. It depends quite heavily on the underlying parallel system architecture. Automatic parallelization of programs is a 50-year old dream in which a program is efficiently matched with the available computing resources. This has become possible, but only for a very limited number of applications, the class of trivially parallelizable programs. For those, the computational work can be divided into parts which can be processed completely independently. Other programs, on the other hand, need manual adaptation to the available resources. This cannot be achieved without a detailed understanding of the algorithm. Intelligent reasoning is necessary to engineer the matching of the patterns of the concurrently operating entities to the pattern of the processors and the network resources, in order to obtain an efficient interplay of computation and communication. The aim of a performance analysis is to provide support for the developer of parallel programs.

The goals of a performance analysis are multifold:

- An understanding of the computational process in terms of the underlying processes: instructions performed, processor cycles spent, cache misses, memory hierarchy utilization, communication, resource utilization per program section, number of iterations, and so on.
- An identification of inefficient patterns, the bottlenecks that unnecessarily slow down the execution process. In particular, performance values that are, given the context, 'abnormally' low and which can be considered for optimization and improvements. They indicate up to which points tuning efforts are most effective.
- A prediction of the performance for a new program or system configurations. A performance model should provide an expectation of the achievable performance with a reasonable fidelity, as a function of program and system parameters.
- The definition of program and system properties that fully characterize their performance, i.e. which allow the quantification of their performance for a wide range of systems and system configurations.

Various tools exist nowadays for automated diagnosis and control. Considerably more effort is needed to improve current work to present the user a simple, comprehensible and reasonably accurate performance evaluation [Pancake, 1999]. Current challenges are further automation, tackling complex situations (e.g. GRID environments [Nemeth et al., 2004]) and providing the software developer with understandable results with a minimum of learning overhead. To sketch the difficulty of the task, consider the study of network performance. Communication delays should be attributed to the different steps of the communication process, such as machine latency, transfer time, network contention, flight time, etc [Badia, 2003]. A correct understanding of the origins of the delays is indispensable. The task of identifying them becomes even more difficult when implementation-specific low level issues come into play, such as specific protocol behavior, window delays or chatter [NetPredict, 2003]. These are not always fully understood and can often not be measured directly.

The first section discusses the parallel performance metrics which where employed to build our performance analysis tool, EPPA. These metrics are based on the lost cycle approach. Overhead ratios are defined to quantify the impact of each type of lost cycle, or overhead, on the overall performance, the speedup. The second section explains which information is recorded by EPPA and compares the tool with related work.

6.1 Parallel Performance Metrics

Parallel processing is the simultaneous execution of a program by multiple processors. **Parallelization** is the rewriting of a sequential program into a program that can be processed in parallel and that gives the same result as the sequential program. The advantage is that the combined computing and memory resources of the processor group can be utilized. Calculation or memory intensive programs can fruitfully exploit the aggregated resources to finish the job in less time.

Example 6.1 (Parallel processing I: Protein folding).

> Proteins are long chains of thousands of amino acids. After creation, the sequence 'folds' into a unique 3-dimensional structure that determines the protein's properties. The shape into which a protein naturally folds is known as its native state. Fig. 6.1 shows an antibody against cholera, unfolded and in its native state. Understanding the structure of a protein is critical in understanding its biological function. The structure of (synthetic) proteins can be determined by running detailed simulations of the folding process. Because of the complexity and multitude of interactions, these computations require 'zillions' of processor cycles. It takes with today's computers about 10000 days to simulate a particular folding of an average protein.
>
> **Folding@Home** is a distributed computing project from Stanford university to tackle this performance problem (`http://folding.stanford.edu/`). People from throughout the world run the software and make one of the largest supercomputers in the world in the form of a computational grid. The participation in this project throughout the world is depicted in Fig. 6.2. Every computer runs a section of the simulation for one of the many protein foldings that need to be calculated in research on Alzheimer's Disease, Cancer, Parkinson's Disease, etc. To contribute, you simply install a small program on your computer which runs in the background only consuming processor time when there is no other work.

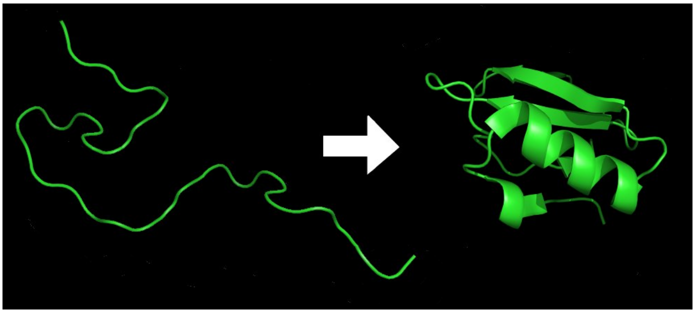


Figure 6.1: Protein folding, from amino acid sequence to a 3-dimensional structure.

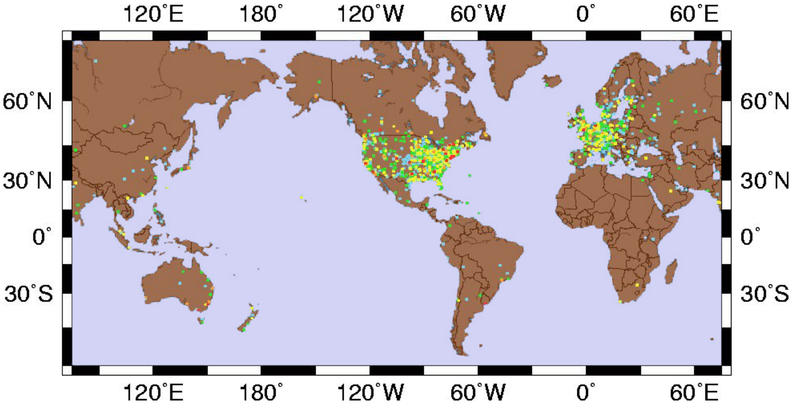


Figure 6.2: Folding@Home's supercomputer. Distribution accross the world of the computers participating in the project.

The runtime of a sequential program is defined as T_{seq}. The parallel version, whose runtime is denoted as T_{par}, will hopefully finish faster. The profit of switching from one to multiple processors is characterized by the **speedup**:

$$Speedup = \frac{T_{seq}}{T_{par}} \tag{6.1}$$

It expresses how much faster the parallel version runs relative to the sequential one. Note that in the context of parallel processing I denote the computation time of the sequential program as T_{seq}, whereas in context of sequential computing I denote the computation time as T_{comp}.

When the parallel program is run on p processors, the **efficiency** is defined as:

$$Efficiency = \frac{Speedup}{p} = \frac{T_{seq}}{p.T_{par}} \tag{6.2}$$

Efficiency measures how well a processor is used during the parallel computation. It represents the effectivity of the set of cooperating parallel processes. Efficiency quantifies the portion of the parallel runtime during which the processors where doing *useful work*, i.e. when the parallel execution is performing parts of the sequential execution. Ideally, the efficiency is 100%, which is equivalent to a speedup of p. Each processor optimally executes an equal part of the sequential program. In practice, the effectivity of the parallel program is limited due to the inevitable *parallel overhead*, such as communication of data between the processors. Hence, speedup will be smaller than p. It can even become lower than 1, which indicates a slow down instead of a speed up. On the other hand is it also possible to attain a speedup higher than p, called *superlinear speedup*. It typically occurs when the parallel program succeeds in a more efficient utilization of the memory hierarchies of the processors. This results in lower access times of the memory hierarchies.

It must be noted that the here developed performance metrics focus on the *computation time* of the process. Other performance metrics, such as energy utilization, are, despite their increasing importance, not considered here. On the other hand, a generic approach is pursued, one that applies for a multivariate analysis in general.

6.1.1 Lost Cycle Approach

For the analysis of the parallel runtime and overhead, I adopt the *lost cycle approach*, as conceived by Crovella and LeBlanc [1994]. It provides a measure of the impact of the overhead on the speedup. Ideally, each processor computes its part of the total work. Thus without additional work, we would have $T_{par} = T_{seq}/p$. The work is divided among the processors of the parallel system. The total useful computational work is characterized by the sequential runtime. The *Speedup* then equals to p. In practice, however, additional processor cycles are needed to manage the parallel execution. **Overhead** is therefore defined as

$$overhead = p.T_{par} - T_{seq}. \tag{6.3}$$

Each process has T_{par} time allocated to perform its part of the job. The cycles during this period that are not employed for useful computation are therefore considered *lost processor cycles*. Take the following example.

Example 6.2 (Parallel processing II: Parallel Matrix Multiplication).

Consider the multiplication of 2 square matrices with size $n \times n$: $C = A \times B$. The elements of the product matrix C are calculated according to the formula

$$C_{i,j} = \sum_{k=1}^{n} A_{i,k}.B_{k,j}, \tag{6.4}$$

with i and j indicating respectively the row and column of the element. The computation involves n^3 multiplications and $n^2 \times (n-1)$ additions. The runtime rapidly increases for higher values of n, what makes it worth for being computed in parallel for high values of n. There exist many ways to calculate the product in parallel. A simple version is illustrated by Fig. 6.3. The A matrix is striped into p blocks n/p of contiguous rows, the B matrix into p blocks of n/p columns. They are distributed among the p processors. Each processor stores a submatrix of A and one of B, labeled in Fig. 6.3, in which p is 3. A master processor does the partitioning and sends the submatrices to the slave processors. The algorithm then alternates p computation and communication steps. In each computation step, each processor multiplies its A submatrix with its B submatrix, resulting in a submatrix of C. The black circles in Fig. 6.3 indicate the step in which each submatrix is computed. After the multiplication, each processor sends it B submatrix to the next processor and receives one from the preceding processor, in such way that the communication forms a circular shift operation. When finished, the slaves send their part of C to the master computer. The timeline of the execution on our

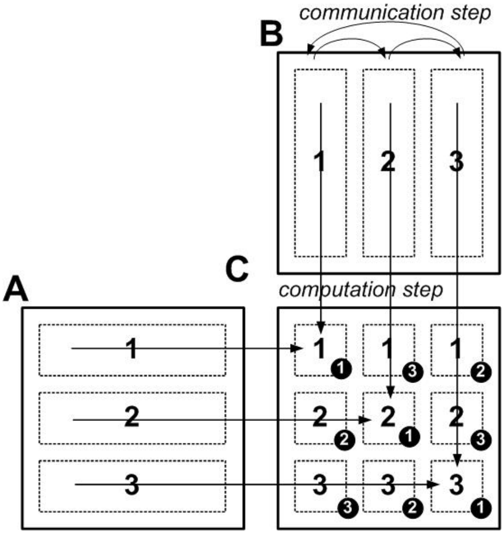


Figure 6.3: Parallel Matrix Multiplication on 3 processors: partitioning, computation and communication in 3 steps. At each step, 3 submatrices are calculated, indicated with black circles.

> cluster of Pentium II processors connected by a 100MBs switch is shown in Fig. 6.4. Two types of overhead can be identified: communication and idling. The speedup for the computation of a 100 × 100 matrix is 2.55 and the efficiency is 85%.

The parallel runtime on processor i consists of its part of the useful work, T^i_{work}, and the cycles spent on the overheads. The impact of the different types of overhead will be analyzed separately. Each overhead type is labeled with an index j. The number of overhead types is denoted with O. $T^{i,j}_{ovh}$ then denotes the time of overhead j on processor i. The runtime on every processor can then be written as:

$$T^i_{par} = T^i_{work} + \sum_j^O T^{i,j}_{ovh} \quad \text{with} \quad i = 1 \dots p \tag{6.5}$$

$$T_{seq} + T_{anomaly} = \sum_i^p T^i_{work} \tag{6.6}$$

where $T_{anomaly}$ is the difference between the sum of all cycles spent on useful work by the different processors and the sequential runtime. In

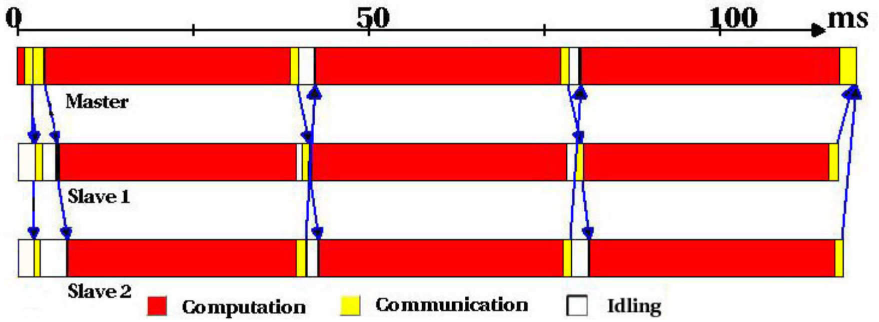


Figure 6.4: Execution profile of a Parallel Matrix Multiplication of two 100x100 matrices.

most cases it is very close to zero. If positive, the execution of the useful work takes more time in parallel. If negative, the parallel execution is faster, for example by a more efficient use of the memory hierarchy. The parallel runtime is the same on all processors:

$$T_{par} = T^1_{par} = \ldots = T^p_{par}. \tag{6.7}$$

Hence, we may write:

$$T_{par} = \frac{\sum\limits_i^p T^i_{par}}{p} \tag{6.8}$$

Together with 6.5 it follows that

$$T_{par} = \frac{\sum\limits_i^p T^i_{work} + \sum\limits_i^p \sum\limits_j^O T^{i,j}_{ovh}}{p} \tag{6.9}$$

$$= \frac{T_{seq} + T_{anomaly} + \sum\limits_j^O T^j_{ovh}}{p} \tag{6.10}$$

with $T^j_{ovh} = \sum\limits_i^p T^{i,j}_{ovh}$, the total time of overhead j. Parallel anomaly is also regarded as overhead, although that it might be negative. It is therefore added to the overheads, T_{ovh}^{O+1} The speedup can then be rewritten as

$$Speedup = \frac{T_{seq}}{\frac{T_{seq} + \sum_{j}^{O+1} T_{ovh}^{j}}{p}} = \frac{p}{\frac{T_{seq}}{T_{seq}} + \frac{\sum_{j}^{O+1} T_{ovh}^{j}}{T_{seq}}} \tag{6.11}$$

Hence [Kumar and Gupta, 1994]:

$$\boxed{Speedup = \frac{p}{1 + \sum_{j} \frac{T_{ovh}^{j}}{T_{seq}}}.} \tag{6.12}$$

The equation expresses how the overheads influence the speedup. The lost processor cycles must be considered relative to the sequential runtime. Without any overhead, the speedup equals to p.

6.1.2 Overhead Ratios

From Eq. 6.12 it follows that the impact of overhead on the speedup is reflected by its ratio with the sequential runtime. I call these terms the **overhead ratios**. They express the relative weight of the overhead term:

$$Ovh_j = \frac{T_{ovh}^{j}}{T_{seq}}. \tag{6.13}$$

The speedup is then:

$$Speedup = \frac{p}{1 + \sum_{j} Ovh_j}, \tag{6.14}$$

and the efficiency gives

$$Efficiency = \frac{1}{1 + \sum_{j} Ovh_j}. \tag{6.15}$$

These definitions differ slightly from the **normalized performance indices** used by the performance tool AIMS, defined as $index_j = T_{ovh}^{j}/T_{par}$ [Sarukkai et al., 1994]. They are always less than one, while the overhead ratios become more than one if the runtime of the overhead surpasses the sequential runtime. The advantage of the overhead ratios is that they are

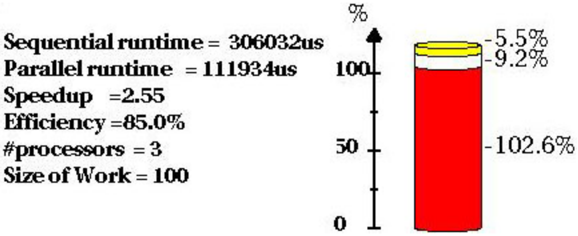


Figure 6.5: Parallel Matrix Multiplication on 3 processors: overall performance.

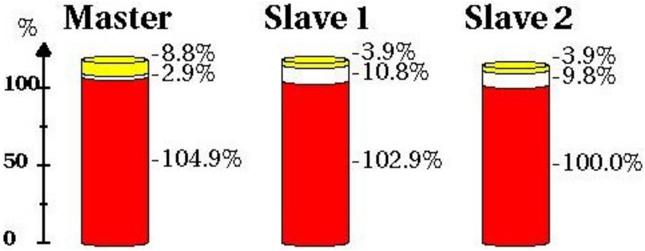


Figure 6.6: Parallel Matrix Multiplication on 3 processors: overhead ratios per process.

independent of the other overheads. This is not the case for the indices, since T_{par} incorporates all overheads. If one overhead increases, its index increases and the indices of the others decrease, since their relative weight decreases.

Example 6.3 (Overheads of Parallel Matrix Multiplication).

Fig. 6.5 shows the overall performance of the run of the previous example. Two overheads are identified: the communication and the idle time. Their ratio with the sequential time, Ovh_j, is given. The sum of the processor's computation times, $\sum_i^p T_{work}^i$, divided by the sequential runtime is also given, but is not equal to 100%. A value of 100% means that the computation time of the useful work is equal for a sequential as for a parallel execution. It is 102.6% instead, which means that the overhead ratio of the parallel anomaly is 2.6%. In parallel, 2.6% more cycles are needed to do the same work. Additionally, Fig. 6.5 shows the overhead ratios per processor individually.

6.1.3 Overhead Classification

The different overheads of a parallel execution can be classified into the following classes:

1. *Control of parallelism* ($T_{ctrlPar}$) identifies the extra functionality necessary for parallelization. This additional work can be further subdivided into the different logical parts of the parallel algorithm, like partitioning or synchronization, as done by several authors [Bull, 1996] [Truong and Fahringer, 2002b].

2. *Communication* (T_{comm}) is the overhead due to the exchange of data between processors. It is defined as the overhead time not overlapping with computation: the computational overhead due to the exchange of data between processes, in the sense of loss of processor cycles due to a communication operation.

3. *Idling* (T_{idle}) is the processors idle time. It happens when a processor has to wait for further information before it can continue. Reasons for idling are for example load imbalances, when the work is unequally distributed among the processes, or a bottleneck at the master, when it has to serve all slaves.

4. *Parallel anomaly* ($T_{anomaly}$) is the difference between the sum of all cycles spent on useful work by the different processors and the sequential runtime (Eq. 6.6). By the alternative speedup formula (Eq. 6.12), $T_{anomaly}$ was regarded as overhead. It influences the speedup, given by its ratio with the sequential runtime.

6.1.4 Granularity

To illustrate how a performance analysis is performed, this section introduces one of its most influential concepts: granularity. The key to the execution of parallel algorithms is the communication pattern between concurrently operating entities. By choosing speedup as the main goal of parallelization, Eq. 6.12 shows that overheads should be considered relatively. Communication overhead T_{comm} must be considered with respect to the computation time. The inverse of the communication overhead ratio is called the **granularity** [Stone, 1990]:

$$Granularity = \frac{T_{comp}}{T_{comm}} = \frac{1}{Ovh_{comm}} \tag{6.16}$$

Granularity is a relative measure of the ratio of the amount of computation to the amount of communication of a parallel algorithm implementation. The bigger the granularity, the more the application spends time in computation relative to communication. Another interpretation is that it expresses the size of the tasks. Since the communication is often the main overhead, the granularity gives a good indication of the feasibility of parallelization. With a granularity of one, the efficiency is 50%.

The communication time can be modeled as a simple linear function of the transmitted data size and a constant additive factor representing link startup overheads (*latency*). This is a conventional approach in analyzing communications for most message-passing, distributed systems [Steed and Clement, 1996]. The communication time can thus be split into a component proportional to the communicated data size and a part proportional to the latency of the communication links. For computing-intensive tasks and large data chunks, the data-proportional part overwhelms the constant part so that the latter can be neglected. Since parallelization is used for computation-intensive tasks, the approximation is valid. The communication overhead time can then be written as $\beta.q_{data}$, with q_{data} the size in bytes of the communicated data. Assume that we can approximate the computation time by $\tau.q_{operations}$, with $q_{operations}$ the number of basic operations of the algorithm and τ the cycles per operation. The granularity can then be rewritten as:

$$Granularity = \frac{T_{comp}}{T_{comm}} = \frac{\tau}{\beta}.\frac{q_{operations}}{q_{data}} \tag{6.17}$$

This ratio depends on hardware and software, so τ/β is called the **hardware granularity** and $q_{operations}/q_{data}$ the **software granularity**. The performance is affected by the overall granularity, independent of how it is spread over software and hardware.

6.1.5 Parameter Dependence

As can be expected, parallel performance heavily depends on program and system configuration. Most numerical, computation-intensive algorithms have a parameter that determines the size of the computational work. I call it the **work size** parameter, which I denote with W. For parallel processing, p, the number of processors participating, is the most important system parameter. The following example gives a typical parameter dependency analysis.

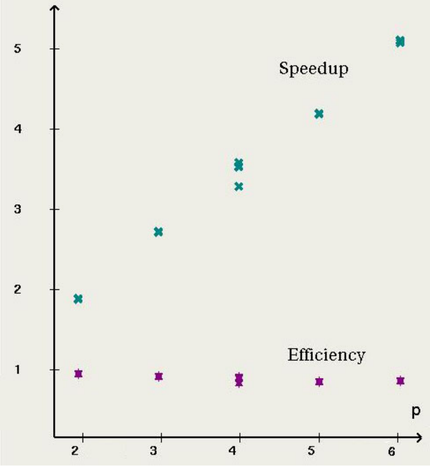


Figure 6.7: Performance of Parallel Matrix Multiplication as a function of number of processors (with $n = 100$).

Example 6.4 (Parameter Dependence of Parallel Matrix Multiplication).

Let's go back to the example of the multiplication of 2 $n \times n$ matrices. Fig. 6.7 shows speedup and efficiency in function of p. Although the speedup increases when more processors are employed, the efficiency decreases.

Performance as a function of matrix size n gives a different picture. Experimental results are shown in Fig. 6.8. Communication increases with increasing n, but computation increases a lot faster, as it is proportional to n^3. The net result is that the impact of the overhead decreases and, consequently, the efficiency increases. With $p = 3$, the ideal speedup is 3. The results show that with increasing p, the speedup asymptotically approaches the ideal speedup. For large matrices, the communication overhead can be neglected and an ideal speedup can be achieved. Software granularity is proportional to n.

The performance results for matrix multiplication are typical for a lot of parallel programs: overhead increases with p so that speedup decreases and overhead relatively decreases with increasing work size W. Applications with a computational part that increases faster than the communication as a function of W are appropriate for parallelization.

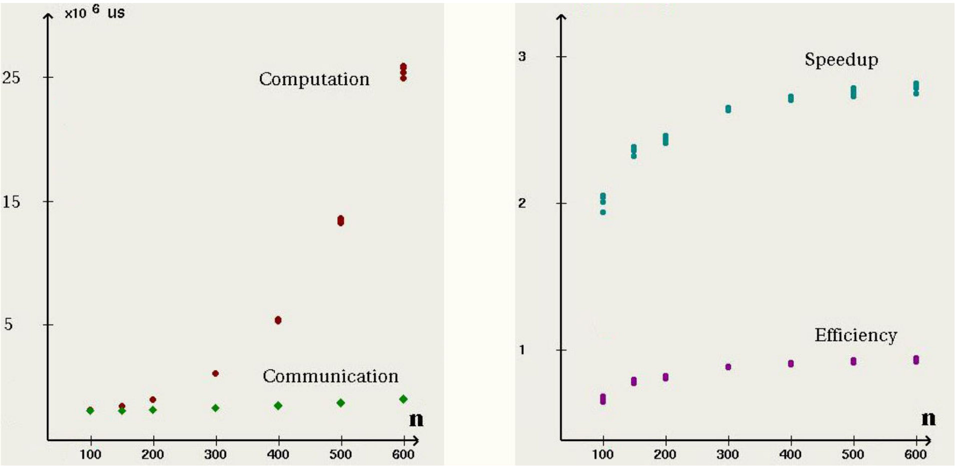


Figure 6.8: Performance of Parallel Matrix Multiplication in function of worksize (with $p = 3$).

The influence of other system parameters, such as clock frequency and memory size, or application parameters, such as the datatype used for the datastructure, can be studied similarly.

6.2 Tool

Between 2000 and 2004 a tool for the performance analysis of parallel application was developed at the parallel lab of the VUB, by Jan Lemeire, John Crijns and Andy Crijns [Lemeire et al., 2004][Lemeire, 2004]. The tool is called **EPPA**, which stands for **E**xperimental **P**arallel **P**erformance **A**nalysis. Experimental data is gathered through the profiling of parallel runs. The post-mortem analysis is based on the performance metrics developed in the previous section. The goal of EPPA is to support the developer of parallel applications with an easy and clear analysis.

6.2.1 EPPA

The EPPA analysis is based on traces of the execution of a parallel program: every phase is identified together with the important characteristics of each phase. The tracing is performed automatically when the program uses MPI [Snir et al., 1996]. MPI, the Message Passing Interface, defines the standard for writing parallel programs based on *message passing*. Parallel processes communicate by exchanging messages. The other approach

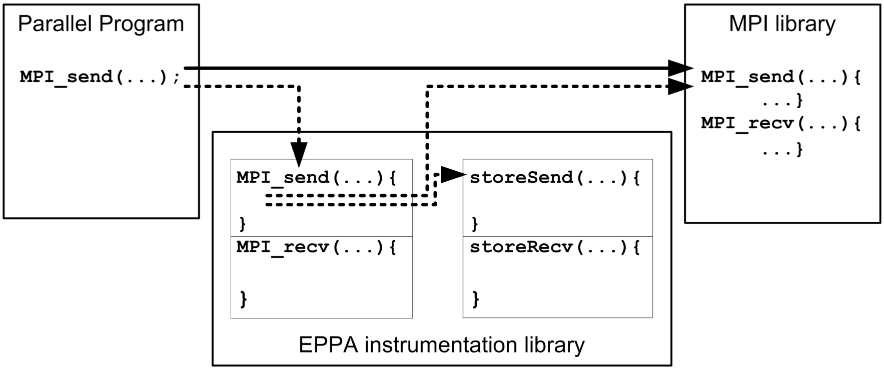


Figure 6.9: The MPI profiling interface to intercept MPI calls (dashed lines).

is *shared memory*, according to which memory may be simultaneously accessed by the different parallel processes.

The MPI profiling interface makes it easy to intercept the MPI calls made by a parallel program. How this works is shown in Fig. 6.9. After linking an MPI program with the EPPA library, each MPI call, before going the MPI library, is intercepted by the library. The information about the MPI operations and their durations are stored in the EPPA database. By this, EPPA collects information about the communication operations: when messages are send, at what times they arrive, when and how long a process is waiting for an incoming message, etcetera. Four phases are identified automatically: computation, sending, receiving and idling. The periods between successive MPI calls are stored as computation phases. What exactly the program is doing, useful work or overhead, cannot be detected automatically. The user has the possibility to clarify this by adding EPPA calls. The idling phases are the periods that a process is waiting for incoming messages. The user program has only to be linked with the EPPA instrumentation library to activate the tracing of all communication activity. This is shown in Fig. 6.10. Programs using the older PVM library for message-passing should be instrumented manually by adding an EPPA function call after each call to PVM.

The EPPA Tool presents the performance analysis in different views:

- The *timeline* shows the program execution of each process (Fig. 6.4).
- The overall performance gives speedup, efficiency and global overhead ratios (Fig. 6.5).
- The overhead ratios per process (Fig. 6.6).

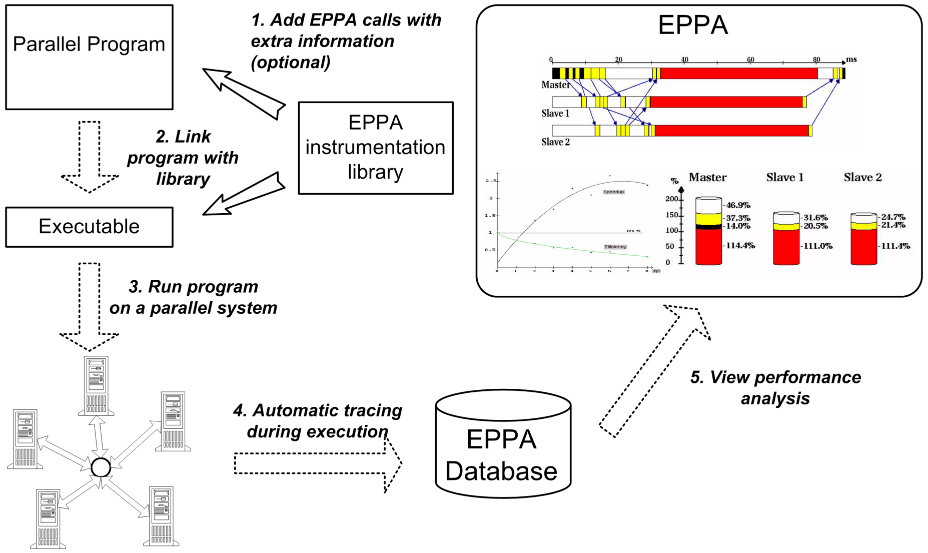

Parallel Program
1. Add EPPA calls with extra information (optional)
EPPA
EPPA instrumentation library
2. Link program with library
Executable
3. Run program on a parallel system
4. Automatic tracing during execution
EPPA Database
5. View performance analysis
0
20
40
60
80
ms
Master
Slave 1
Slave 2
2.5
2
1.5
1
0.5
0
1
2
3
4
5
6
7
8
#pr
Speedup
100 %
Efficiency
%
200
150
100
50
0
Master
Slave 1
Slave 2
46.9%
37.3%
14.0%
114.4%
31.6%
20.5%
111.0%
24.7%
21.4%
111.4%

Figure 6.10: Scheme of the EPPA tool.

- The performance variables in function of number of processors p or work size W (Fig. 6.7 and 6.8). Besides the visualization, a regression analysis can be applied on the displayed functions, returning the curve that best fits the data.

Besides the information about the program's communication that is collected automatically, the user is given the possibility to specify additional information. The data collected in this way facilitates the refinement of the analysis. EPPA provides the following options:

- The user can differentiate computational phases. EPPA automatically traces computation phases, as the cycles between two successive MPI calls. But it can not know whether these computations are part of useful work or overhead (control of parallelism). To make this difference, the user can add EPPA calls to specify the role of each computational phase.

- System parameter p and the program's work size parameter W are added for each experiment. Besides these, the user can add other system or program parameters. The performance variables can then be studied as a function of these parameters.

- The size of each message in bytes is automatically recorded by EPPA. The communication performance can be studied as a function of message size. Additionally, the user can specify the number of *quantums* that are processed and communicated in each phase. The definition of a quantum depends on the specific program. For a matrix operation, a quantum is an element of the matrix. EPPA provides the functionality to visualize performance metrics in function of the number of quantums.

- Finally, the main part of an algorithm usually is the repetitive execution of a basic *operation.* For matrix multiplication, the main computations consist of a multiplication and addition. The number of basic operations can also be passed to EPPA and studied in detail.

6.2.2 EPPA Measurement Overhead

Measurement implies interference. Measurement is impossible without adding overhead. In the case of tracing the MPI calls during parallel execution the overhead constitutes the measurement of the phase duration and the recording of all phase information. The phase information is kept in memory during execution and is only written to the EPPA database when the experiment is finished. The database calls do not have to be added to the overhead.

To verify the impact of profiling, the same experiment can be run with and without the tracing. The overhead was estimated with a test program in which the number computations and the size of communicated data could be varied. Experiments were run on our Linux computers with Athlon processors of 1.6GHz. The differences between the runtimes with and without tracing give an overhead per call between 4 and 5 μs. Time measurement has a precision of only 0.5 μs. This overhead, however, only has an impact on the performance for experiments with a low number of computations and small messages. Experiments show that the error on the parallel runtime is smaller than 5% if the messages are larger than 600 Bytes.

Concluding, coarse grain applications, in which large data sets have to be communicated that need many computations, the overhead that EPPA introduces can be neglected. For fine grain applications, with many small messages and short computation times, the overhead is not negligible. It must, however, be noted that when the MPI calls are equally distributed among the processes, the impact of the overhead on each process is the same. Each phase is equally extended with the same measurement overhead. The parallel runtime will be affected by the overhead, but not the execution profile. EPPA thus still offers a valid overhead evaluation. Except when one process performs much more MPI calls than the other processes. Finally, it must be noted that runtime and speedup can always be measured exactly by switching off the detailed phase tracing.

6.2.3 Related Performance Analysis Tools

XPVM (`http://www.netlib.org/utk/icl/xpvm/xpvm.html`) and XMPI (`http://www.lam-mpi.org/software/xmpi/`), tools that come together with PVM and MPI, automatically trace the computation-communication-idle phases of the execution profile. Fundamentally, EPPA does the same thing. But EPPA provides the possibility to the user to specify extra information. EPPA is mainly used for multiple experiment analysis, while XPVM and XMPI are limited to the analysis of a single experiment.

More advanced profiling tools like SCALEA [Truong and Fahringer, 2002a], Pablo [Reed et al., 1993], KOJAK [Mohr and Wolf, 2003a] or VAMPIR [Nagel et al., 1996] (semi-) automatically instrument the parallel program and use hardware-profiling to measure very detailed performance data. In the post-mortem analysis, they automatically filter out relevant parts (bottlenecks, situations of inefficient behavior, performance losses) from the huge amount of low-level information and try to map them onto the developers program abstraction. They rely on a detailed classification of overheads (see e.g. [Bull, 1996] or [Fahringer and Seragiotto, 2002]) and inefficiencies that are identified with an advanced search engine. Our research focuses on a multivariate analysis. Current tools that support multiple experiment analysis plot performance variables (SCALEA [Truong and Fahringer, 2002b]) and inefficiencies (Aksum [Fahringer and Seragiotto, 2002]) as a function of application and system parameters. Others, such as AIMS [Yan et al., 1995], additionally provide a regression analysis.

6.3 Summary of Chapter

This chapter introduced parallel processing and the performance metrics to evaluate parallel programs. For optimal performance, the occurrence of *lost processor cycles* during parallel execution should be avoided. The impact of an overhead, a source of lost cycles, on the speedup is quantified by its ratio with the sequential runtime. When considering the main phases of a parallel execution, overheads can be classified according to 4 types: control of parallelism, communication, idling and parallel anomaly. The tool EPPA can be used to automatically trace these phases during a run of a parallel program. The execution is visualized together with the different performance metrics. The user can augment the analysis by providing additional information about the parallel program.

The here developed performance analysis and tool allows the statistical analysis of experimental data retrieved from the execution of parallel programs. Qualitative statistical analysis, which will be applied to the data, is discussed in the next chapter.

Chapter 7

Qualitative Multivariate Analysis

SCIENCE endeavours to find laws relating to the world. The world consists of what we can observe about it[1]. Observation happens by signals that arrive to us. The sources of the signals are the 'things' we observe from the world. The 'things' have different 'states' at different moments[2]. By calling the things *variables* and the states the *values* of the variables, we talk about a *multivariate statistical analysis.* The goal of statistical analysis is to learn something from the observations about the system that generated the observations. Multivariate analysis is thus more or less basic to science. Causal inference is a statistical analysis which tries to reveal the underlying structure of the system under study.

Multivariate statistical analysis involves observation and analysis of more than one statistical variable at one time. At each observation the state of the variables is recorded. The observed data, called a **sample**, can be presented in a table:

	variable 1	**variable 2**	**variable 3**	...
experiment 1	7	10.45	white	...
experiment 2	3	0.655	black	...
experiment 3	2	34.93	red	...
...				

[1] "There could be more to the world, but if we cannot sense it, if we are not influenced by it, without interaction or without contact, we cannot say anything substantial about it," Frans Lemeire, my father (besides many other philsophers who say the same, but I learned it from him).

[2] Things having the same state at any moment, are boring and easy to study scientifically. Moreover, the question remains whether we would be conscious of such things.

I restrict myself to an analysis that is based on the assumption that the observations are **independent and identically distributed** (i.i.d.):

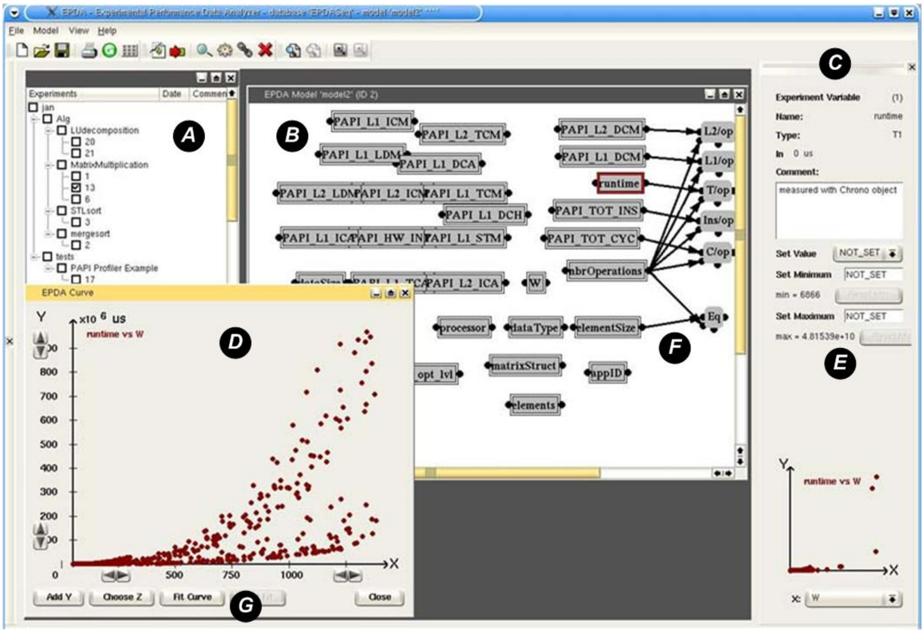
- Observations are mutually 'independent', an observation made at time t does not influence the outcome of an observation made at time t'.
- 'Identically distributed' means that observations made at different moments come from the same underlying probability denstity distribution. The system under study does not change during the experiments.

The i.i.d. assumption is true for data that is gathered during separate runs of programs. The execution of a program on a computer system can be assumed to be to a great extent independent of an execution that happened just before.

This chapter attempts to elucidate the practicalities of the multivariate analysis that will be employed in the next chapters. The focus lies on a qualitative analysis, i.e. the discovery of the relational structure among the variables, as explained in the first part of this work. The purpose is to get to understand systems by analyzing the data gathered during experiments. For this, the EPDA (Experimental Performance Data Analysis) tool was developed for analyzing performance data [Lemeire, 2006]. Besides its focus on performance data, it is applicable to any multivariate statistical analysis. It allows for easy recording, storage and visualization of the experimental data. For data analysis, it groups a set of statistical technologies, comprising the calculation of derived variables, regression analysis, kernel density estimation, outlier detection, conditional probability table compression and, last but not least, the extended causal structure learning algorithm (Chapter 5). A combined use of these techniques allows for the construction of qualitative and quantitative models. The chapter concludes with practical details about the causal inference algorithm, the use of mutual information as dependency measure and the calibration of the kernel density estimation algorithm.

7.1 EPDA

EPDA is based on the recording of experiments, during which values are attributed to the variables of interest. EPDA is written in C++, runs on Linux and the data is stored in a MySQL database. Per database, experiments are grouped in applications and applications in projects, allowing



Experiment Variable (1)
Name: runtime
Type: T1
In 0 us
Comment:
measured with Chrono object
Set Value NOT_SET
Set Minimum NOT_SET
min = 6866
Set Maximum NOT_SET
max = 4.81539e+10

Figure 7.1: Screenshot of EPDA.

the user to manage his experiments. The database also stores the properties of each variable: its type (categorical, discrete or continuous), if applicable its unit, whether the variable is an in- or output, if discrete or categorical its domain (the set of possible values) and additional comment.

Nominal or **categorical variables** have a domain which contains categories. Each value is chosen from a set of non-overlapping categories. A set of data is said to be **ordinal** if the values or observations belonging to it can be ranked (put in order) or have a rating scale attached. You can count and order ordinal data. Among the ordinal variables, the **discrete** ones only take on value of a countable set of numbers such as integers, while **continuous** variables can take on any real number in some finite or infinite interval.

7.1.1 Visualization Facilities

First of all, EPDA allows the user to explore the data, as shown in Fig. 7.1:

- (A) View and select experiments: the experiments are grouped per user, project and application.

- (B) View all variables.
- (C) View the selected variable's properties.
- (D) Plot data of one variable as a function of another and, optionally, in function of a third one.
- (E) Specify a fixed, a minimum or a maximum value for a variable. These will act as constraints for the data to be loaded.

The data that is loaded comes from the under (A) selected experiments and is filtered by the constraints on the variables, under (E). Together they form the query by which the data is retrieved from the database. This allows the analysis of specific experiments in a specific *context*. Limiting a variable to a certain value allows for the analysis of the data for that specific value only. For example, when the system's behavior depends on a categorical variable, when it changes qualitatively for different values. Filtering the data by specific values make it possible to analyze each behavior separately. The variable then determines the context. Another example is when a system's behavior changes when the value of a variable crosses a threshold. By filtering the data, the system under study can be analyzed separately for each context.

7.1.2 Statistical Technologies

The statistical techniques that can be applied to the observed data within EPPA are the following:

1. Creation of *derived variables*, variables for which the value is calculated from the values of others. The user selects the dependent variables and specifies the formula (chosen from a set of equations, comprising sum, product, quotient, comparison and equality). This is shown in Fig. 7.1(F) by new nodes which have the nodes of the dependent variables as inputs.

2. Calculation of the *partial derivative* of metric f. Consider f a function of parameters x, y and z. The partial derivative is the derivative of f with respect to one of those parameters, say x, with the others held constant, written as $\left(\frac{\partial f}{\partial x}\right)_{y,z}$. It quantifies the change of f by a change of x. The derivative is named df_dx. This functionality is not shown in Fig. 7.1.

3. Application of a regression analysis, *curve fitting*, for finding the best fit on the selected data in the chart. This is shown by letter (G) in Fig. 7.1. Regression analysis quantifies the relations. Curve fitting applied on the submodels allows the construction of an analytical model. In Section 2.4.2 the MDL approach was explained. The Java library, written by Dr Michael Thomas Flanagan (http://www.ee.ucl.ac.uk/∼mflanaga), is used for fitting functions on data. The routines return the closest fit of the given function according to the minimization of the sum of squared errors.

4. Learning the causal structure by *causal inference*: send the current data to the causal structure learning module of TETRAD. The user will be asked to select the variables on which the algorithm should be applied. The practical implementation of the algorithm is given in Section 7.3.

5. A *kernel density estimation* (KDE), which estimates the underlying continuous probability distribution from a given sample. The principles of KDE were discussed in Section 2.2.1, the algorithm to bring it into practice and its calibration is given in Section 7.3.3.

6. An *information analysis*: estimation of the classical information quantities such as entropy and mutual information. The calculation is based on the probability distributions estimated by the KDE. Practical application of the formulas is given in Section 7.3.2.

7. *Outlier detection*: set the outlier property of an experiment. Exceptional data can be identified by labeling the experiment. This is discussed in more detail in Section 7.1.3.

8. Apply a *conditional probability table (CPT) compression* on a discrete variable. This will be explained in Section 7.1.4.

Although the focus of our research lies on causal inference, the other techniques will show their value in combination with causal learning.

The outcome and relevance of a multivariate analysis relies on the observed or calculated variables. Techniques 1, 2 and 8 define new variables that are added to the data. These new variables possibly identify interesting properties and provide additional insight into the system.

7.1.3 Outlier Detection

Exceptions or **outliers** deviate so much from other observations that they arouse suspicions, in the sense that they could be generated by a different mechanism than the assumed behavior. Outliers occur frequently in experimental data, due to accidental, temporal, unexpected non-uniformities in behavior of certain components of the system. This is a fortiori true in the field of parallel processing as interactions between system resources are much more dynamic than in computers of the sequential (Von Neumann) type.

Outliers greatly affect the modeling, since they lie beyond the normal regime. A curve fitted on data containing outliers will be distorted. As the square of the distance is taken as a score, outliers contribute disproportionally to the error sum as they lie far from the actual curve. Their part in the error term will be much greater than that of other points, so that they greatly affect the result of the fitting. Outliers should therefore be excluded from a quantitative analysis.

A lot of research is done on identifying outliers in data, using distances to the mean value [Rousseeuw and van Zomeren, 1990], iterative methods [Chen and Liu, 1993], clustering techniques [Arshad and Chan, 2003] or information-theoretic approaches [Pynnonen, 1992]. None of these methods are currently integrated in EPDA. The user is required to identify the outliers by manual inspection of the data. The visualization capabilities of EPDA greatly facilitate this task. For these data points, the user activates the 'exception' flag of the experiment the datapoint comes from. The flag is a boolean variable initially set to 'false'. It can be used to filter the data (removing the exceptions) or alternatively search for possible explanations of the exceptions, which can be performed by a causal analysis.

7.1.4 Probability Table Compression

In a causal model each variable is generated by its direct causes, which are its parents in the graph. The relation between a node, for example Z, and its parents (denoted $parents(Z)$) can exhibit striking regularities, called local structure. Sometimes it is useful to model these regularities explicitly. If the parents of Z are discrete variables, the conditional probability distribution (CPD) $P(Z \mid parents(Z))$ is stored in a table, called a conditional probability table (CPT). The table stores one value for each combination of the elements of the domains of $parents(Z)$ and Z. Regularities among the values allow compression of the table and reduction in the description length of the table.

The algorithm will group values of $parents(Z)$ that are associated with the same probability distribution over Z. These groups are labeled by a new variable (Fig. 7.2). Some values of X exert the same influence on Z. These resemblances are valuable information, worth of being added

X	P(Z=1| X)	Y
X_0	0	Y_0
X_1	1	Y_1
X_2	1	Y_1
X_3	0	Y_0
X_4	1	Y_1

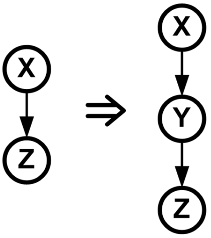


Figure 7.2: Reduction of a conditional probability table and insertion of a new variable.

explicitly to the model. This is accomplished by introducing a new variable Y, which takes one value for each group of values of X having the same $P(Z \mid X)$. The influence of X on Z is then completely captured by Y. It can replace X in the model as a direct parent of Z. The relation between X and Y is not of causal nature, but logical. Y contains redundant information. Nevertheless, the addition of the deterministic variable is useful, it provides more detailed precise information about the influence of X on Z. *Y characterizes X with respect to Z.* The relation of Y with Z is simpler, and in many cases the variable corresponds to a meaningful quantity. Y represents a property of X. Take the relation of the datatype in a computation with the processor instructions. The *double* and *float* datatype require exactly the same number of instructions on some modern processors.

If only one value of a discrete variable of $parents(Z)$ gives a different distribution on Z, this is modeled by adding a boolean variable that is true for the specific value. This new variable will replace the discrete variable in $parents(Z)$. Note that similar work goes further in looking for regularities in conditional probability tables [Boutilier et al., 1996].

7.1.5 Sequential Experiment Probing

The EPDA library provides a probe object, written in C++, to monitor runs of sequential programs on a single computer and write the data to the EPDA database [Lemeire, 2006]. Each experiment measures the following variables:

- runtime (T_{comp}),
- date of experiment,
- computer's properties: name and clock frequency,

- values of the hardware counters: the standard PAPI library [Browne et al., 2000] provides access to a processor's hardware counters: at the end of each experiment the EPDA Probe reads a processor dependent number counters. The most interesting are:
    - PAPI_TOT_CYC: total number of cycles, it corresponds to T_{comp}/f_{clock}.
    - PAPI_TOT_INS: total number of instructions executed.
    - PAPI_L1_DCM: level 1 data cache misses.
    - PAPI_L2_DCM: level 2 data cache misses.
    - PAPI_L1_DCA: level 1 data cache accesses.
    - PAPI_L2_DCA: level 2 data cache accesses.
- Additional variables of interest, specific to the program, are user-defined:
    - algorithm parameters, like the work size W,
    - algorithm variables, such as the number of basic operations or iterations of the main loop of the program,
    - system parameters and characteristics, such as the memory hierarchy sizes.

7.1.6 Loading Parallel Performance Data

The EPPA tool, discussed in the previous chapter, is used to track experiments with parallel programs. EPPA is focused on providing the user a visualization of the experimental outcomes. For detailed statistical analysis, experimental data is transferred from the EPPA database to the EPDA database, as shown in Fig. 7.3. For example, the computation time of an experiment or the delay of an individual message. The former results in 1 EPDA record per EPPA experiment, the latter in 1 EPDA record for each communication operation that happened during the EPPA experiment. I come back to this in the following chapter, when the experimental setups are discussed.

7.2 The Modeling Process

Fig. 7.4 shows the different parts of the modeling process. The EPDA tool offers the statistical facilities in a semi-automatic way, the user guides the process by choosing the sequence of tasks to be executed.

The modeling steps are:

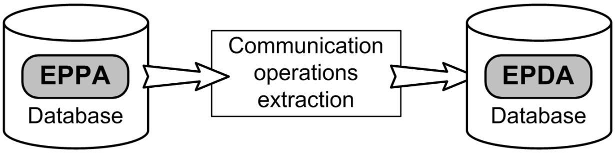


Figure 7.3: Extracting performance data from EPPA into EPDA.

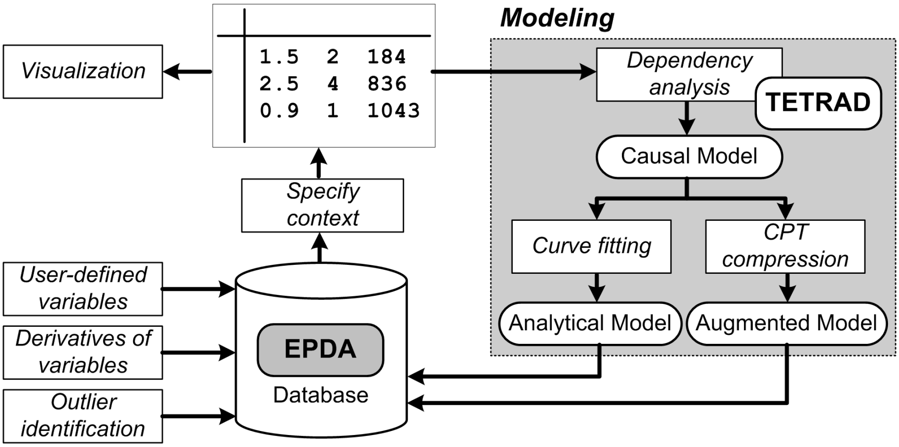


Figure 7.4: Scheme of the modeling process.

- The user chooses the context of the data that has to be loaded: which experiments and for which parameter values.
- Data can be inspected by the visual facilities of EPDA.
- The addition of interesting derived variables or partial derivatives.
- The user identifies outliers. For these experiments, the exception flag is activated. This flag can be used to clean up the data by filtering the outliers. On the other hand, for studying the outliers, one can detect which variables affect the exceptions.
- TETRAD's causal structure learning algorithms are used reveal the qualitative, causal model.
- Curve fitting applied on the qualitative model results in an analytical model. The quantification of submodels with a discrete variable produce multiple curves, one for each discrete value of the variable. This is called parameter identification in statistics. It is not the main purpose of our work.
- Compression of the conditional probability tables (CPTs) leads to the addition of variables that refine the model.

7.3 Causal Structure Learning

EPDA is linked to TETRAD which contains the causal structure learning algorithms. The current algorithms, however, had to be extended in two ways to cope with the full complexity of performance data:

- *The presence of a mixture of categorical, discrete and continuous data.*
- *The presence of non-linear relations.*
- *The presence of deterministic relations.*

Most research on causal learning does not consider models that contain such a wide variety of variables and relations. They focus on one type of variable, categorical/discrete or continuous, where the continuous variables are most often expected to be approximately linearly related with Gaussian or non-Gaussian disturbances [Spirtes et al., 1993] [Shimizu et al., 2005]. Currently, TETRAD applies the Pearson correlation coefficient for measuring association amongst continuous variables and the G^2 test for categorical or discrete variables. The analysis of data with a mix of both types is not supported. Moreover, Pearson's correlation coefficient is only correct for linear or quasi-linear relations among the variables, it measures the proximity of a relation to linearity (Section 3.1.1).

In case of deterministic relationships, one should exclude variables from the dataset that are definable in terms of other variables in the set [Scheines et al., 1996]. As discussed in Chapter 5, deterministic relationships imply additional conditional independencies that cannot be captured by faithful models. Current constraint-based learning algorithms also fall short when applied on them. In Section 5.2.1 was argued that deterministic variables contain valuable information, worth being left in the model.

These limitations were overcome by extending the current algorithms in two ways:

- the independency test employed in the dependency analysis is based on the information-theoretic concept of *mutual information* (defined in the introductory chapter). It measures the decrease of uncertainty of one variable when observing other variables. This solves two problems: independencies containing a mixture of discrete and continuous variables can be quantified and it offers a form-free dependency measure.
- TETRAD's PC algorithm was extended to discover information equivalences and adding them to the models. The extensions for handling information equivalences were discussed in detail in Chapter 5.

The alternative independence test is discussed in the next subsections. But first I discuss the assumptions of the PC algorithm and the utilization of background knowledge. Finally it must be noted that the *depth* of the algorithm is set to 2.

Validity of Assumptions

Section 3.4.2 listed the assumptions under which the PC algorithm is known to learn correct models. The two most important assumptions are faithfulness and causal sufficiency.

The existence of a faithfulness graph is broken by the presence of deterministic relations. This problem was solved by proving that faithfulness could be reestablished by adding the information of deterministic relations to an augmented causal model. Causal sufficiency, or the absence of an unknown common cause, is guaranteed in a performance analysis by including all parameters of algorithm and system to the data. They constitute the ultimate causes of all performance-influencing variables.

Background knowledge

The construction of performance models is in some ways simpler than those of other domains since a lot of background knowledge is available. This information is given to the learning algorithm as forbidden or required edges and known orientations.

- Performance data is a result of what in statistics is called a set of *controlled experiments*: at the start of each experiment, the input variables are set to specific values. While modeling, input variables are not connected with each other and they only have outgoing edges. They do not have causes, they are assumed to be the ultimate causes of the other variables.
- Performance models aim at understanding some performance metrics. These variables can be regarded as output variables as they only have incoming edges.
- While not all edges can a priori be guessed by the user, the orientation of the edges is in most cases easy to determine. Any case of doubt by the learning algorithm can be solved by the user's knowledge.
- In fact, a lot of dependencies and independencies are known a priori. They can also be passed to the algorithm as background knowledge.

7.3.1 Pearson Correlation Coefficient

Causal inference is based on conditional independencies, which have to be estimated from the data. The **Pearson product moment correlation coefficient** is widely used to measure the degree of association between two continuous random variables. For variables X and Y, it is defined as

$$r_{XY} = \frac{\text{cov}(X,Y)}{\sqrt{\text{var}(X)\text{var}(Y)}} \tag{7.1}$$

where cov() measures the covariance between two variables and var() the variance of a continuous variable:

$$\mu(X) = \frac{1}{n}\sum_i^n x_i \tag{7.2}$$

$$\text{var}(X) = \frac{1}{n-1}\sum_i^n (x_i - \mu(X))^2 \tag{7.3}$$

$$\text{cov}(X,Y) = \frac{1}{n-1}\sum_i^n (x_i - \mu(X))(y_i - \mu(Y)) \tag{7.4}$$

$\mu()$ gives the mean value of a statistical variable. Pearson's coefficient gives a measure of how close a relation approximates linearity. The standard independence test of TETRAD is based on the comparison of Pearson's coefficient with a threshold value, by default set to 0.05.

Conditional independencies are measured by partial correlations. They can be calculated directly from the correlation coefficients, but only if linearity holds. The first-order correlation of X and Y given Z is defined as:

$$r_{XY.Z} = \frac{r_{XY} - r_{XZ}r_{YZ}}{\sqrt{(1 - r_{XZ}^2)(1 - r_{YZ}^2)}} \tag{7.5}$$

Higher-order coefficients are defined similarly. All dependencies can thus be calculated from the correlation matrix, where the **correlation matrix** is the $n \times n$ matrix whose i, j entry is the correlation of the variables with indices i and j. For independence, the same threshold of 0.05 is used.

Correlations lead to good results for quasi-linear associations. Partial correlations, however, fail if the relations diverge too much from linearity. This was confirmed by our experiments.

7.3.2 Information-Theoretic Dependence

Mutual information gives a form-free measure of association. The *mutual information* quantifies the decrease in uncertainty of a variable when knowing a set of other variables. Association is measured in terms of the information that variables share. This is independent of the form of the relation, whereas Pearson's correlation coefficient cannot distinguish between the deviances of linearity and the uncertainty of the relation (explained in Section 3.1.1).

Recall from classical information theory (discussed in Section 2.1) that the mutual information can be written as

$$I(X;Y) = \sum_{x \in A} \sum_{y \in B} p(x,y) \log_2 \frac{p(x,y)}{p(x)p(y)} \tag{7.6}$$

Conditional information, $I(X;Y \mid Z)$, is calculated similarly. The G^2 **independence test** [Bishop et al., 1975], used by the TETRAD tool for identifying independencies among discrete variables, is defined as:

$$G^2 = 2 \sum P_{observed} . \ln \frac{P_{observed}}{P_{expected}} \tag{7.7}$$

It is of the same form as Eq. 7.6. The nominator, $p(x,y)$, stands for the observed joint distribution, while the denominator, $p(x)p(y)$, stands for the mutual probability in case both variables are independent.

For computing the entropy of continuous variables, their distributions have to be discretized. For a variable X with density $f(x)$ and division of the range of X in bins of length Δ, the entropy of the quantized distribution is [Cover and Thomas, 1991]

$$H(X^{\Delta}) = -\sum_{-\infty}^{+\infty} f(x_i)\Delta \log_2(f(x_i)\Delta) \tag{7.8}$$

The discretized definition of mutual information is then of the same form as Eq. 7.6. After discretization, the handling of continuous and discrete variables is identical. This allows for the calculation of dependencies for a mixture of continuous and discrete variables. The difference with Pearson's correlation coefficient is that mutual information considers each value of X independently: it sums for every value of X the decrease in uncertainty of Y. The relation between X and Y may be arbitrary. Whereas Pearson seeks for a linear trend in the (x, y) points over the whole range of X.

The main disadvantage of mutual information is that larger sample sizes are necessary. The definition suggests that for every value of X multiple data points are needed. This approach would be the same as discretizing the continuous variable and would also induce a quantization error. Kernel density estimation overcomes these problems. The reason is that $P(X = x)$ will also be influenced by the data points in the neighborhood of x, so that the sample size can be limited.

7.3.3 Independence Test Calibration

The causal inference algorithm relies on the correct qualification of the independencies between the variables. The independence test must measure the dependency strength between two variables, and use a threshold below which the variables are classified as being independent. Dependency strength is based on the mutual information calculated from an estimation of the underlying distribution of the variables. If the mutual information falls below a *threshold*, the variables are classified as being independent.

The estimation of the underlying distribution uses kernel density estimation (KDE), discussed in Section 2.2.1. I chose the width of the kernel, called bandwidth, to be proportional to the average distance of each data point to its closest neighbor. The exact proportion is called the *bandwidth factor*. It is the main parameter. I performed the calibration of this parameter with a series of about 300 independence tests for which the result is determined by manual inspection of the data. I selected a set of tests that cover the whole range of possibilities: continuous or discrete

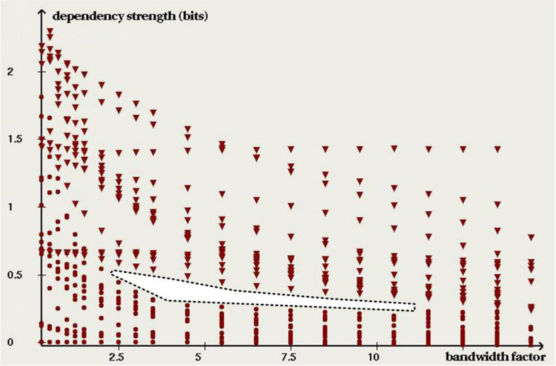


Figure 7.5: The KDE dependency strength between variables as a function of the KDE bandwidth factor for 300 experiments. The circles denote independent variables, the triangles dependent variables.

variables, mixtures of both and by conditioning on one or more variables. Fig. 7.5 shows the experimental results of the dependency strength by varying the bandwidth factor. The points denoted by circles represent tests of independent variables, the quadrangles dependent variables. An increased bandwidth factor smoothens the distribution, hence the mutual information decreases. A small bandwidth factor results in high dependency values, even for independent variables. The estimate then consists of narrow non-overlapping kernels centered on each datapoint. The bandwidth factor should be chosen between those two extremes. Fig. 7.5 shows that a bandwidth factor between 3.5 and 5 gives a maximal difference in dependency strength for positive and negative tests. The independence threshold is set to 0.35.

Fig. 7.6 shows the influence of the number of experiments (the sample size) on the accurateness of the dependency estimation. For estimations based on more than 100 points, the dependency strength can differentiate independent from dependent variables.

7.4 Summary of Chapter

This chapter was devoted to the technologies offered by our EPDA tool for the statistical analysis of experimental data. Although that the tool is generic, special attention is made for data about the performance of sequential programs, executed on a single computer, or parallel programs, executed on a collection of connected computers. The latter is traced by the EPPA tool, as explained in the previous chapter. EPDA allows

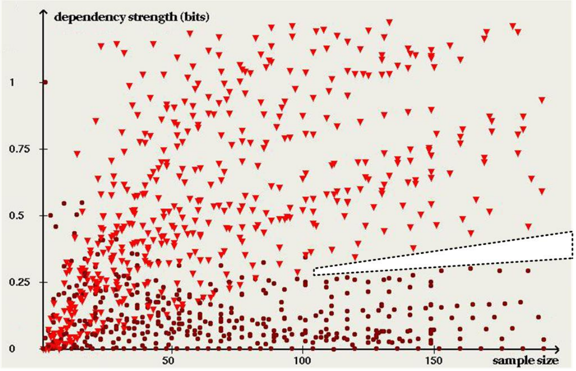


Figure 7.6: The KDE dependency strength between variables in function of the sample size. The circles denote independent variables, the triangles dependent variables.

you to organize, filter, visualize and analysis of your data. Among the various statistical techniques that EPDA offers, the focus lies on the causal inference algorithm. The tool will be used in the next two chapters to gather and analyze experimental data about the performance of parallel and sequential programs. I will show the benefits of causal inference.

Chapter 8

Causal Inference for Performance Analysis

THIS chapter researches how causal models and causal inference, both thoroughly discussed in Part I, can be beneficial for the performance analysis of applications. Basically, the user analyzes an application by measuring the states of various variables during various experiments as explained in the previous chapter. The variables comprise parameters, performance metrics, system counters (such as cache misses), program counters (such as the number of iterations of a loop instruction), user-defined variables, etc. The causal learning algorithms will reveal the relations among the observed variables. The relevance of the analysis is therefore limited to the set of defined and observable variables.

The following properties of causal inference will prove their utility:

1. The construction of causal models is based on a *dependency analysis*: causal models reveal how variables relate.
2. Causal model theory is based on the *Markov property* (defined in the introductory chapter): recall that for Markov chain $A - B - C$, variable A affects C, but B 'screens off' the influence of A on C. A becomes independent from C by conditioning on B; if the state of B is known, learning about A's state offers no additional information about C.
3. A causal model corresponds to a *decomposition*: independent submodels are identified by a causal analysis.

This chapter begins with an overview of the potential benefits of causal models and causal inference to the analysis of the performance of computer programs. After comparing my approach with related work, causal performance models are defined. It is investigated to what extent the models can be labeled as describing relations of ‘causal’ nature. Next the results of the analysis of data from experiments with sequential and parallel programs are presented. Three different algorithms are studied: a standard LU decomposition algorithm, the Kakadu implementation of the JPEG-2000 standard for image compression and a differential equation solver of the Aztec benchmark library. Overall performance is studied, the communication overhead of parallel programs and explanations for exceptional data are sought. The results prove that relevant models are learned showing the influence of all relevant variables on the performance.

8.1 Utility

The utility of causal performance models and the learning algorithms is twofold: they support the modeler in building performance models as well as the user in understanding the performance of his application.

8.1.1 Support of the Modeling Process

Model Construction

The learning algorithms provide the ability to construct models from experimental data. This is useful in situations when the relations among variables are not a priori clear or can be used to discover unexpected dependencies.

Model Validation

Any analysis is explicitly or implicitly based on independency assumptions made by the modeler. The dependency analysis allows the verification of these independencies.

Reuse of Autonomous Submodels

Causality takes the position that the world is reducible, that we can learn something about the whole by studying the parts. In chapter 4 was shown that causal models can be decomposed into independent, local submodels of the form $P(X_i \mid pa_i)$. Causality thus inherently provides autonomous submodels that can be reused when modeling systems in a different context.

Reveal Performance Characteristics

For being able to reuse submodels for performance modeling of systems independent of the application (previous subsection), the submodels must consist of application-independent characteristics. The Markov property enables the validation of performance characteristics of applications. Such quantities aim at fully characterizing the performance behavior of the application so that they can be used to generically explain and predict performance:

$$application \rightarrow characteristics \rightarrow performance. \tag{8.1}$$

The *characteristics* should hold all performance information of the application. This property is captured by independence *application* $\perp\!\!\!\perp$ *performance* | *characteristics*.

Reveal Explanations

A performance analysis should not only provide models for performance prediction, but also provide insight in the performance results. The Markov property enables to find and validate explanations for performance issues. The requested models should be of the form

$$application \rightarrow explanations \rightarrow performance. \tag{8.2}$$

The *explanations* should hold all performance information of the application, which is characterized by independence $application \perp\!\!\!\perp performance \mid explanations$. The right part of the model is then generic - reusable in the analysis of other applications. It provides insight into the features that determine performance.

Combination of Equations 8.1 and 8.2 results in:

$$application \rightarrow characteristics \rightarrow explanations \rightarrow performance. \tag{8.3}$$

The application characteristics determine *what* influences the performance. The explanations tell *how* they affect the performance.

Flexibility

Flexibility of the causal modeling framework enables the refinement of models by adding variables of interest. They can provide further explanation and deepen the insight. Or they can provide additional information by which performance can be predicted with a higher accuracy. These new variables were latent (unknown) variables in the initial models. Assume initial causal relation $A \rightarrow B$ and that variable C is added. The additional variable can intervene with the causal relation in 3 possible ways:

1. C is an effect of A or B, e.g. $A \rightarrow B \rightarrow C$. The variable is a new metric which is explained by the original variables.

2. C provides a deeper explanation for B, the model becomes $A \rightarrow C \rightarrow B$. A becomes an indirect cause of B. We can say that the model is *refined*.

3. C is a cause of either A or B, resulting in new models respectively $C \rightarrow A \rightarrow B$ or $A \rightarrow B \leftarrow C$. The states of A and B are explained by an until then unknown variable. However, a common cause of A and B would cause a problem for the initial model. The presence of latent common causes is not supported by the PC algorithm (Section 3.4.2). Fortunately, all ultimate causes are known in a performance analysis, namely the parameters of application and system. This case can therefore be excluded.

8.1.2 Presentation of a Clear Performance Report

Causal models explicitly encode relational information. They offer a formalization of a structured representation of the relations among the variables of interest. They provide insight into complex situations with many variables and dependencies.

Structuring the Variables

In order to understand complex situations with many variables and dependencies, a structured representation of the relations is required. This is offered by causal models.

Filtering Relevant Information

Causal models correspond to physical mechanisms and enable the filtering of relevant information, by the statistical analysis, which reveals the impact of every factor, the most influential factors can be filtered.

Reasoning under Uncertainty

At the start, a model can be kept simple, as an approximation with a limited number of variables. Additional information can easily be integrated into the model later. Probabilistic models, such as Bayesian networks, were developed to reason under uncertainty, to work with incomplete information. They estimate the information of variables when knowing some other variables: $P(\textit{unknown variables} \mid \textit{state known variables})$.

Qualitative Reasoning

Besides the explanatory facilities, causal models can be exploited to *reason* about the performance. They provide answers to questions like: *Do the cache misses affect the runtime of a program? Which program parameters affect the cache misses? Do the chosen data structures in the program affects the cache misses? Is the knowledge of the size of the data structures sufficient for predicting the cache misses?*

8.2 Related Work

Causal models are not widely used for performance analysis yet. Only one reference could be found. Cohen and Chase use Tree-Augmented Bayesian Networks (TANs) to identify combinations of system-level metrics and threshold values that correlate with high-level performance states in a three-tier web service under a variety of conditions [Cohen et al., 2004].

Many tools exist for automated performance analysis. They are integrated in frameworks for coordinated monitoring and control of computer applications. The complexity of deployed systems surpasses the ability of humans to diagnose and respond to problems rapidly and correctly. Research on automated diagnosis and control - beginning with tools to analyze and interpret instrumentation data - should provide the means to guide the developer and user with understandable information. Our research focuses on exploiting the statistical learning techniques in cases that models or relations are not a priori known. This approach assumes little or no domain knowledge, is therefore generic and has the potential to adapt to changes in the system and its environment. The most common approach is to incorporate a priori models, which explicitly or implicitly represent how variables relate to each other. Other approaches let the user himself discover the interrelational structure incrementally. These approaches have several limitations: the models are difficult and costly to build, they may be incomplete or inaccurate in significant ways, and inevitably become brittle when systems change or unanticipated conditions are encountered.

Most performance monitoring tools fall under two categories. One collecting statistical data from multiple experiments, *concerned with counts* *and durations.* The other category is based on event traces, *the exact sequence of actions that took place during a run is recorded.* Statistical data is more compact than that of event traces, yet the predictive power is limited [Carrington et al., 2005]. Our approach is only applicable to the first category. It is based on a multivariate analysis that tries to characterize the relations among the data. The difference with current work, is that they work with relational structures that are a priori chosen or have to be configured manually. Current tools that support multiple experiment analysis plot performance variables (PMaC [Snavely et al., 2002], PERFORM [Hey et al., 1997] and SCALEA [Truong and Fahringer, 2002b]) and inefficiencies (Aksum [Fahringer and Seragiotto, 2002]) as a function of application and system parameters. Several tools allow for automatic bottleneck detection. Examples are KappaPi [Espinosa et al., 1998], Kojak [Mohr and Wolf, 2003b] and Paradyn [Karavanic et al., 1997]. They detect patterns in statistical data or event traces. Our approach cannot detect patterns, since it only analyzes variables once they have been measured or derived from others. Once an interesting pattern is defined, it can be added to the model as a variable. Causal inference can subsequently be used to find possible causes or effects of the pattern. It reveals which variables influence or are influenced by the occurrence of the pattern.

8.3 Causal Performance Models

The causal structure learning algorithms applied on data gathered from performance experiments result in 'causal performance models'. In this section I analyze what models can be expected and the validity of the causal interpretation.

8.3.1 Definition

Causal performance models are defined as causal models over a set of variables relevant in a performance analysis. Take the following example of a simplified performance model of a LU decomposition algorithm.

Example 8.1 (LU decomposition runtime performance model).

> *LU decomposition* is a matrix decomposition which writes a matrix as the product of a lower and upper triangular matrix

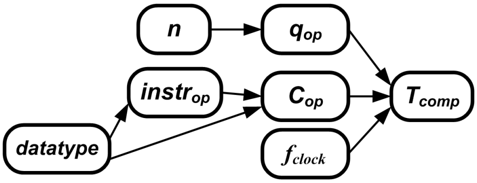


Figure 8.1: Simplified causal model of the performance of the LU decomposition algorithm

[Horn and Johnson, 1985]. For a 3 × 3 matrix, this becomes

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{12} & 1 & 0 \\ l_{13} & l_{23} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \tag{8.4}$$

This decomposition facilitates solving a system of linear equations, $Ax = b$, or finding the inverse of a matrix. The sequential implementation of the algorithm mainly consists of 3 nested for-loops with the inner loop having a division, a multiplication and a subtraction of matrix elements. The number of iterations of each loop is proportional to n, with n the row and column size of the matrix. Only square matrices are considered. The instructions of the inner loop form the main computations that have to be performed. The total number of times they are executed is proportional to n^3. This is identified by q_{op}, the number of basic operations.

The performance model shown in Fig. 8.1 is constructed in an intuitive way. It represents a simplified model of the computation time T_{comp}. The *datatype* of the matrix ($float$, *double*, *integer*, ...) influences the number of instructions $instr_{op}$ needed to perform 1 basic operation, but also C_{op}, the number of processor cycles needed for 1 basic operation. Together with the processor's clock frequency f_{clock} and q_{op} they determine the runtime T_{comp}.

The model is a *first-order approximation.* It gives a good indication for the computation time, but fails to characterize the full complexity of the computational process. It therefore should be regarded as a probabilistic model. The relations are probabilistic. They are not able to exactly determine the computation time.

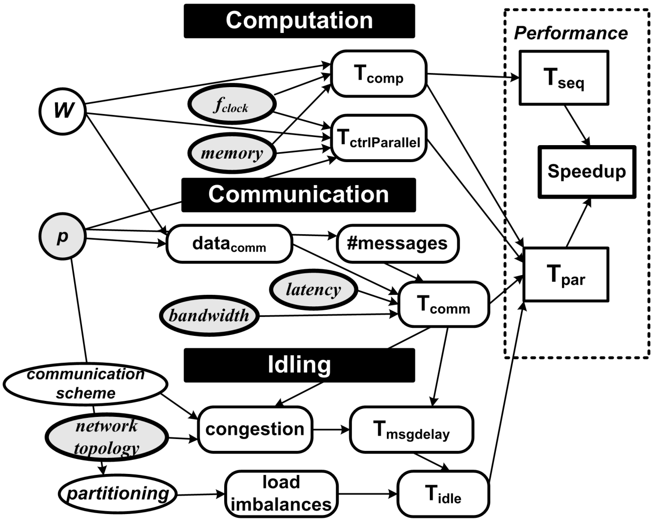


Figure 8.2: General model of parallel performance

8.3.2 Parallel Performance Models

Based on the metrics defined in Section 6.1, a causal model of the variables involved in a parallel performance analysis can be constructed. It is shown in Fig. 8.2. As application parameter, the work size W of the application is shown. System parameters, like the number of processors p and the processor's clock frequency f_{clock}, are depicted. Overall performance variables, sequential runtime T_{seq}, parallel runtime T_{par} and speedup, placed at the right side constitute the output of the model. Intermediate variables like the runtime for control of parallelism $T_{ctrlParallel}$, communication time T_{comm} and idle time T_{idle} measure the overheads. Other variables in their turn explain the values for the overheads. Most of them can be measured in the application, in the system (like the cache misses) or by the MPI profiling capabilities (like the size of the communication data or the number of messages). Finally, the partitioning and the communication scheme denote performance characteristics of the application.

8.3.3 Causal Interpretation

The question arises whether multivariate models about performance can be interpreted as causal models. Spirtes et al. use 3 properties as an axiomatic foundation of causal models: the Causal Markov condition, minimality and faithfulness [Spirtes et al., 1993] (Section 3.4.2). The Causal Markov condition consists of the Markov condition and the causal interpretation of the relations. Chapter 4 showed that causal models basically correspond to a decomposition into independent submodels $X = P(parents(X))$. Let us analyze whether these 5 properties apply to performance models:

1. *Markov condition*: A variable is expected to be determined by its parents only.

2. *Causal interpretation*: The causal interpretation of the edges should be read as "a change of the state of A causes a change of the state of B".

3. *Minimality*: The model is assumed not to contain redundant relations.

4. *Faithfulness*: The paths between the variables show the dependencies among the variables. This corresponds to the faithfulness condition. If the alternative definition is used, $faithful_{eq}$, it incorporates deterministic relations.

5. *Independence of submodels*: As will be shown by example 8.4.4, the genericity of submodels is limited. The validity of a submodel depends on the specific application. Nevertheless, similar submodels will reappear across different applications. Hence, submodels are only partially independent, they depend on the context.

One might argue that not all relations among the variables are of causal nature. Consider the deterministic relation *datatype* → *elementsize*, *elementsize* is a property of *datatype*. Nevertheless both variables represent different quantities. The size in bits of a datatype can vary across different platforms. Also variables that rely on a definition can be called into question. Take the definition of efficiency, $Efficiency = Speedup/p$. The relation cannot be interpreted as a causal relation. Here, *Speedup* must not be seen as a cause of $Efficiency$ (or vice versa). They are both affected by other variables, those of application and system that determine the overall performance. *Speedup* and $Efficiency$ share these causes.

8.4 Experiments with a Sequential LU Decomposition Algorithm

The first series of experiments deals with the performance analysis of the LU decomposition algorithm, which was explained in Example 8.3.1. 125 sequential experiments were run on a standard off-the-shelf computer with

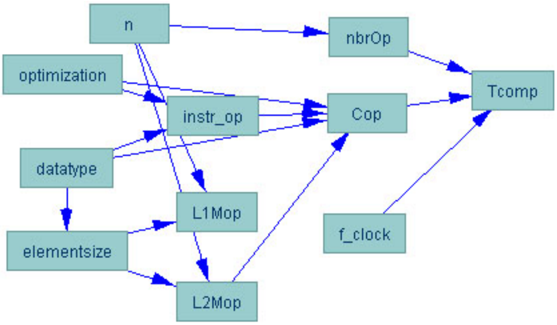


Figure 8.3: Detailed performance model of LU Decomposition.

an AMD Athlon XP processor running Linux and by performing a parameter sweep over the following parameters:

- Matrix size n, ranges from 5 to 300.
- *datatype*: short (2 Bytes), int (4 B), float (4 B), double(8 B), long (8 B) and longdouble (12 B).
- *optimization*: the inner loop of the calculation can be optimized. The result of a division that is performed twice is stored in a temporary variable so that it does not have to be recalculated the second time.

The EPDA probe was used to record the relevant performance variables (see section 7.1.5). The EPDA tool and its facilities were used to analyze the data and TETRAD for applying the extended PC algorithm on the data. As background knowledge for the algorithm, the input (parameters) and output variables (overall performance metrics and partial derivatives) were indicated.

8.4.1 Performance Modeling

Fig. 8.3 shows the performance model of second-order approximation. We see that only the level 2 cache misses L_2M_{op} have a non-negligible influence on the performance, not the level 1 cache misses. But even with these additional variables, *datatype* is still directly related to C_{op}. In other words, L_2M_{op} and $instr_{op}$ are insufficient to explain all processor cycles.

The model shows that *optimization* only influences the number of instructions, not the cache misses. But the number of instructions cannot characterize completely the effect on the performance. *Optimization* is

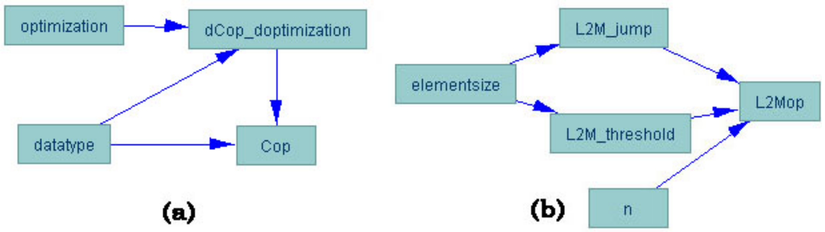


Figure 8.4: Parts of the performance model of LU Decomposition. Effect of the optimization (a) and modeling of the cache misses (b).

also related to the number of cycles C_{op}. This reconfirms that the number of instructions is not a good measure for explaining the overall performance. It is the number of cycles we have to look at. To investigate deeper the effect of the optimization on the performance, the partial derivative $\partial C_{op}/\partial optimization$ is calculated. It gives the decrease in number of cycles by the optimization. By adding this variable, we can learn on which variables the effect of the optimization depends. Fig. 8.4(a) shows that the decrease in cycles depends on the datatype. Quantitative results reveal it is about 50 cycles for integers, only 18 cycles for floats or doubles, but 110 for longs.

8.4.2 Model Validation

Dependency analysis also allows for the validation of models presumed by the user. The influence from *datatype* on C_{op} goes via L_2M_{op} and $instr_{op}$, as shown in Fig. 8.3. Omitting the link $datatype \rightarrow C_{op}$, however, would be incorrect. This can be verified by applying the Markov condition (see section 3.3) on node C_{op}: without the link, *datatype* would be independent of C_{op} given L_2M_{op} and $instr_{op}$. Yet, the independent test applied on the experimental data gives a mutual information $I(datatype; C_{op} \mid L_2M_{op}, instr_{op}) = 0.45$. This is above the threshold of 0.35, so Markov is violated and both nodes must be connected in the model.

8.4.3 Datatype Characterization

The goal is to characterize the influence of the discrete variable *datatype* on the performance. It turns out that the cache misses can be predicted with the size of the *datatype*, *elementsize*, but that the penalty cycles caused by a miss cannot be predicted by an application-independent feature.

Fig. 8.5 presents the experimental data of the level 2 cache misses in

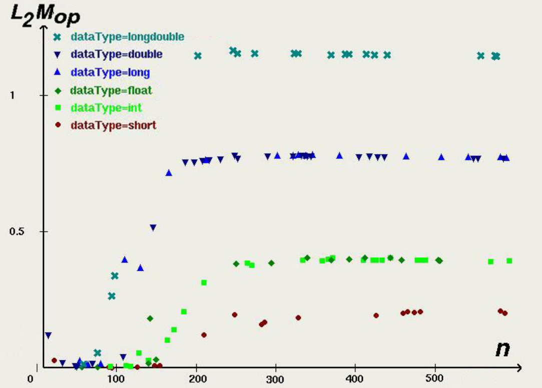


Figure 8.5: Experimental results of the LU Decomposition: level 2 cache misses versus matrix size n in function of the data type.

function of matrix size. It shows clearly how the misses jump to another level when the cache memory is filled completely. Parameterization of L_2M_{op} in function of its parents is done by a regression analysis performed on the curves for each *datatype* separately. *Datatype* is a discrete variable that does not allow defining continuous functions over it. The curve fitting thus seeks for $L_2M_{op} = f_i(n)$ with i corresponding to the datatype. It results in a step function:

$$\begin{aligned} L_2M_{op} &= \pm 0 \quad if \quad n < threshold_{L_2M} \\ &= jump_{L_2M} \quad if \quad n > threshold_{L_2M} \end{aligned} \tag{8.5}$$

The cache misses before the jump can be neglected, since it is smaller than the resolution of the estimation. Then, the parameters $jump_{L_2M}$ and $threshold_{L_2M}$ of the functions are added to the model. They both depend on *elementsize*, as shown in Fig. 8.4(b). A regression analysis reveals that both are linearly related to *elementsize*. This can be expected. For big matrices, the cache data is badly reused since only the first element of the cache line (of 64 bytes) fetched from the lower memory level is effectively used once. The cache line is overwritten during the following instructions, when needing data further in the matrix.

Next, C_{op} is analyzed quantitatively, in order to reveal how many cycles are spent to computation and how many cycles the processor is just waiting for memory accesses. The curve C_{op} is fitted for each datatype separately and gives

$$C_{op} = CPI.\#instr_{op} + C_{L_2M}.L_2M_{op} \tag{8.6}$$

It results in an equation with 2 unknowns, CPI, the Cycles Per Instruction, and C_{L_2M}, the penalty cycles due to a cache miss. Dependency analysis reveals that CPI still depends on the *datatype*, but unexpectedly also does C_{L_2M}. The penalty cycles increase from 166 per miss for the *integer* datatype, 295 for *double*, to 418 for *longdouble*. These values, however, depend on the application. This was shown by a dependency analysis performed on data retrieved from experiments with other applications [1].

8.4.4 Reusability of Cache Misses Submodel

The previous analysis resulted in a submodel explaining the cache misses which happen during the execution of the LU decomposition. The number of cache misses was found to depend on the size of the data type whenever the matrix size exceeds a certain threshold. This is shown by the curves depicted in Fig. 8.5.

Other applications might exhibit a different behavior, but some applications will give a similar behavior. Submodels are only partially generic. The genericity depends on the given application and system. The analysis of the cache misses for a set of applications will result in a set of models, each describing a specific cache miss behavior. Each model thus explains the behavior for a certain number of applications.

8.5 Image Compression with Kakadu

The modeling approach proposed in this work is applied for modeling the parameter sensitivity of a very different application, the Kakadu implementation for the compression of still images. The analysis focuses on the sensitivity of the algorithm's performance to the parameter configuration.

8.5.1 The Algorithm

Kakadu is a C++ implementation of the JPEG-2000 standard [ISO/IEC, 2002], which has been developed to address a number of weaknesses in the existing JPEG standard and to provide a number of new features. It supports lossless and lossy compression, progressive recovery of images by resolution, region of interest coding, random access to particular regions of an image, etc. It therefore is extremely suitable for internet applications. The coding algorithm mainly consists of two stages. The first stage

[1] A matrix multiplication and a merge sort, which can be regarded as quite similar to the LU decomposition. They exhibit the same cache miss pattern as depicted in Fig. 8.5

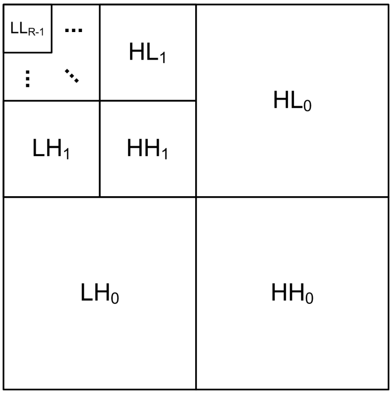


Figure 8.6: Wavelet decomposition of an image into subbands by the JPEG-2000 compression.

splits the image into frequency bands through the iterative application of a wavelet transform. Each transform yields four subbands: horizontally and vertically lowpass (LL), horizontally lowpass and vertically highpass (LH), horizontally highpass and vertically lowpass (HL) and horizontally and vertically highpass (HH). The wavelet decomposition is associated with R resolution levels, where at each level the LL band is further decomposed, as shown in Fig. 8.6. Due to the statistical properties of these subband signals, the transformed data can usually be coded more efficiently than the original untransformed data.

During the second stage, the subbands are partitioned into code blocks, typically 64 x 64, which are independently coded using a bit-plane coder (all first bits of the pixels of the block are coded together, followed by the second bits, etc.). It generates a sequence of symbols that is compressed by an entropy coder: the MQ coder. Rate scalability is achieved through quality layers. The coding passes containing the most important data (based on the lowpass subbands) are included in the lower layers, while the coding passes associated with finer details (based on the highpass subbands) are included in higher layers. During decoding, the reconstructed image quality improves incrementally with each successive layer processed. Consult [Adams, 2001] for an easy accessible introduction, and [Taubman and Marcellin, 2002] for an elaborate discussion.

The Kakadu implementation, written in C, has its key focus on memory efficiency and execution speed. I investigated the parameter sensitivity of the compression execution time - the cost in performance of changing the algorithm's configuration. Due to Kakadu's high number of parameters,

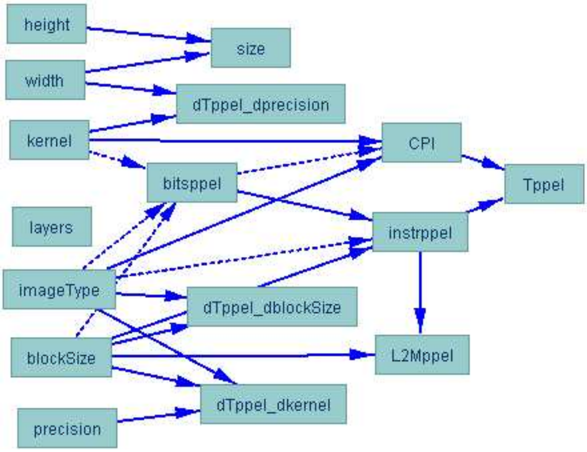


Figure 8.7: Performance Model of JPEG-2000 Image Compression with Kakadu. Arrows with dashed lines only apply for lossless compression, for which *bitsppel* is not set as a parameter.

this is a task well suited for automation. Experiments were performed with 12 different images, scaled to different widths and heights (ranging both from 500 to 10000 pixels), and various parameter settings that overspan the entire configuration space. After some try-outs I selected the most interesting parameters for running the final experiments: the *precision* of the data representation (16-bit or 32-bit), the *kernel* used for the wavelet transform (the 5/3 or 9/7 kernel), the *blockSize* used in the second stage (ranging from 4x4 up to 64x64), the number of quality *layers* (varied from 2 to 12) and the bitrate of the highest layer (set between 0.5 to 10 bits/pixel). This last parameter is only set when lossy compression is employed. At each run, the following variables are measured: the image *size*, the runtime T, the number of processor instructions *instr*, the resulted number of bits of the compressed image *bits*, the level 2 caches misses L_2M and the number of processor Cycles Per Instruction (CPI). All these quantities, except *size* and CPI, are divided by the image size to get per pixel values. They are denoted by the suffix '*ppel*'. The results show that most of these values become size independent and hence the per pixel values simplify the model. Furthermore, some interesting partial derivatives are calculated. They are named dX_dY signifying the derivative of the function of X with respect to parameter Y, while the other parameters are kept constant.

8.5.2 Performance Model

Fig. 8.7 shows the learned model for lossless and lossy compression. Lossless compression results in a bitrate, variable *bitsppel*, measured in bits per pixel, which reflects the information necessary to describe the image. It is determined by the compressibility of the image. Lossy compression allows the bitrate to be chosen by the user. The data for lossless and lossy compression were analyzed independently, after which both resulting models were merged into one. Arrows with dashed lines indicate relations that only apply for lossless compression.

The results for lossless compression comprise the following interesting observations:

- All performance characteristics increase linearly with the image size, the values per pixel are therefore size independent.
- The number of desired quality *layers* has no significant impact on the bitrate or performance.
- The resulting bitrate, *bitsppel*, influences the number of instructions, *instrppel*, and the the cycles per instruction, CPI. The bitrate is, however, an outcome of the compression and can thus not be a cause. The correct interpretation is that it is the compressibility of the image, determining *bitsppel*, that also affects the performance.
- The cache misses do not affect the runtime. This confirms the memory efficiency of the implementation. The model reveals that the number of instructions are a good indicator for the number of cache misses.
- The increase of the runtime *Tppel* by choosing a higher *precision* is not considered as significant by the independency test. The derivative *dTppel_dprecision* is, however, positive, and depends on the *kernel* and the image *width*.

The model simplifies for lossy compression, when *bitsppel* is set as a parameter. In that case, the number of instructions, *instrppel*, is only affected by *bitsppel* and *blockSize*. Furthermore, CPI is not influenced by the compressibility of the image.

8.6 Parallel Experiments With Aztec

Experiments with parallel processing were conducted with the Aztec library [Tuminaro et al., 1999](`http://www.cs.sandia.gov/CRF/aztec1.html`), which is often used as a benchmark for performance evaluation. It provides a parallel algorithm for solving of partial differential equations, defined over a 3-dimensional grid. First I study the overall performance of the algorithm and then the communication performance in more detail.

8.6.1 Experimental Setup

Aztec's algorithm supports 2 sparse matrix formats, a point-entry modified sparse row (MSR) format and a block-entry variable block row (VBR) format. Experiments are run on the dedicated cluster of the lab, which contains 8 Pentium II computers connected by a 100MHz non-blocking switch. The number of equations is varied between 100 and 400 and the number of grid points between 5^3 and 20^3. The same experiments are performed for both matrix formats.

The performance analysis tool EPPA uses the MPI profiling facilities to automatically trace the MPI calls and writes them to a database, as explained in Chapter 6. The extended PC algorithm with default options is applied onto our data. As background knowledge the user indicates which are the input (parameters) and output variables (overall performance), and by which variables the derived variables are calculated. The parameters are $nbrProcessors$, $matrixFormat$, $nbrEquations$ and $nbrGridPoints$.

8.6.2 Overall Performance

Fig.8.8 shows the model of Aztec's performance constructed by TETRAD. The variables are ordered from input to output, starting with the 4 input parameters at the left and the parallel *runtime* and *speedup* at the right. The model that was discovered is interesting, since it shows how the performance is generated. In order to analyze this in depth, 2 variables were registered which the algorithm returns at the end of the execution, namely the total number of flops needed for the computation ($totalFlops$) and the number of iterations ($totalIterations$) that the algorithm performed. The variable $totalIterations$ is completely determined by the number of grid points and does not affect the performance in a direct way. This can be understood by the fact that $totalFlops$ incorporates all information, except for $nbrProcessors$, about the *runtime*, communication time T_{comm} and idle time T_{idle}. I added an additional variable $eqXpoints$ which is

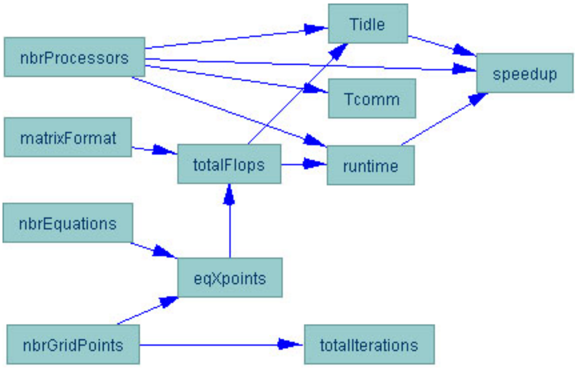


Figure 8.8: Causal model of Aztec's performance

simply the product of *nbrEquations* and *nbrGridPoints*. This is a useful variable since it explains how *totalFlops* depends on *nbrEquations* and *nbrGridPoints*. *totalFlops* is much bigger for the VBR matrix format as well as for the MSR format. Finally, the *speedup* depends on the number of processors, the computation, communication and idle time.

8.6.3 Global Communication Performance

Fig. 8.9 shows the learned model for the total communication time T_{comm} (top right). The 4 input parameters are placed at the left side. The communication performance is completely defined by the number of processors, the number of messages and the size of the communicated data. I registered 4 variables which are counted by the algorithm. The *internalUnknowns* are the elements that can be updated using only information on the current processor. *externalUnknowns* refers to the off-processor elements that are required during the calculations by the *borderUnknowns* elements. *unknownsSentToNeighbors* represents the number of elements actually sent. These definitions confirm the model. The communication is primary affected by *unknownsSentToNeighbors*, which on its turn is influenced by *borderUnknowns* and this by *externalUnknowns*. The VBR matrix format generates more messages than the MSR format. The variable *eqXpoints*, which is the product of *nbrEquations* and *nbrGridPoints*, determines *internalUnknowns* in combination with the number of processors. On the contrary, the relation of *externalUnknowns* with *nbrEquations* and *nbrGridPoints* cannot be replaced by *eqXpoints* alone.

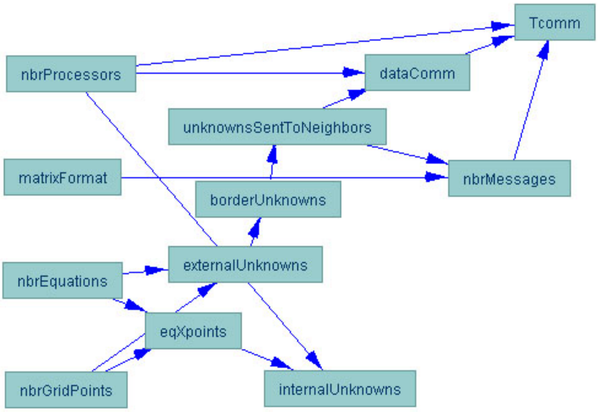


Figure 8.9: Causal model of Aztec's communication time

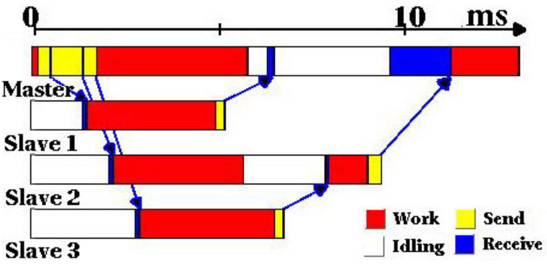


Figure 8.10: Execution profile (EPPA) of a parallel sort of 9000 integers on 4 processors.

8.6.4 Point-to-point Communication Performance

Next, I model the performance the individual point-to-point communication operations of parallel algorithms running on a cluster of computers. They are retrieved from the EPPA database as explained in section 7.1.6.

A point-to-point message involves a great number of processes. It is passed by the MPI layer to the lower TCP/IP communication layers, and via the hardware interface to the receiving process. This study, however, only intends to give the user insight in the implications for the overall performance, not into the decomposition of it. The communication system is regarded as a black box. Fig. 8.11 shows the impact of a communication operation on the execution of a parallel application. The sending and receiving of a message will consume processing time at the sender and receiver, called respectively the *senderOverhead* and *receiverOverhead*. The delays caused by the transmission of the message over the network

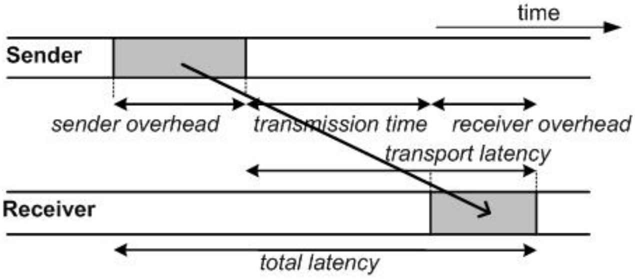


Figure 8.11: Network Performance Metrics of a Point-to-point Message

do not directly influence the execution time, but can indirectly cause processes to wait for incoming messages, which results in time-consuming idling. This is affected by the *transportLatency*, the time spent due to the transportation of the message.

Fig. 8.10 shows the typical execution profile of a parallel sort. A master process sends a part of the array to be sorted to the slaves and merges them in a final sequential step. The number of processors that participate in the experiments is varied between 2 and 8. The different phases of the execution profile, as shown in Fig. 8.10, are recorded and stored in a database by the EPPA tool. For each point-to-point communication, the relevant variables are extracted from the execution profile and written to the EPDA database. The performance is quantified by the *senderOverhead*, the *receiverOverhead* and the *transportLatency*. The input consists of the application-dependent message characteristics, such as *messageTag*, *nbrProcessors*, *phaseIndex* (the index of the phase in the execution of the process), *msgFromMaster* (does the message comes from the master process) and the application parameters such as *workSize*. Possible explanations are *messageSize*, *source*, *destination*, plus extra variables that were added later during the modeling process which is discussed in the following section.

8.6.5 Unexpected Dependencies

Fig. 8.10 shows clearly that the sender overheads are not constant for the messages the master process initially sends to the slaves, even though the sizes of the messages are identical (9000 Bytes). The sender overhead for the message to slave 2 is more than twice as much as that of the messages to slave 1 and slave 3.

A superficial analysis would suggest that indeterministic events at the master processor are responsible for these non-linear effects. This illustrates the necessity for a wide diversity of experiments. Attributing the master process to different nodes makes it possible to draw conclusions about processor-specific performance characteristics. Experimental data from different system configurations confirm that all nodes of the cluster are identical. Additional variables have to be added for explaining the variations in the sender overheads. By manual inspection of the execution profiles and data, the following, among many others, characteristics are identified: [2]

- *otherCommSenderBefore*: the number of other messages also sent by the sender but not yet arrived when the communication started.
- *nbrMessageArrivals*: the number of messages also sent by the sender that arrived during the send overhead.
- *receiverIdleTime*: the time the receiving process was idling longer before it started receiving the message.

Causal analysis confirms their usefulness, the model depicted in Fig. 8.12 shows that they render the explanations for the *senderOverhead* application-independent. Variables *otherCommSenderBefore* and *nbrMessageArrivals* reflect the background sending of the messages preceding the communication operation. *NbrMessageArrivals* is the main indication for the background communication, it is generated by *otherCommSenderBefore*. However, it cannot explain the entire *sendOverhead* increase, *otherCommSenderBefore* is therefore also connected with the *sendOverhead*. However, both variables cannot be regarded as true causes of the *sendOverhead*, they actually represent observable manifestations of the hidden processes. The causal sufficiency assumption is violated here, since the background communication, which is the common cause of all three variables, is not characterized. The left side of the model shows, in the context of the sort application, the application characteristics that affect the background communication. Finally, the transport latency is proportional to the message size, but is also affected by the other communications at the sender and the receiver's idle time.

8.6.6 Explanations for Outliers

In the set of data from the experiments on the laboratory non-dedicated network, some exceptional data (outliers) were detected. There are a few long message delays recorded for messages with size of about 26000 bytes,

[2] I only report about the variables that appeared to be relevant during the analysis.

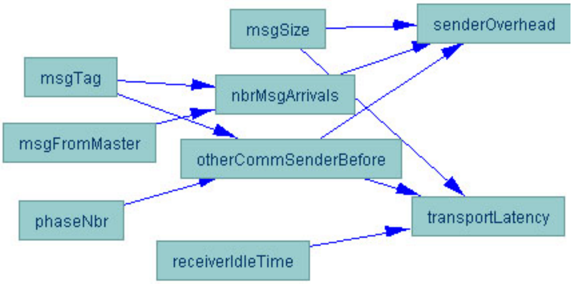


Figure 8.12: Causal model of the communication performance of a parallel sort application for messages smaller than 64KB

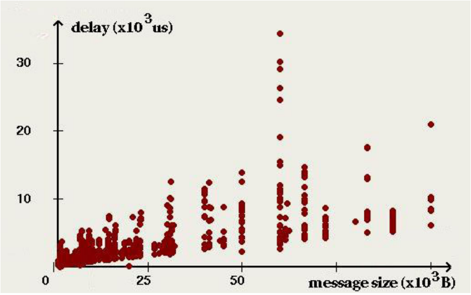


Figure 8.13: Exceptions in measured delays for experiments on the laboratory cluster

as can be seen in Fig. 8.13. Causal inference is applied to identify the cause(s) of the exception. The experiments are tagged with a boolean indicating that the data points are exceptional and loaded into TETRAD. TETRAD reveals that the exceptional data was caused by one parallel experiment and one processor which was the destination of the messages, as shown in Fig. 8.14. Experiment 46728 and node 7 were extracted from the conditional probability table, as explained in Section 7.1.4. By these results, we may hypothesize that during experiment 46728, computer 7 delayed the handling of the incoming messages, probably due to another process running simultaneously. The model should, however, be interpreted with care. The experiment and the destination cannot be considered as the 'true causes' of the exceptional data, more as indications of the time and place where the event that caused the extra delays happened.

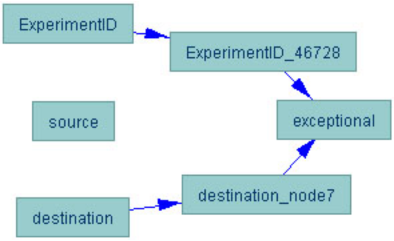


Figure 8.14: Causal model explaining exceptional data

8.7 Summary and Conclusions of Chapter

This chapter demonstrated the utility of integrating causal inference in the performance modeling process. It offers several benefits to support the performance modeling process as well as the development of high-performant software. Causal models of a performance analysis are called causal performance models. The relations among the variable can be attributed a causal interpretation. Analysis of experiments with sequential and parallel applications demonstrated that correct and useful models are learned about their performance. Moreover, unexpected dependencies were discovered, assumptions about independencies could be validated and potential explanations for outliers were found. Concluding, for the purpose of performance analysis, causal inference must be regarded as an interesting technique, which can be utilized to perform some of the subtasks encountered during performance modeling.

The following table summarizes the potential benefits that were proposed in the first section of this chapter. The second column indicates whether the benefit is supported by the experimental results (+: benefit is proved; ±: benefit is still disputable ; ?: results do not give a definite conclusion yet; FW: future work):

Possible Benefits	OK?	Comment
Model construction	+	
Validation	+	
Reuse of submodels	?	Possible in some cases
Reveal characteristics	±	Is analyzed in more depth in the next chapter.
Reveal explanations	+	
Flexibility	FW	Theoretically possible, but learning algorithms should be adapted for this.
Report	?	The benefits of a graphical representation is yet to be shown.
Filtering relevant information	FW	
Reasoning under uncertainty	FW	Must be based on a quantitative analysis.
Qualitative reasoning	FW	

Chapter 9

The Genericity of Performance Models

HOW complex things can get? How difficult is it to predict the performance of the execution of applications on systems? Obviously, computing-intensive parallel applications running on high-performant clusters lead to an overall performance difficult to master. But even simple algorithms cannot be reduced to a linear combination of factors influencing the performance, such as instructions (eventually grouped by type), cycles per instruction, number of cache misses and memory access penalty cycles. The ingenuity of present computing units make that calculations and data access operations are executed in a partially parallel and pipelined way. This results in a complex game of intertwining micro-processes. Performance thus greatly depends on the match of the application's instructions sequence and the specific way the system handles the instructions.

Another example in which the performance depends on a match is that of a delivery of a set of messages through an interconnection network. I will show that generic models are difficult to construct for this problem. They are limited by the patterns of the communication and of the network topology. Random communication on a random network behaves statistically and is predictable. While specific combinations of regularities may result in a specific behavior. The regularity of the communication might, or might not, match with the regularity of the topology, resulting in a respectively better or worse than expected communication time. Only a particular model adequately captures the performance behavior.

The first section begins with the discussion of the convolution method to generic characterization of systems and applications. Some modeling examples show the difficulty of building generic models. The rest of this

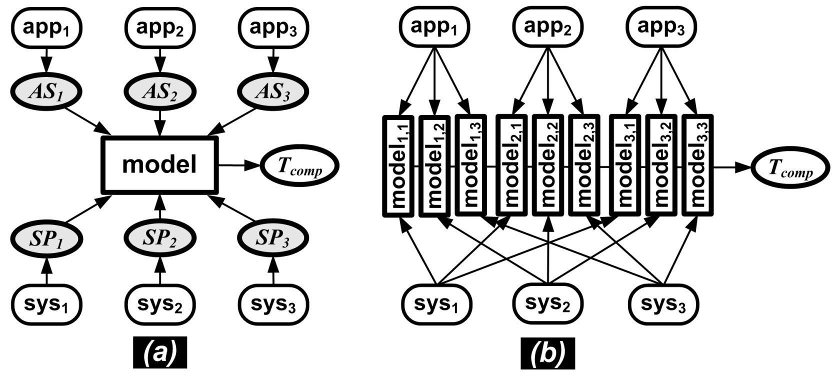


Figure 9.1: Applications and computer systems can be characterized by generic performance properties, respectively the Application Signature (AS) and the System Profile (SP) (a). Every combination of application and system entails a specific model (b).

chapter is devoted to the modeling of the performance of structured communication in a structured network. Firstly, the problem setting is discussed. Next, the conclusions for qualitative and quantitative performance models are drawn.

9.1 Runtime Performance Models

This section discusses the question whether generic models exits for the prediction of the runtime of the execution of an application on a system.

9.1.1 The Convolution Method

The goal of the convolution method is the prediction of the runtime of an application on an arbitrary system Snavely et al. [2001]. This requires the existence of independent application and system performance characteristics - called *application signature* and *system profile* - and a functional relation to calculate from both the performance of an application running on a system. The separation of signatures and profiles means that signatures need only be gathered once to support prediction on multiple machines. Fig. 9.1(a) depicts the method. The performance sensitivity of each application app_i is completely characterized by Application Signature AS_i and each system sys_j by System Profile SP_j. The genericity of these characteristics ensures the existence of a function which accurately predicts the performance of any combination of application and system:

$$T_{comp}(app_i, sys_j) = f(AS_i, SP_j) \tag{9.1}$$

This results in the following independencies:

$$T_{comp} \perp\!\!\!\perp app_i \mid AS_i, \tag{9.2}$$

$$T_{comp} \perp\!\!\!\perp sys_j \mid SP_j. \tag{9.3}$$

The genericity of the characteristics signifies that application characteristics are independent of the system, and thus valid for all systems. Vice versa, system characteristics are application independent, adequate for predicting the performance of other applications. These requirements are expressed by the following independencies:

$$AS_i \perp\!\!\!\perp sys_j, \tag{9.4}$$

$$SP_j \perp\!\!\!\perp app_i. \tag{9.5}$$

The absence of such generic properties would imply that specific combinations result in specific models. The worst case is depicted by Fig. 9.1(b). Each combination of application and system behaves in a unique way so that it requires a specific model:

$$T_{comp}(app_i, sys_j) = f_{i,j}(app_i, sys_j) \tag{9.6}$$

The question is at which point we are between both extremes. Does one model and generic characteristics suffice for accurate performance estimation and understanding; or does every combination require a different model?

Related Work

A similar strategy is employed by Marin and Mellor-Crummey Marin and Mellor-Crummey [2004]. They also try to separate the contribution of application-specific factors from the contribution of architectural characteristics to overall application performance. Results indicate, however, that the use of a single, simple synthetic metric, or a linear combination of such simple metrics, to predict the performance of high-performance application performs poorly Carrington et al. [2005].

The approach here advocated falls under the category of tools that monitor performance by collecting statistical data of multiple experiments. The other approach is to study event traces, according to which the exact sequence of actions that took place during a run is recorded. Statistical data is more compact than event traces, but it is already noticed that the predictive power is limited Carrington et al. [2005]. Better predictions can be acquired by simulating the recorded trace on a model. This approach is limited since it doesn't give parameterized models. It also provides little insight into the factors that determine the overall performance.

More research about analytically based performance prediction was conducted by Kerbyson et al. [2002]. Lastovetsky and Reddy [2004] constructed detailed models, including the performance of memory accesses, for predicting computation time.

9.1.2 Generic Characteristics for the LU Decomposition Algorithm

The modeling results presented in this and the next subsection illustrate the difficulty of creating generic models.

Recall the LU decomposition algorithm studied in Section 8.4. With the variables considered in the analysis, one can try to construct a model in the sense of the convolution method. Variables have to be identified which characterize the application, independent from the system, on the one hand and application-independent variables characterizing the system on the other hand. The first constitutes the application signature, the second the system profile.

Fig. 9.2 depicts the proposal for a generic model. Variables C_{instr} and C_{mem} are added to differentiate between the cycles spent on executing instructions and those spent idling due to memory accesses. The model supposes that $threshold_{L_2M}$ can be calculated by the *memory usage* of the application and the *memory size* of the machine. The model shows which variables *could* constitute the application signature and which the system profile, respectively the top and bottom variables. This characterization is, however, too simple. C_{L_2M} (the penalty cycles due to a level 2 cache miss) is not a system constant. The results show that it depends on the datatype and application. Also CPI, the cycles per instructions, is application dependent. Moreover, $threshold_{L_2M}$ cannot be determined by the memory requirements and memory size only. The modeling thus failed in finding independent system characteristics.

9.1.3 Image Characterization for the Kakadu Compression Algorithm

Reconsider the Kakadu image compression algorithm studied in Section 8.5. Fig. 9.3 recalls the model for the performance of lossless and lossy

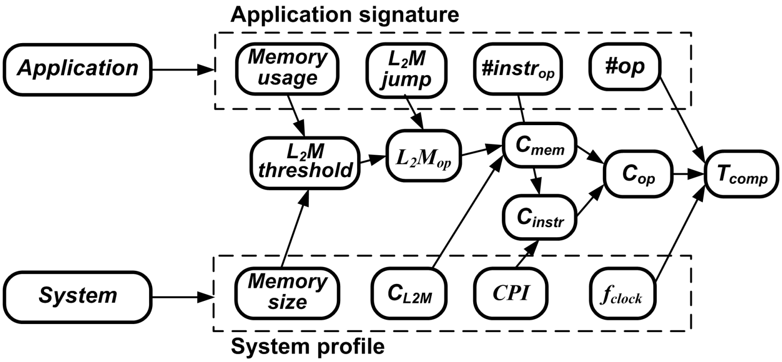


Figure 9.2: Fictive performance model of LU decomposition with independent application and system characteristics.

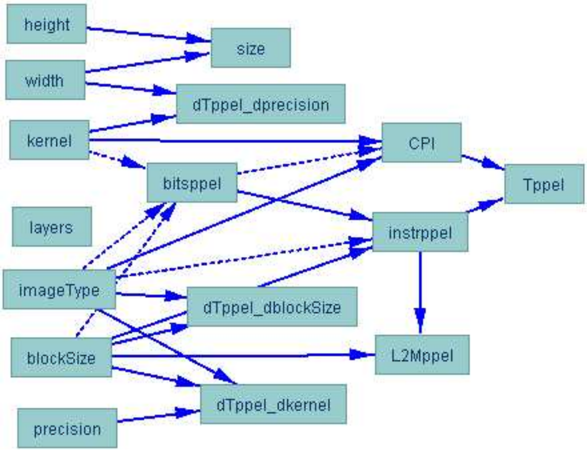


Figure 9.3: Performance Model of JPEG-2000 Image Compression with Kakadu. Arrows with dashed lines only apply for lossless compression, for which *bitsppel* is not set as a parameter.

compression that was learned in the previous chapter. Ideally, every image can be characterized by one single compression performance factor, which determines the number of instructions as well as the runtime, and is valid for all possible parameter configurations. This is certainly not the case as can be seen from the model. For lossless compression, *imageType* influences the compressibility *bitsppel*, *instrppel* and *CPI* independently. This means that an image has at least three distinct features characterizing the compression performance. None of the three variables contains the information of *imageType* so that it is able to predict the values of the 2 others without needing the value of *imageType*. Next, the influence of *imageType* on *dTppel_dkernel* indicates that the compression behavior of an image with one kernel says little about its behavior with another kernel. Indeed, inspection of the data reveals that some images are compressed faster with kernel 5/3 and others with 9/7. Images also behave differently under different blocksizes, expressed by the relation between *imageType* and *dTppel_dblockSize*. Only the computational cost of using higher precision data representations, *dTppel_dprecision*, is image independent.

9.2 Execution of Communication Schemes on Network Topologies

The performance of parallel programs greatly depends on the time the system needs to exchange data among processors. The communication overhead depends on the amount and size of the messages and the transmission capacities, in terms of latency and bandwidth, of the underlying interconnection network. But this overhead also depends on the distribution of the messages and on the **network topology** (NT). The ensemble of messages sent among the processors is called the **communication scheme** (CS) of the program. My research focuses on the effect of *regularities* in network topology and/or communication scheme.

9.2.1 Problem Description

Consider a star NT, where every processor is connected to one central processor, or a ring NT, in which the connections of the processors form a ring (Fig. 9.4). Consider a structured CS such as a one-to-all broadcast or a shift collective communication. In the former one process sends a message to all other processes, where in the latter every process sends a message to its direct neighbor (Fig. 9.4). It is intuitively clear that a one-

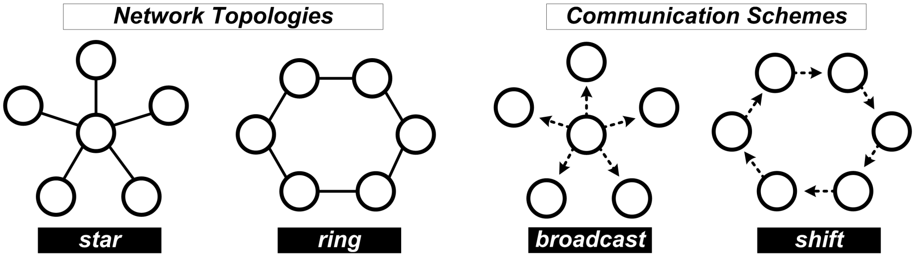


Figure 9.4: Two Regular Network Topologies and Two Regular Communication Schemes.

to-all broadcast communication on a star network and the shift operation on a ring network are very efficient. In contrast, a broadcast on a ring or a shift on a star results in a much higher runtime. The efficiency by which communication can be completed depends on the *match* of CS and NT. The patterns in CS and NT affect which and how variables affect the communication time.

Two different models have to be considered: one for *understanding* the causes of the performance and one for *prediction* of the performance.

The question I will try to answer is under which circumstances and to what extent generic models can be constructed. Generic models must be valid for any combination of CS and NT and rely on generic performance properties of communication schemes and network topologies. This problem is equivalent to that posed during the discussion of the convolution method and depicted by Fig. 9.1. A generic model based on generic characteristics of CS and NT has the same form of model (a). The absence of a generic model could lead to the situation in which each combination of CS and NT requires a different model, as is shown by situation (b).

I will show experimentally that random NTs and random CSs can be characterized by a single performance value. The characteristic allows accurate prediction of any combination of CS with NT. The model based on this characteristic can, however, become very inaccurate when patterns occur in CS or NT. Non-random NTs or CSs, exhibiting certain patterns, can differ not only quantitatively, but also qualitatively by the properties of NT and CS influencing the performance. The results reveal that it is not possible to characterize the performance of an NT while not considering the patterns of the CS. An NT can perform well for one CS, but not for another.

In the study of the performance of the internet it is also discovered that the development of accurate models poses many problems. The interconnection topology is not random or a simple planar graph, but a constantly changing heterogeneous combination of regular structures [Floyd and Paxson, 2001][Zegura et al., 1997][Calvert et al., 1997]. On the other hand, internet traffic at the level of IP packages resembles more self-similar than Poisson processes [Floyd and Paxson, 2001]. Our results show that regularities in network or traffic can give different quantitative and qualitative results. Modeling these regularities for the internet is therefore required to obtain accurate simulations.

9.2.2 Experimental Setup

The analysis is based on experimental data retrieved from simulation of the communication on the network. Random schemes and random topologies will be considered, but also schemes and topologies having a certain structure. Our research is limited to homogeneous network topologies with bidirectional links all having identical latency and bandwidth. The time a message takes to make one hop is approximated by $latency + messagesize/bandwidth$. The simulation model of the network is described by a graph in which each node represents a processor and each edge a communication link. A communication scheme is a set of messages with a certain source, destination and message size. The message size is chosen to be the same for all messages, since its influence on the execution is not a part of my investigation. The message hop time is thus a constant. It is set to 1 time unit. Messages take the shortest path to go from source to destination. Messages are queued if the link through which they have to pass is occupied. The simulator is a discrete event simulator [Banks et al., 2005], the operation of a system is represented as a chronological sequence of events. Each event occurs at an instant in time and marks a change of state. In simulating network traffic, each hop a message makes is an event.

The following graph types are considered for the network topologies, each of them having specific parameters:

- **Random graph**: is constructed by probabilistically adding edges to a given set of nodes. Parameters are the number of nodes, $nodes$, and the relative node connection degree, $relConnect$, defined by the average number of links of a node divided by the total number of nodes. It is ensured that there exists a path between any two nodes.

- **Torus**: is a closed surface defined as the product of two circles. Parameters are the number of nodes of the first circle, $nodesC_1$, and that of the second circle, $nodesC_2$.

- **Ring**: is a graph in which all nodes are connected so that they form a ring. Its parameter is the number of nodes, $nodes$.

- **Star**: is a graph that consists of one node in the middle which is connected to all other nodes. Its parameter is the number of nodes, $nodes$.

- **Neighbor graph**: is a graph in which nodes are randomly positioned in a plane and randomly connected but with a higher probability for neighbor nodes. The probability of having an edge between two nodes decreases quadratically with the distance between them. The parameter is the relative node connection degree, $relConnect$. It is the same to that of a random graph.

- **Planar graph**: is a graph in which nodes are randomly positioned in a plane and randomly connected, but in such way that none of the edges intersect. The sole parameter is the relative node connection degree, $relConnect$.

Each type is represented by a set of instantiations, generated by randomly chosen parameter values. The values are picked out of the following ranges according to a uniform distribution: $nodes \in [20, 100]$, $relConnect \in [0.05, 0.9]$ and, for the torus, $nodesC_1 \in [4, 14], nodesC_2 \in [4, 14]$.

Considered communication schemes and their parameters are:

- **Random communication**: each node sends a number of messages, randomly chosen with average $nodeMsgs$, to randomly selected destinations. The range of the parameter is: $nodeMsgs \in [20, 100]$.

- **One-to-all broadcast**: one node, the node with index 0, sends a personalized message to each of the other nodes.

- **All-to-all broadcast**: each node sends a personalized message to each of the other nodes.

- **Shift**: each node i sends a message to the node with index $i + indexShift$. Parameter $indexShift$ is chosen from the range $[1, 5]$.

Variables that are measured during each experiment are the following:

- **Graph properties**: $avgDistance$, average distance between two nodes (distance is the minimal number of hops required to get from one node to another) and $diameter$, which is the maximal distance between any two nodes.

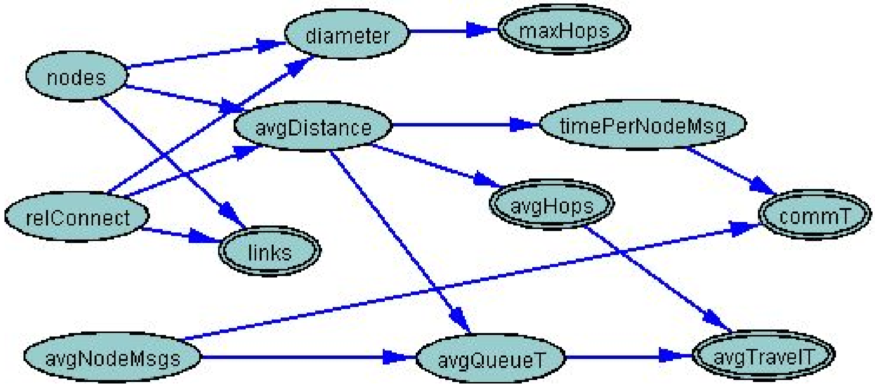


Figure 9.5: Model of communication performance of random communication on a random topology.

- **Communication properties**: average number of messages per node, $avgNodeMsgs$.

- **Overall performance**: total time to complete all communication, $commT$, the runtime divided by the average number of messages per node, $timePerNodeMsgs$, average time it takes for a message to arrive at its destination, $avgTravelT$, average number of hops of a message, $avgHops$, maximal number of hops of a message, $maxHops$, and average time a message is queuing, $avgQueueT$.

9.3 Qualitative Performance Models

The purpose is a qualitative model, which uncovers the causal dependence of communication time on properties of CS and NT. The extended PC learning algorithm discussed in Chapter 7 is employed to infer causal models from experimental data.

First a model is inferred from the data of a set of 200 couples of random CSs and random NTs. Next, the models for all combinations of CS types and NT types are explored.

9.3.1 Random Communication Schemes on Random Network Topologies

Analysis of experiments with random communication on random interconnection networks results in the model shown in Fig. 9.5. The communication time depends linearly on the number of messages per node,

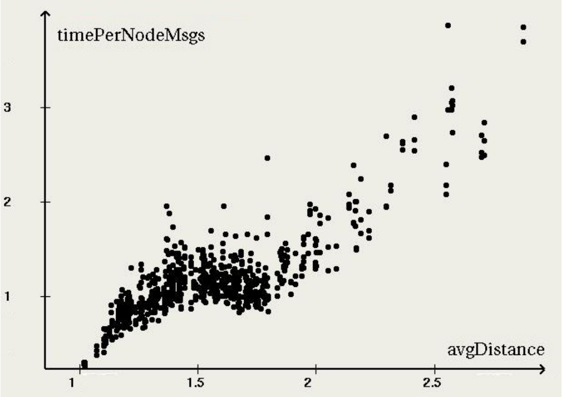


Figure 9.6: Communication time per number of messages per node versus average distance for random NT and random CS.

avgNodeMsgs. Hence it made sense to introduce the ratio *timePerNode−Msgs*. It gives a quantity that is independent of the number of messages and therefore reduces the complexity of the model. Deterministic variables, which are a function of its parents in the graph, are depicted with double-bordered circles. The average number of hops, *avgHops*, is determined by the average distance, *avgDistance*, of the graph. The model also reveals that the variable *avgDistance* gives maximal information about the time per node message. *avgDistance* is, however, almost completely determined by *relConnect*, except for low values of *relConnect*. With a low number of links, the graph's randomness makes that the average distance can vary a lot from graph to graph. Variable *relConnect* thus also greatly influences *timePerNodeMsgs*, but does not capture all information about it. Variable *avgDistance* does.

Fig. 9.6 shows the statistic *timePerNodeMsgs* as a function of *avgDis−tance*. For any given parameter configuration, *nodes*, *relConnect* and *avgNodeMsgs*, there is still a high uncertainty on the communication time. In the next section, a quantitative model will be proposed to characterize individual NTs and CSs so that the uncertainty can be reduced.

9.3.2 Regular Communication Schemes on Regular Network Topologies

The models that are learned for the combinations of CS and NT classes give models that are sometimes quite different than that of complete randomness (Fig. 9.5). The following table summarizes these qualitative differences. '=' denotes that the model corresponds with the model for random communication on random networks, '≠' that they are different. When different, the numbers in brackets explain the differences.

	random NT	*torus*	*neighbor*	*planar*	*star*	*ring*
random CS	reference model	≠ (6)	=	= (1)	=	=
broadcast	≠ (1)(3)(4)	≠ (4)	≠ (5)	= (1)	≠ (8)	= (2)
all2all	=	≠ (6)	=	=	≠ (9)	=
shift	= (1)(3)	≠ (7)	= (5)	=	≠ (8)	≠ (7)

Explanation of the differences:

- (1) The relations have the same shape, but there is a higher uncertainty on the relations.
- (2) All relations are deterministic.
- (3) The relation *avgDistance* - *avgHops* is not deterministic.
- (4) The number of nodes, $nodes$, also influences $timePerNodesMsgs$.
- (5) Is identical to model of random communication with the same CS.
- (6) Besides *avgDistance* influencing the communication time, it is also influenced by $nodesC_1$ and $nodesC_2$. The message delivery is less optimal the more $nodesC_1$ and $nodesC_2$ differ. If $nodesC_1$ is low, there exist only a few paths connecting the circles of the second dimension. This leads to congestion in these paths. A variable corresponding to the absolute difference between both, $|nodesC_1 - nodesC_2|$, was added to verify this.
- (7) The index shift, $indexShift$, also determines performance. Moreover, for a ring network it is the only parameter affecting the performance variables $timePerNodesMsgs$, $avgHops$ and $maxHops$.
- (8) All performance variables have a constant value.
- (9) $timePerNodesMsgs$ is a constant, $nodes$ affects $avgQueueT$.

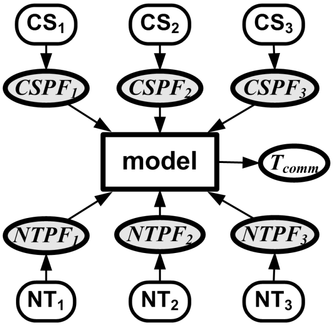


Figure 9.7: Generic quantitative model to predict the execution time of a communication scheme (CS) on a network topology (NT).

These results show clearly that the model of Fig. 9.5 becomes invalid. But it is difficult to draw general conclusions. Random communication behaves quite similar for all kinds of graphs, except for a torus, in which case the average distance is not the sole direct cause of $timePerNodesMsgs$. The neighbor graph gives the same models as for the random graph. But the table shows that specific regularities in the communication can result in very specific matches, such as on star or ring graphs. The correctness of the learned models can be verified when one reasons about how the execution of a specific regular communication on a structured topology behaves. For instance, consider a shift collective communication performed on a ring which results in a very efficient execution. The transmission happens synchronously. Each transmission channel is occupied by exactly one message at each time instance. No message has to queue before arriving at its destination. $indexShift$ determines the number of hops each message has to make and thus also the communication time.

9.4 Quantitative Performance Models

The previous section was devoted to qualitative models, providing insight into the performance. In this section I am interested in the prediction of the communication time. I will build a generic model, in the sense of Fig. 9.7 and the convolution method, which adequately estimates the runtime of random communication on random topologies. A generic performance property is defined that characterizes individual NTs and CSs, and a simple function that calculates the runtime of any combination of NT and CS. In the subsequent subsection I investigate whether this quantitative model is also valid for combinations of regular CSs and NTs.

9.4.1 Random Communication Schemes on Random Network Topologies

I propose a definition of a performance factor for characterizing individual NTs and CSs, called respectively $NTPF$ and $CSPF$. Its value will be experimentally measured on a benchmark. The benchmark consists of 50 random graphs having 25 nodes and varying *relConnect* and 50 random communication schemes having 25 messages per node. The benchmark average runtime, denoted $refT$, is the average runtime of all 50×50 combinations of the benchmarks NT and CS. It serves as a reference for characterizing the performance of NTs and CSs. The factor of topology i, $NTPF_i$, is defined as

$$NTPF_i = \mathit{benchmarkT}(NT_i)/refT \tag{9.7}$$

with $\mathit{benchmarkT}(NT_i)$ the average runtime of running graph i on the 50 benchmark communication schemes. This factor lies around 1. If it is smaller, it means that the communication finishes faster than on the benchmark graphs. If higher, the communication needs more time to complete. Likewise is the factor of CS j defined by the average runtime of the simulation of the CS on the benchmark topologies, denoted $\mathit{benchmarkT}(CS_j)$:

$$CSPF_j = \mathit{benchmarkT}(CS_j)/refT \tag{9.8}$$

The function to predict the runtime of running CS j on NT i is defined as follows:

$$\mathit{predictionT}(CS_j, NT_i) = refT \times CSPF_j \times NTPF_i \tag{9.9}$$

Fig. 9.8 present the results for the comparison of the estimated with the real runtime for a test set of 100 random NTs and 100 random CSs. Although there is a difference, a correlation coefficient of 0.98 indicates that a good estimation is achieved.

9.4.2 Regular Communication Schemes on Regular Network Topologies

Random communication on random topologies results in statistical values. Deviations of the average for specific instantiations are accurately modeled by the CS and NT performance factors. This section will uncover that the method based on the benchmark NTs and CSs is not always successful in predicting the performance of regular NTs and CSs. NTs containing 25

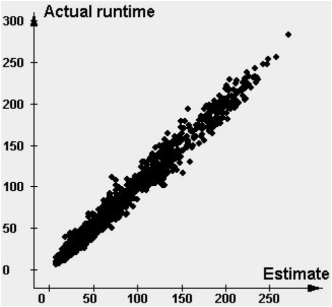


Figure 9.8: Real versus estimated communication time for random communication on random topologies.

nodes are considered. The same equations, Eq. 9.7 and Eq. 9.8, were used for experimentally attributing a value to each CS and NT type. A set of instantiations for each type (see section 9.2.2) was benchmarked. The results are shown in the following table. The numbers come from averaging over the results of 50 different sets measured on 10 different benchmarks. It gives the average benchmark runtime, the average factor and the average standard deviation of the factor for the different instantiations in each set.

The latter is an indication of how much the performance factor varies from one instantiation to another. Its value is not filled in if there is only one instantiation. The average runtime of the benchmark, $refT$, is 46.9 time units. Recall that 1 time unit is needed for a message to perform 1 hop.

	benchmarkT	*performance factor*	*stddev*
random graph	50	1.0	0.9
torus	123	2.4	1.3
neighbor graph	141	2.8	2.1
planar graph	191	3.8	1.7
star	77	1.5	-
ring	190	3.7	-
random	50	0.99	0.09
broadcast	8.7	0.17	-
all2all	44	0.87	-
shift	25	0.49	0.27

Most graphs have a factor that is more than 1, thus performing worse than a random graph. One must note that the number of links is not taken into account. A low factor is therefore not necessarily an indication of a bad efficiency when the number of edges is taken as the cost of the network.

Once the factors established, the quality of the estimated runtime calculated with Eq. 9.9 can be verified for each combination of regularity type. I calculate the relative estimation error according to the ratio of the average sum of squared errors (SSQ) with the average runtime:

$$relative\ estimation\ error(NT_i, CS_j) = \frac{\sqrt{SSQ/n}}{averageT(NT_i, CS_j)} \qquad (9.10)$$

with n the number of data points. The results are shown in the following table with the CSs defining the rows and the NTs the columns. The value in brackets gives the relative difference between the average estimated time and average real runtime for the set.

	random graph	*torus*	*neighbor graph*
random	14% (+0.8%)	7% (-0.3%)	9% (-0.5%)
broadcast	**72% (+11%)**	**40% (-25%)**	**94% (-50%)**
all2all	12% (-0.2%)	6% (-6%)	10% (+6%)
shift	**61% (+5%)**	**270% (-160%)**	**97% (-46%)**
	planar graph	*star*	*ring*
random	8% (+0.2%)	8% (+1%)	6% (-0.4%)
broadcast	**95% (-70%)**	**1150% (-1150%)**	**39% (39%)**
all2all	9% (+6%)	**37% (-37%)**	5% (-5%)
shift	**90% (-65%)**	16% (-12%)	**330% (-250%)**

The standard deviation of the values over the 50 different experiments is about 7% of the given values. This shows that there is a quite large 95% confidence interval for the values of about 15%. Nevertheless is there a big difference between combinations that give quite good estimations and those for which the prediction is completely wrong (put in boldface).

The standard deviation of the estimation of random CS on random NT is 14%. But the low value of 0.8% for the difference between average estimation and real time indicates that the estimation is unbiased. On the contrary, a broadcast on a planar graph executes consequently faster than expected. An average difference of -70% for a relative error of 95% shows that almost all estimations are too high. Several combinations lead to better matches than expected by the benchmark. While other combinations are well predicted, such as the all2all communication scheme. An all2all resembles random communication.

9.5 Summary and Conclusions of Chapter

This chapter brought the attention to the problem of model genericity and the existence of generic performance characteristics. The problem was outlined for models trying to explain or predict the runtime of an application. The results given in literature show that even for simple applications the existence of generic models can not be guaranteed. Finding generic application and system characteristics is difficult. Even more, the mere existence of generic characteristics can be questioned. The rest of the chapter was devoted to the study of the execution time of a Communication Scheme (CS) through a Network Topology (NT). It is demonstrated that the execution time can be adequately modeled when the NT corresponds to a random graph and the CS is correctly modeled by messages with randomly chosen source and destination. A generic property and a simple function were defined that are able to predict the performance for any combination of random CS and random NT. The property is measured on a benchmark which consists of a set of random CSs and random NTs. However, when NT or CS exhibit specific regularities, the qualitative and quantitative models may become invalid. Regular graphs give good predictions when confronted with random communication, but not when combined with certain structured communication types. No general conclusions can be drawn. The qualitative and quantitative models can be radically different for specific combinations of CS regularities and NT regularities.

Concluding, the regularities in NTs and CSs cannot be ignored for effective understanding and prediction of communication performance. Simple models can only be realized when limited to a specific combination of regularity types.

Chapter 10

Conclusions

SCIENCE is law-making. Computer science is about automation. My work has dealt with the automation of law-making. The algorithms for causal inference form one of the machine learning approaches to inductive inference. They aim not only at learning useful models, but also intend to uncover the underlying mechanisms from a system under study.

The growing complexity of computer systems and applications makes that the performance of the match of both has become increasingly challenging to master. Approaches for automated analysis are more than welcome. I studied the application of causal inference to a performance analysis, in such a way that valuable models can be learned automatically from experimental data.

Scientific Results

The value of causal inference is given extra impetus by the following contributions:

- The interpretation of causal model theory is related to that of the Kolmogorov Minimal Sufficient Statistic (KMSS). The validity of causal inference is related to the presence of regularities not represented by causal models.

- The faithfulness property is interpreted in a broader sense as the capability of a model to explain all regularities of the observations.

- The hypothesis is put forward that the implications of the causal interpretation of the models correspond to those of a model decomposition.

- The theory is extended to incorporate information equivalences. The complexity of relations is used to determine which one, among information equivalent relations, is the direct cause.
- A form-free independence test is implemented, which is based on a kernel density estimation and calculated with the definition of mutual information. Moreover, it allows for the analysis of data containing a mixture of continuous, discrete and categorical variables.
- The utility of causal inference for the performance analysis of applications and computer systems was demonstrated.

The contributions to a performance analysis are multifold:

- Two tools (EPPA and EPDA) were developed to record the results of experiments with sequential and parallel applications and interprete the experimental data according to a multivariate statistical analysis.
- The developed and studied algorithm for causal inference provides a useful technique to reveal the relations among the variables of interest. It can be integrated into current tools and allow the further automation of the performance modeling process. It must not be regarded as a solution for performance analysis, but as a part of the solution. Whenever questions arise about the relations among variables, the algorithm can provide this information.
- Regularities in communication scheme or network topology were shown to considerably influence the match of both. This dependence on regularities limits the existence of generic models. Certain combinations of regularities need specific qualitative and quantitative models.

Qualitativeness

The different lines of my research all point to the concept of *qualitative properties*. However, a formal definition or treatment of qualitativeness is not given. The scientific results only offer a glimpse of how a theory about qualitative information would look like.

The main motivation of Pearl to develop the theory of graphical causal models was the observation that conditional independencies are qualitative properties [Pearl, 1988, p.79] (discussed in Section 3.1.2). Conditional independencies play a different role in knowledge as opposed to the quantitative information of probabilities. This observation triggered Pearl's research on Bayesian networks and causal models. Bayesian networks were proposed as models which explicitly describe these qualitative properties. The 'faithfulness' property was defined to express that a model describes all conditional independencies of the system under study.

My work showed that a qualitative analysis can contribute to the understanding of the performance of programs. Many questions about the performance of applications are of qualitative nature.
Which parameters influence the performance considerably?
Which variables are sufficient to predict overall performance?
Which program sections allow for optimization?
Does the computation time increases in a super-linear way with increasing work size?
In contrast, questions about exact values of the performance metrics are quantitative questions.

I demonstrated that regularities determine the delivery of messages in an interconnection network. Regularities in communication scheme and network topology cannot be neglected. They determine whether the distribution of the messages occurs in a synchronized or chaotic fashion. Different combinations of regularities result in different behavior. These differences cannot be explained by a single model without taking the regularities into account. Regularities result in qualitative differences.

These considerations give a proposal for a definition of qualitative properties: the regularities of an object. In the sense that regularities allow the effective compression of the description of the object. The KMSS provides a formal instrument to separate information into a meaningful and meaningless part, where the meaningful part is based on the regularities of the data. The results are, however, too preliminary to draw a definite conclusion about the formalization of qualitative properties or regularities.

The importance of regularities to modeling was underlined by their important role in the discussion of the validity of causal inference. If all conditional independencies follow from the causal structure, causal inference works and the causal interpretation offers a likely hypothesis. This is expressed by the faithfulness property. In the perspective of inductive inference, this principle can be interpreted as that 'a model should be able to explain all qualitative properties of the observations'. The correspondence of a model with reality seems to heavily rely on this condition.

Limitations and Directions for Future Research

Finally, I comment on the limitations of this work and give some possible directions for future research.

Chapter 2: *Principles of Inductive Inference.*

Currently, the application of Kolmogorov Complexity to inductive inference is limited to the selection of the minimum model from an a priori chosen model class. This contrasts with the scientific activity of actively searching for regularities. However, a formal definition of regularities is lacking, as well as an algorithmic approach to the detection of regularities. Formally, the latter is related to the intractability of Kolmogorov complexity; the absence for a proof a minimality. It seems that we are far from replacing the human creative mind with a computer.

Chapter 3: *Graphical Causal Models.*

The existence of a faithful graph is a necessary assumption for the correct course of the constraint-based learning algorithms; at least a relaxed version of faithfulness, as defined by Ramsey et al. [2006]. Anyway, the major weakness of the constraint-based learning algorithms is that they are not *robust*. A wrongly classified relation as dependent or independent has a great effect on the learned model. Take model $A \rightarrow B \leftarrow C$ in which the association between B and C only becomes apparent when conditioned on A. This may happen when the influence of A on B is much greater than that of C on B. Especially in the case of non-linear relations the association of B and C is difficult to measure. The independence $B \perp\!\!\!\perp C$ leads to the wrongly deletion of the edge $B - C$.

Chapter 4: *The Meaningful Information of Distributions.*

The counterexamples of causal inference indicate that in some cases the faithfulness assumption is unreliable. Not all independencies come from the system's causal structure. Therefore, according to my viewpoint, causal models should be extended to capture other regularities. Background knowledge seems indispensable in knowing which classes of regularities should be taken into consideration.

To further investigate the importance of causal model theory in understanding causality, the relation with mechanisms must be laid down more precisely. The correspondence of the causal component with a decomposition must be researched further.

It must be noted that the limitations of causal structure learning from observation can be overcome by doing experiments based on interventions [Korb and Nyberg, 2006]. Experimental manipulation of variables gives extra information about the causal relations. This approach seems indispensable for correct causal inference. It can be compared with the common scientific practice of isolating submodels in order to study them independently.

Chapter 5: *Information Equivalence.*

The detection of information equivalences is performed by checking, for each conditional independency $X \perp\!\!\!\perp Z \mid Y$ that is found together with dependency $X \not\!\perp\!\!\!\perp Z$, whether $Y \perp\!\!\!\perp Z \mid X$ or $X \perp\!\!\!\perp Y \mid Z$ also holds. Both cases indicate a violation of the intersection condition, which I indicated as an information equivalence. One can question the utility of the augmented Bayesian model and argue that he or she prefers to take out the deterministically related variables from the data. Even then it is safe to add the check of the intersection condition. Since its occurrence is deadly for the constraint-based algorithms; a violation results in the removal of the two edges connecting the information equivalent variables with the reference variable. A violation might be caused by unknown deterministic relations or quasi-deterministic relations. Analogously, other checks could be built in to protect the algorithm for faulty assertions.

The extensions for information equivalences do not check for deterministic relations, since they are also characterized by information equivalences. Deterministic relations can however be detected directly using kernel density estimation. If $P(X \mid \boldsymbol{Y})$ is below a threshold, the uncertainty of X can be regarded as zero when $\boldsymbol{Y}$ is known. Hence, variables $\boldsymbol{Y}$ completely determine the state of X.

The complexity criterion, to choose among information-equivalent relations, can also be used in the scoring-based algorithms [Korb and Nicholson, 2003]. Information equivalences lead to models having equal scores. Then, the complexity of the edges not appearing in every model enables an objective selection criterion.

Chapter 7: *Qualitative Multivariate Analysis.*

The results for the performance analysis underlined the importance of an integration of statistical techniques. The available statistical technologies can be extended with for example, form-free regression or algorithms for automatic outlier detection.

Mutual information was used as a form-free measure of association, in contrast with Pearson's correlation coefficient. But the estimation of it is more error-prone. A good calibration, per problem, is still necessary. More fine-tuning is necessary to create a more reliable independence test.

Chapter 8: *Causal Inference for Performance Analysis.*

The results so far are limited to the analysis of specific applications. In order to create models which are valid for multiple applications, *context* should be taken into account. As we have seen, parts of the performance models are application-dependent, such as cache miss behavior (Sec. 8.4.4). Nevertheless will these submodels be reusable in specific applications exhibiting the same behavior. The feasibility of learning contextual causal relations was already successfully studied by my colleague Borms [2006]. The results need, however, to be integrated in the current overall modeling process.

The results of a performance analysis should be used as the knowledge for the optimization of applications and systems. This would really prove the value of causal inference.

Besides quantitative information, other types of qualitative information would also give valuable information, such as the form of the curves or the occurrence of different regimes in the system's behavior.

Chapter 9: *The Genericity of Performance Models.*

The study of the match of communication and network topology showed that regularities limit the existence of generic characteristics and models. It must be further researched whether patterns are also the reason for the failure of generic models for the performance prediction of applications and computer systems. Also the difficulty of modeling internet traffic can be analyzed according to the same philosophy.

Finally, we might ask ourselves why humans instantly see that a shift on a ring can be performed in only one step, just like a broadcast on a star. *How do we get to understand so easily that a broadcast on a ring or a shift on a star is very inefficient? How do we, humans, reason about the matching problem?* The approach of my research was the application of causal inference on the results obtained by simulation. Human reasoning, however, functions differently. It is still a great mystery why reasoning with regularities is such a natural thing for us. A question that I posed at the beginning of this work, but which I have to leave unanswered. The quest continues...

List of Figures

Bibliography

Russell L. Ackoff. *Redesigning the Future: A Systems Approach to Societal Problems.* John Wiley & Sons: New York, 1974.

Michael D. Adams. Coding of still pictures, the jpeg2000 still image compression standard, iso/iec jtc1/sc29/wg1 n2412. http://www.ece.uvic.ca/∼mdadams, 2001. URL `citeseer.comp.nus.edu.sg/86478.html`.

Muhammad H. Arshad and Philip K. Chan. Identifying outliers via clustering for anomaly detection. Technical Report CS-2003-19, Florida Institute of Technology, june 2003.

Rosa M. et all. Badia. DIMEMAS: Predicting MPI applications behavior in grid environments. In *Workshop on Grid Applications and Programming Tools (GGF8)*, 2003.

Jerry Banks, John Carson, Barry L. Nelson, and David Nicol. *Discrete-Event Simulation - fourth edition.* Prentice Hall, 2005.

Y. M. M. Bishop, S. E. Fienberg, and Holland. *Discrete multivariate analysis. Theory and practice.* Cambridge: MIT Press, 1975.

Joris Borms. Integratie van contextuele informatie in causale modellen en ondersteunende leeralgoritmes. Master's thesis, Vrije Universiteit Brussel, 2006.

Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in Bayesian networks. In *Uncertainty in Artificial Intelligence*, pages 115–123, 1996. URL `citeseer.ist.psu.edu/article/boutilier96contextspecific.html`.

S. Browne, J.J. Dongarra, N. Garner, G. Ho, and P. Mucci. A portable programming interface for performance evaluation on modern processors.

International Journal of High Performance Computing Applications, 14 (3):189–204, 2000.

J. Mark Bull. A hierarchical classification of overheads in parallel programs. In Innes Jelly, Ian Gorton, and Peter R. Croll, editors, *Software Engineering for Parallel and Distributed Systems*, volume 50 of *IFIP Conference Proceedings*, pages 208–219. Chapman & Hall, 1996. ISBN 0-412-75740-0.

Mario Bunge. *Causality and Modern Science.* Dover Publications, New York, 1979.

Kenneth L. Calvert, M. B. Doar, and Ellen W. Zegura. Modeling internet topology. *IEEE Communication Magazine*, pages 160–163, 1997.

Laura C. Carrington, Michael Laurenzano, Allan Snavely, Roy L. Campbell, and Larry P. Davis. How well can simple metrics represent the performance of HPC applications? In *Proc. of the 2005 ACM/IEEE conference on Supercomputing*, page 48, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 1-59593-061-2.

Nancy Cartwright. What is wrong with Bayes nets? *The Monist*, pages 242–264, 2001.

Gregory J. Chaitin. On the length of programs for computing finite binary sequences: statistical considerations. *J. ACM*, 16(1):145–159, 1969. ISSN 0004-5411.

C. Chen and L-M. Liu. Joint estimation of model parameters and outlier effects in time series. *Journal of the American Statistical Association*, 88:284–297, 1993.

A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345–363, 1936.

Barry R. Cobb and Prakash P. Shenoy. Inference in hybrid Bayesian networks with deterministic variables. In *in P. Lucas (ed.), Proceedings of the Second European Workshop on Probabilistic Graphical Models (PGM-04)*, pages 57–64, 2004.

Ira Cohen, Jeffrey S. Chase, Moisés Goldszmidt, Terence Kelly, and Julie Symons. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *OSDI*, pages 231–244, 2004.

Joshua W. Comley and David L. Dowe. General Bayesian networks and asymmetric languages. In *Proc. 2nd Hawaii International Conference on Statistics and Related Fields*, 2003.

Joshua W. Comley and David L. Dowe. Minimum message length and generalised Bayesian networks with asymmetric languages. In *In Advances in Minimum Description Length: Theory and Applications, P.D. Grünwald, I.J. Myung, and M.A. Pitt, Eds.* MIT Press, 2005.

Thomas M. Cover and Joy A Thomas. *Elements of Information Theory.* John Wiley & Sons, Inc., 1991.

R.G. Cowell, A.P. Dawid, S.L. Lauritzen, and D.J. Spiegelhalter. *Probabilistic Networks and Expert Systems.* Springer, New York, 1999.

Mark Crovella and Thomas J. LeBlanc. Parallel performance using lost cycles analysis. In *SC*, pages 600–609, 1994.

A. P. Dawid. Prequential analysis, stochastic complexity and Bayesian inference. In *In Bayesian Statistics, J. M. Bernardo, J. Berger, A. P. Dawid, and A. F. M. Smith, Eds*, pages 109–126. Oxford University Press, 1992.

Rina Dechter and Robert Mateescu. Mixtures of deterministic-probabilistic networks and their and/or search space. In *AUAI '04: Proc. of the 20th conf. on Uncertainty in artificial intelligence*, pages 120–129, Arlington, Virginia, United States, 2004. AUAI Press. ISBN 0-9749039-0-6.

M. Druzdzel and H. Simon. Causality in Bayesian belief networks. In *Proceedings of Ninth Conference on Uncertainty in Artificial Intelligence,* Washington, DC, pages 3–11. Morgan Kaufmann, 1993.

Antonio Espinosa, Tomàs Margalef, and Emilio Luque. Automatic detection of parallel program performance problems. In José M. L. M. Palma, Jack Dongarra, and Vicente Hernández, editors, *VECPAR*, volume 1573 of *Lecture Notes in Computer Science*, pages 365–377. Springer, 1998. ISBN 3-540-66228-6.

Thomas Fahringer and Clovis Seragiotto. Automatic search for performance problems in parallel and distributed programs by using multi-experiment analysis. In Sartaj Sahni, Viktor K. Prasanna, and Uday Shukla, editors, *HiPC*, volume 2552 of *Lecture Notes in Computer Science*, pages 151–162. Springer, 2002. ISBN 3-540-00303-7.

R. A. Fisher. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London, Ser. A*, 222:309368, 1922. URL `http://digital.library.adelaide.edu.au/coll/special/fisher/18pt1.pdf`.

Sally Floyd and Vern Paxson. Difficulties in simulating the internet. *IEEE/ACM Trans. Netw.*, 9(4):392–403, 2001. ISSN 1063-6692.

D.A. Freedman and P. Humphreys. Are there algorithms that discover causal structure? *Synthese*, 121:2954, 1999.

Péter Gács, J. Tromp, and Paul M. B. Vitányi. Algorithmic statistics. *IEEE Trans. Inform. Theory*, 47(6):2443–2463, 2001.

D. Geiger. *Graphoids: A Qualitative Framework for Probabilistic Inference.* PhD thesis, University of California, Los Angeles, 1990.

P. Grünwald. *The Minimum Description Length Principle and Reasoning under Uncertainty, ILLC Dissertation series 1998-03.* PhD thesis, University of Amsterdam, 1998.

P. Grünwald, I.J. Myung, and M.A. Pitt. *A Tutorial Introduction to the Minimum Description Length Principle.* MIT Press, 2005.

Peter Grünwald and Paul M. B. Vitányi. Kolmogorov complexity and information theory. with an interpretation in terms of questions and answers. *Journal of Logic, Language and Information*, 12(4):497–529, 2003.

Anthony J. G. Hey, Alistair N. Dunlop, and Emilio Hernández. Realistic parallel performance estimation. *Parallel Computing*, 23(1-2):5–21, 1997.

Roger A. Horn and Charles R. Johnson. *Matrix Analysis.* Cambridge University Press, 1985.

ISO/IEC. Iso/iec fcd15444-1, information technology - jpeg 2000 image coding system. `http://www.jpeg.org`, 2002.

E. T. Jaynes. Information theory and statistical mechanics. *Physical Review*, 106(4):620+, May 1957.

E. T. Jaynes. *Probability Theory: The Logic of Science.* Cambridge University Press, 2003.

Karen L. Karavanic, Jussi Myllymaki, Miron Livny, and Barton P. Miller. Integrated visualization of parallel program performance data. *Parallel Computing*, 23(1–2):181–198, 1997.

Darren J. Kerbyson, Harvey J. Wasserman, and Adolfy Hoisie. Exploring advanced architectures using performance prediction. In *IWIA '02: Proceedings of the International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA'02)*, page 27, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1635-1.

Andrei N. Kolmogorov. Three approaches for defining the concept of information quantity. *Problems of Information Transmission*, 1(1):1–7, 1965.

Kevin B. Korb and Ann E. Nicholson. *Bayesian Artificial Intelligence.* CRC Press, 2003.

Kevin B. Korb and Erik Nyberg. The power of intervention. *Minds and Machines*, 16(3):289–302, 2006. ISSN 0924-6495.

Harald Kosch, László Böszörményi, and Hermann Hellwagner, editors. *Euro-Par 2003. Parallel Processing, 9th International Euro-Par Conference, Klagenfurt, Austria, August 26-29, 2003. Proceedings*, volume 2790 of *Lecture Notes in Computer Science*, 2003. Springer. ISBN 3-540-40788-X.

Vipin Kumar and Anshul Gupta. Analyzing scalability of parallel algorithms and architectures. *Journal of Parallel and Distributed Computing (special issue on scalability)*, 22(3):379–391, 1994.

W. Lam and F. Bacchus. Learning Bayesian belief networks: An approach based on the MDL principle. *Computational Intelligence*, 10:269–293, 1994.

Alexey L. Lastovetsky and Ravi Reddy. Data partitioning with a realistic performance model of networks of heterogeneous computers. In *IPDPS*. IEEE Computer Society, 2004. ISBN 0-7695-2132-0.

Jan Lemeire. Documentation of EPPA tool (Experimental Parallel Performance Analysis). `http://parallel.vub.ac.be/eppa`, 2004.

Jan Lemeire. Documentation of EPDA tool (Experimental Performance Data Analysis). `http://parallel.vub.ac.be/epda`, 2006.

Jan Lemeire, Andy Crijns, John Crijns, and Erik F. Dirkx. A refinement strategy for a user-oriented performance analysis. In Dieter Kranzlmüller, Péter Kacsuk, and Jack Dongarra, editors, *PVM/MPI*, volume 3241 of *Lecture Notes in Computer Science*, pages 388–396. Springer, 2004. ISBN 3-540-23163-3.

Ming Li and Paul M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications.* Springer Verlag, 1997.

Gabriel Marin and John Mellor-Crummey. Cross-architecture performance predictions for scientific applications using parameterized models. In *SIGMETRICS '04/Performance '04: Proceedings of the joint international conference on Measurement and modeling of computer systems*, pages 2–13, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-873-3.

Ernesto San Martín, Michel Mouchart, and Jean-Marie Rolin. Ignorable common information, null sets and basu's first theorem. *Sankhya: The Indian Journal of Statistics*, 67:674–698, 2005.

Bernd Mohr and Felix Wolf. KOJAK - a tool set for automatic performance analysis of parallel programs. In Kosch et al. [2003], pages 1301–1304. ISBN 3-540-40788-X.

Bernd Mohr and Felix Wolf. Kojak - a tool set for automatic performance analysis of parallel programs. In Kosch et al. [2003], pages 1301–1304. ISBN 3-540-40788-X.

W. E. Nagel, A. Arnold, M. Weber, H. C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and analysis of MPI resources. *Supercomputer*, 12(1):69–80, 1996.

Zsolt Nemeth, Gabor Gombas, and Zoltan Balaton. Performance evaluation on grids: Directions issues and open problems. In *12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'04) February 11 - 13, 2004, A Coruna, Spain*, 2004.

Inc NetPredict. Common mistakes in performance analysis, white paper. NetPredict Inc, 2003.

Isaac Newton. *Philosophiae Naturalis Principia Mathematica.* 1687.

Robert M. Oliver and James Q. Smith. *Influence Diagrams, Belief Nets and Decision Analysis.* Wiley, 1990.

Cherri M. Pancake. Applying human factors to the design of performance tools. In Patrick Amestoy, Philippe Berger, Michel J. Daydé, Iain S. Duff, Valérie Frayssé, Luc Giraud, and Daniel Ruiz, editors, *Euro-Par*, volume 1685 of *Lecture Notes in Computer Science*, pages 44–60. Springer, 1999. ISBN 3-540-66443-2.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* San Mateo, CA, Morgan Kaufman Publishers, 1988.

J. Pearl. *Causality. Models, Reasoning, and Inference.* Cambridge University Press, 2000.

K. R. Popper. *The Logic of Scientific Discovery.* Basic Books, New York, 1959.

S. Pynnonen. Detection of outliers in regression analysis by information criteria. Technical Report Discussion Papers 146, University of Vaasa, 1992.

Joseph Ramsey, Jiji Zhang, and Peter Spirtes. Adjacency-faithfulness and conservative causal inference. In *Proceedings of the 22nd Annual Conference on Uncertainty in Artificial Intelligence (UAI-06)*, Arlington, Virginia, 2006. AUAI Press.

D. A. Reed, R. A. Aydt, R. J. Noe, P. C. Roth, K. A. Shields, B. W. Schwartz, and L. F. Tavera. Scalable Performance Analysis: The Pablo Performance Analysis Environment. In *Proc. Scalable Parallel Libraries Conf.*, pages 104–113. IEEE Computer Society, 1993.

J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

J. Rissanen. *Stochastic Complexity in Statistical Enquiry.* World Scientific, Singapore., 1989.

P.J. Rousseeuw and B.C. van Zomeren. Unmasking multivariate outliers and leverage points. *Journal of the American Statistical Association*, 85: 663–639, 1990.

Sekhar R. Sarukkai, Jerry Yan, and Jacob K. Gotwals. Normalized performance indices for message passing parallel programs. In *ICS '94: Proceedings of the 8th international conference on Supercomputing*, pages 323–332, New York, NY, USA, 1994. ACM Press. ISBN 0-89791-665-4.

Richard Scheines, Peter Spirtes, Clark Glymour, Christopher Meek, and Thomas Richardson. *TETRAD 3: Tools for Causal Modeling - User's Manual.* `http://www.phil.cmu.edu/projects/tetrad/tet3/master.htm`, 1996.

Shohei Shimizu, Aapo Hyvarinen, Yutaka Kano, and Patrik O. Hoyer. Discovery of non-gaussian linear causal models using ica. In *Proceedings of the 21th Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 525–53, Arlington, Virginia, 2005. AUAI Press.

Bill Shipley. *Cause and Correlation in Biology.* Cambridge University Press, 2000.

A. Snavely, N. Wolter, and L. Carrington. Modeling application performance by convolving machine signatures with application profiles. In *WWC '01: Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 149–156, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7803-7315-4.

Allan Snavely, Laura Carrington, Nicole Wolter, Jesús Labarta, Rosa M. Badia, and Avi Purkayastha. A framework for performance modeling and prediction. In *SC*, pages 1–17, 2002.

Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI: The Complete Reference.* MIT Press, 1996.

Ray J. Solomonoff. A formal theory of inductive inference. part II. *Information and Control*, 7(2):224–254, 1964. URL `citeseer.ist.psu.edu/solomonoff64formal.html`.

Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction, and Search.* Springer Verlag, 2nd edition, 1993.

Peter Spirtes, Clark Glymour, Richard Scheines, and Joseph Ramsey. The TETRAD project. `http://www.phil.cmu.edu/projects/tetrad/`.

Wolfgang Spohn. Bayesian nets are all there is to causal dependence. In *In Stochastic Causality, Maria Carla Galaviotti, Eds.* CSLI Lecture Notes, 2001. URL `citeseer.ist.psu.edu/475768.html`.

Michael R. Steed and Mark J. Clement. Performance prediction of pvm programs. In *IPPS '96: Proceedings of the 10th International Parallel Processing Symposium*, pages 803–807, Washington, DC, USA, 1996. IEEE Computer Society. ISBN 0-8186-7255-2.

Harold S. Stone. *High-performance computer architecture (2nd ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990. ISBN 0-201-51377-3.

Milan Studeny. On non-graphical description of models of conditional independence structure. In *HSSS Workshop on Stochastic Systems for Individual Behaviours*, Louvain la Neuve, Belgium, January 2001.

David S. Taubman and Michael W. Marcellin. *JPEG 2000: Image Compression Fundamentals, Standards and Practices.* Kluwer Academic, Boston, USA, 2002.

Jin Tian and Judea Pearl. A general identification condition for causal effects. In *AAAI/IAAI*, pages 567–573, 2002.

Hong Linh Truong and Thomas Fahringer. Performance analysis for MPI applications with SCALEA. In *In Proc. of the 9 th European PVM/MPI Conf., Linz, Austria (September 2002).*, 2002a.

Hong Linh Truong and Thomas Fahringer. SCALEA: A performance analysis tool for distributed and parallel programs. In Burkhard Monien and Rainer Feldmann, editors, *Euro-Par*, volume 2400 of *Lecture Notes in Computer Science*, pages 75–85. Springer, 2002b. ISBN 3-540-44049-6.

R. S. Tuminaro, M. Heroux, S. A. Hutchinson, and J. N. Shadid. Official Aztec user's guide: Version 2.1. Technical Report SAND95-1559, Sandia National Laboratories, dec 1999.

Nikolai K. Vereshchagin and Paul M. B. Vitányi. Kolmogorov's structure functions with an application to the foundations of model selection. In *FOCS*, pages 751–760. IEEE Computer Society, 2002. ISBN 0-7695-1822-2.

Paul M. B. Vitányi. Meaningful information. In Prosenjit Bose and Pat Morin, editors, *ISAAC*, volume 2518 of *Lecture Notes in Computer Science*, pages 588–599. Springer, 2002. ISBN 3-540-00142-5.

Paul M. B. Vitányi. Algorithmic statistics and Kolmogorov's structure functions. In *In Advances in Minimum Description Length: Theory and Applications, P.D. Grünwald, I.J. Myung, and M.A. Pitt, Eds.* MIT Press, 2005.

Chris S. Wallace. *Statistical and Inductive Inference by Minimum Message Length.* Springer, 2005.

Chris S. Wallace and David L. Dowe. An information measure for classification. *Computer Journal*, 11(2):185–194, 1968.

M.P. Wand and M.C. Jones. *Kernel Smoothing*. Chapman & Hall, London, UK, 1995.

Jon Williamson. *Bayesian Nets And Causality: Philosophical And Computational Foundations*. Oxford University Press, 2005. ISBN 019853079X.

Yang Xiang, S. K. Wong, and N. Cercone. Critical remarks on single link search in learning belief networks. In *Proceedings of the 12th Annual Conference on Uncertainty in Artificial Intelligence (UAI-96)*, pages 564–571, San Francisco, CA, 1996. Morgan Kaufmann Publishers.

Jerry Yan, Sekhar Sarukkai, and Pankaj Mehra. Performance measurement, visualization and modeling of parallel and distributed programs using the aims toolkit. *Software - Practice and Experience*, 25 (4):429–461, 1995. ISSN 0038-0644. URL `citeseer.ist.psu.edu/yan95performance.html`.

Ellen W. Zegura, Kenneth L. Calvert, and Michael J. Donahoo. A quantitative comparison of graph-based models for Internet topology. *IEEE/ACM Transactions on Networking*, 5(6):770–783, 1997. URL `citeseer.ist.psu.edu/zegura97quantitative.html`.