Learning Bayesian Networks with Large-scale Problems and Computing Paradigms

Jacinto Arias Martínez

Ph.D. Thesis, February 2019
PhD. Program:
Doctorado en Tecnologías Informáticas Avanzadas

PhD. Thesis Dissertation:
Learning Bayesian Networks with Large-scale Problems and Computing Paradigms

PhD. Student:
Jacinto Arias Martínez

Advisors:
José A. Gámez and José M. Puerta
To Oli and Gor, who never let me write more than I should.
Acknowledgements

This work is dedicated to the people that have motivated me throughout my life to follow the path of curiosity and science.

To Sonsoles, who sparked the beauty of knowledge and allowed a little boy to tinker freely with books and computers. For a life of constant support and dedication, I am deeply thankful.

To Luismi, for breaking the barrier of boring school lessons, for treating us like equals in the difficult years of puberty and for later becoming a good lifelong friend.

To Antonio, for the best maths lessons ever and especially for being a model of intelligence and humility. A true inspiring person that I will always look up to.

To José Antonio, José Miguel, Luis and Ismael, who discovered me the path I wanted to pursue for the rest of my life and welcomed me into their scientific family.

Thank you, to every person that enjoys teaching, there are many of you that I cannot directly mention here, but you can be sure you own a part of every accomplishment that we achieve.

It would have been impossible for me to follow this path alone. Learning is a social activity and I am grateful for having kept the best company.

To Jorge for being an always unreachable source of inspiration, I hope to keep learning from your for many years to come. To Javi for all those times you lent me your brain when mine did not allow for more. We have finished exhausted and this is just beginning.
To my colleagues at the lab and my teammates at demosense, for all the support and tolerance. To the hospitable Danish and Danish-Spanish people, for giving me new perspectives. And of course, thanks to my family, specially my wife, mother, father and brother, for their support and understanding.

This work has been financially supported by the MECD grant FPU13/00202 and has been developed within the context of projects TIN2010-20900-C04-03, TIN2013-46638-C3-3-P and TIN2016-77902-C3-1-P supported by FEDER funds and the Spanish Government (MINECO) and by JCCM through project PEII-2014-049-P.
Contents

Contents v
List of Figures ix
List of Tables xiii
List of Algorithms xv

1 Introduction 1
  1.1 Motivation .............................................. 1
  1.2 Outline of the dissertation .............................. 3

2 Background 5
  2.1 Notation .................................................. 5
  2.2 Bayesian Networks ...................................... 5
  2.3 Learning Bayesian Networks ............................. 7
    2.3.1 Structural Learning ................................. 7
    2.3.2 Parametric Learning ............................... 10
  2.4 Markov Random Fields .................................. 12
  2.5 Supervised Classification with Bayesian Networks .... 13
    2.5.1 Bayesian Network Classifiers ...................... 13
  2.6 Multi-dimensional Classification ....................... 21
    2.6.1 Evaluation ........................................... 22
    2.6.2 Transformation methods ........................... 23
    2.6.3 Chain classifiers .................................. 24
    2.6.4 Adaptation methods ................................ 25
CONTENTS

2.6.5 Multi-dimensional Bayesian Networks classifiers .... 25
2.6.6 Ensembles ........................................... 26
2.7 Notation Summary ..................................... 27

3 Computing Paradigms for Large-scale Problems 29
  3.1 Introduction ............................................. 29
  3.2 The MapReduce Paradigm ............................... 31
  3.3 The Big Data Ecosystem ................................. 33
    3.3.1 In-memory Distributed Computing ................. 34

4 Efficiency trade-off in Structural Learning 37
  4.1 Introduction ............................................. 38
  4.2 Constrained Hill Climbing Methods .................... 38
  4.3 Proposal ................................................... 40
    4.3.1 Releasing Constraints from Variable Neighbourhood
          Levels ............................................... 44
    4.3.2 Limiting the Forbidden Parent Set Size ........... 44
    4.3.3 Releasing Constraints Based on the Confidence in Con-
          ditional Independence Tests ....................... 48
  4.4 Experimental Evaluation ................................ 49
    4.4.1 Algorithms .......................................... 50
    4.4.2 Implementation and Running Environment ........... 50
    4.4.3 Performance Indicators ............................. 51
    4.4.4 Experiments ......................................... 51
    4.4.5 Empirical Results ................................ 52
    4.4.6 Combining Marginal Tests with other Modifications . 55
    4.4.7 Statistical Tests ................................ 57
  4.5 Conclusions .............................................. 59
  4.6 Reproducibility and Extended Results .................. 61
  4.7 Detailed Results ....................................... 61

5 Factor-based Multi-dimensional Classifier 67
  5.1 Introduction ............................................. 68
  5.2 A General Framework for Multi-Dimensional Classification .. 69
## CONTENTS

5.2.1 Preprocessing Capabilities .................................. 73
5.3 Scalable Learning and Inference .............................. 74
5.4 Experimental Evaluation ................................. 78
  5.4.1 Experimental Set up .................................... 78
  5.4.2 Comparison Between Pruning Strategies ............. 81
  5.4.3 Comparison with State-of-the Art Classifiers ......... 87
5.5 Conclusions .................................................. 93
5.6 Reproducibility and Extended Results ...................... 93

6 Distributed Learning of BNCs .................................. 95
  6.1 Introduction .................................................. 95
  6.2 Distributed Bayesian Network Classifiers ............... 97
    6.2.1 Horizontal Parallelism in MapReduce .......... 99
    6.2.2 Vertical parallelism in MapReduce ........... 102
  6.3 Empirical Evaluation ...................................... 103
    6.3.1 Execution Environment ............................. 104
    6.3.2 Synthetic Data Experiments ................... 105
    6.3.3 Real Data Experiments ......................... 109
    6.3.4 Sequential Implementation ..................... 110
    6.3.5 Vertical Parallelism ............................... 112
    6.3.6 In-Memory vs Disk Frameworks ................. 112
    6.3.7 Size of the Models ................................ 113
  6.4 Conclusions ................................................. 116
  6.5 The spark-bnc Spark Package .............................. 117

7 New Ensemble Approaches for BNCs .......................... 119
  7.1 Introduction ................................................ 120
  7.2 Decomposition of Classification Error in terms of Bias and Variance ........................................... 121
  7.3 Bias and Variance in Random Forest ................... 123
  7.4 KEDE Under the Ensemble Perspective ................ 125
    7.4.1 On the Effectiveness of Model Selection .......... 130
  7.5 Variance Reduction in Large Datasets ................... 132
CONTENTS

7.6 Alternative Approaches for Ensembles of Bayesian Network Classifiers ........................................ 136
7.7 Conclusions ................................................. 138
7.8 Reproducibility .............................................. 138

A R package: exreport ................................. 143
B Publications .............................................. 145
References ................................................. 147
## List of Figures

2.1 Naive Bayes classifier (NB) ........................................ 14
2.2 Tree-augmented naive Bayes classifier (TAN or kDB with $k = 1$) 16
2.3 Individual models of an A1DE classifier for $n = 4$ .............. 18

3.1 Graphical representation of a MapReduce job. .................. 32
3.2 Word count example in MapReduce. .............................. 33

4.1 BDeu score and computed statistics comparison for FastCHC and Hill Climbing ........................................ 43
4.2 Distribution of forbidden parents against dimensionality of the training data. ..................................................... 46
4.3 Ratio between the total forbidden parents discovered by FastCHC against its first iteration. ................................. 47
4.4 Trade-off between score metric calls and score metric for the first set of experiments. ....................................... 55
4.5 Trade-off between score metric calls and score metric for the second set of experiments. ................................... 57
4.6 Ranks computed for the Friedman test regarding BDeu Score. 59
4.7 Ranks computed for the Friedman test regarding score metric calls. ................................................................. 60
4.8 Approximate execution times for HC, FastCHC and each modification on each dataset. ............................... 64

5.1 FMC structure example with naive Bayes as base classifier. . . 72
5.2 Average rank obtained for each pruning strategy in FMC. 83
<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3</td>
<td>Number of pairs selected by each pruning strategy in FMC.</td>
<td>85</td>
</tr>
<tr>
<td>5.4</td>
<td>Proportion between selected pairs and classes in the dataset for each pruning strategy in FMC.</td>
<td>85</td>
</tr>
<tr>
<td>5.5</td>
<td>Breakdown of the runtime in seconds for the different stages in the FMC framework.</td>
<td>86</td>
</tr>
<tr>
<td>5.6</td>
<td>Logarithmic runtime measured in seconds for the selected models and datasets when using exact and approximate inference. Missing bars correspond with experiments that hit maximum wall time.</td>
<td>89</td>
</tr>
<tr>
<td>5.7</td>
<td>Comparison of execution times between FMC and state-of-the-art classifiers.</td>
<td>92</td>
</tr>
<tr>
<td>6.1</td>
<td>Graphical diagram of the horizontal MapReduce algorithm.</td>
<td>100</td>
</tr>
<tr>
<td>6.2</td>
<td>Graphical diagram of the combination of horizontal and vertical parallelism to the MapReduce algorithm.</td>
<td>104</td>
</tr>
<tr>
<td>6.3</td>
<td>Structure of a random synthetic network generated where the number of attributes is indefinite.</td>
<td>106</td>
</tr>
<tr>
<td>6.4</td>
<td>Results for each of the 4 learning algorithms and each synthetic dataset.</td>
<td>107</td>
</tr>
<tr>
<td>6.5</td>
<td>Execution time (logarithmic scale) for all combination of number of attributes (y-axis), number of instances (y-axis) and number of parallel tasks (different plots). The color represents the execution time where blue is the lowest towards orange, red and black being the greatest.</td>
<td>108</td>
</tr>
<tr>
<td>6.6</td>
<td>Speed-up and efficiency for multiple executions of MapReduce Algorithms.</td>
<td>109</td>
</tr>
<tr>
<td>6.7</td>
<td>Execution time for each algorithm and real dataset.</td>
<td>110</td>
</tr>
<tr>
<td>6.8</td>
<td>Comparison between sequential and MapReduce implementations.</td>
<td>111</td>
</tr>
<tr>
<td>6.9</td>
<td>Comparison between different proportions of horizontal and vertical MapReduce tasks.</td>
<td>113</td>
</tr>
<tr>
<td>6.10</td>
<td>Comparison between Apache Hadoop and Apache Spark implementations.</td>
<td>114</td>
</tr>
</tbody>
</table>
6.11 Sizes of the resulting models for each classifier and problem. . 115

7.1 Bias and variance decomposition of error for a random forest ensemble. ................................................................. 126
7.2 Bias and variance decomposition of error for A1DE. .............. 128
7.3 Bias and variance differences between ensemble and individual models. ................................................................. 129
7.4 Incremental selection of models in AkDE for different criteria. . 131
7.5 Evolution of bias and variance for large data samples in random forest. ................................................................. 133
7.6 Evolution of bias and variance for large data samples in BNCs. . 135
7.7 Bias and variance decomposition of error for a RKDB with $k = 1$. ................................................................. 139
7.8 Bias and variance decomposition of error for a RKDB with $k = 2$. ................................................................. 140
7.9 Bias and variance differences between ensemble and individual models including RKDB. ............................................ 141
List of Tables

2.1 Notation summary .................................................. 28

4.1 Main characteristics of the networks used in the experiments. .... 52

4.2 BDeu Score and Score Metric Calls ratio computed for each
algorithm with respect to Hill Climbing for the first set of
experiments. ......................................................... 54

4.3 BDeu Score and Score Metric Calls ratio computed for each
algorithm with respect to Hill Climbing for the second set of
experiments. ......................................................... 56

4.4 Statistical analysis for BDeu score results for the second set
of experiments ....................................................... 58

4.5 Detailed results for BDeu score from the first set of experiments. 62

4.6 Detailed results for score metric calls from the first set of ex-
periments ......................................................... 63

4.7 Detailed results for BDeu score calls from the second set of
experiments ......................................................... 65

4.8 Detailed results for score metric calls from the second set of
experiments ......................................................... 66

5.1 Effect of the individual discretization feature selection tech-
niques applied in a FMC classifier. ................................ 74

5.2 Multi-label/Multi-dimensional datasets used in the experiments. 80

5.3 Results on global accuracy for various pruning strategies in
FMC. ............................................................... 82
5.4 Results on Hamming accuracy for various pruning strategies in FMC. ................................................................. 82
5.5 Comparison between exact and approximate inference in an FMC. ................................................................. 88
5.6 Global accuracy for the literature classifiers and FMC. ....... 90
5.7 Hamming accuracy for the literature classifiers and FMC. .... 90
5.8 Statistical tests comparing the state-of-the art algorithms to FMC. ................................................................. 91

6.1 Computational and spatial complexity of BNCs. ............... 98
6.2 Properties of the synthetic datasets generated for the study. . 106
6.3 Properties of the large-scale real datasets included in the experiments. ......................................................... 109
6.4 Sizes of the resulting model for the A1DE and A2DE. ....... 115

7.1 Properties of the datasets used in the experiments. .......... 124
7.2 Statistical comparison between BNCs and random forest. .. 129
7.3 Statistical comparison between different model selection criteria in A1DE .................................................. 131
7.4 Statistical comparison between RkDB and other ensembles. . 137
List of Algorithms

2.1 Structural learning of BNs by using a Hill Climbing algorithm [Gámez et al., 2012] ........................................... 11

4.1 Structural learning of BNs by using a Constrained Hill Climbing (CHC) algorithm [Gámez et al., 2012] ............... 41

4.2 Structural learning of BNs by using a Fast Constrained Hill Climbing (FastCHC) algorithm [Gámez et al., 2012] .... 42

5.1 Forward greedy search with BDeu score model selection for FMC. 79
Chapter 1

Introduction

1.1 Motivation

Anyone working in machine learning requires a particular balance between multiple disciplines. A solid mathematical background, proficiency with computer science tools and a deep expert domain are one of the most commonly attributed. We can find a large number of real problems that can be solved using machine learning techniques and usually require the involvement of many experts, where each of them contribute their specialty to the particular solution.

We can say that machine learning depends heavily on technology, however the opposite is true as well. Artificial intelligence has motivated some of the most important advances in computer science of the last years. From the hardware point of view, computing power has been key in the development of this discipline and machine learning is perhaps the field which utilises more specialized architectures such as GPUs and the recent TPUs for deep learning.

In addition to computing advances, the last decades have witnessed the arising of the data revolution, with the advent of big data as a scientific and business discipline, it is perhaps one of the most widely adopted technologies since the Internet. Again, the stimulus of this field has propitiated the cheapening and optimization of data storage technology, leading humanity
to produce and store more information in a couple of years than in all known history.

Methodological computer science has evolved as well, with the appearance of new data science languages and libraries that have shaped the profiles of many professionals from different fields. Expressiveness and programming models that are closer to mathematical language are required in order to create the next wave of intelligent software and the ingestion and processing of such a vast amount of data.

All of this, happening at an ever increasing rate that permanently menaces every new technology with becoming obsolete.

Scientific research in the field of machine learning studies new approaches to improve existing algorithms as well as new unexplored approaches to improve this domain. The field is reaching maturity with many results been already adopted by the industry. However, at the current rate of technological expanse we should revisit and adapt existing techniques to new software and hardware developments.

In this dissertation we will focus on computational scalability, specifically in the approaches to learn from massive amounts of data in large distributed computing technologies. This has been an active field of research both in academia and industry with ground breaking results that have defined what we know today as big data. We will focus on a particular framework known as probabilistic graphical models and especially on Bayesian networks [Pearl, 2014].

This formalism combines a mathematically sound background such as the probability theory and many other desirable properties, like the capacity for providing explanation and interpretation of their results, something that many of the algorithms lack or provide artificially. This, in addition to a range of successful practical applications has gained it a recognised spot in the machine learning community.

However, to the best of our knowledge, this models are not considered by the industry as part of the available big data solutions and are not present in any of the standard libraries that are being applied in real life problems.

While other techniques are perhaps more easily portable to such scalable
technologies, Bayesian networks are scalable both from a theoretical and practical point of view and have been studied from this perspective in the past prior to the popularity of new big data computing frameworks.

We will explore several applications of this formalism to scalable problems in many domains and specific machine learning problem such as structural learning and supervised classification.

1.2 Outline of the dissertation

In Chapter 2 we will begin with a thorough introduction to the basic concepts of probabilistic graphical models and Bayesian networks that we will use during the remaining of the dissertation.

We will follow with an introduction to the technological development of computing architectures in Chapter 3, starting from the single computer and disk machine, to the distributed cluster and finally developing into their evolution to modern specialized big data architectures and cloud computing. We will describe the new computing paradigms that have been developed for these architectures and that are perhaps among the most demanded technological skill in the development and machine learning community. These will be used as well during the dissertation as the technological base on which we will build our proposals.

Chapter 4 proposes a new scalable structural algorithm for Bayesian networks, developed for scalability in traditional computing architectures. We will explore existing scalable algorithms and identify some problems for which we will propose novel solutions. The metrics and algorithms introduced in this sections will serve as a basis for other developments that will be introduced further during the dissertation.

In Chapter 5 we explore applications of probabilistic graphical models to multi-dimensional supervised classification, a problem that is less known in the community but very common in practice. We propose a new framework to deal with this type of data and provide experiments and sound conclusions to apply it to real life problems in scalable architectures.

Following with Chapter 6 we will shift our focus to supervised classifi-
1. INTRODUCTION

cation, which is one of the most demanded problems in the machine learning scalable libraries. We will provide a wide overview of the posibilities of Bayesian networks scalable applications to this problem, specifically for the current state-of-the-art in big data computing. We propose a general framework to deal with massive problems as well as a thorough experimentation in order to provide future practitioners with guidelines to use our results.

The last contribution in Chapter 7 explores original approaches on Bayesian networks, specifically designed to work in big data problems and technologies. We will identify the common pitfalls of existing methods and provide new strategies to exploit the benefits of new highly distributed computing paradigms.
Chapter 2

Background

2.1 Notation

Let us start by defining the basic mathematical notation that will be used consistently throughout this dissertation. The reader will find a summary of the most relevant concepts in Table 2.1 at the end of this chapter, we believe this offers a quick reference to help during the reading of the document.

We will use standard notation where variables are denoted by upper case letters (e.g. $X_i$) and states or values by the same lower case letter (e.g. $x_i$). Bold font is used for sets and $n$-dimensional configurations (e.g. $X$, $\Pi$). Calligraphic fonts are used to denote special and mathematical structures such as databases and graphs (e.g. $\mathcal{D}$, $\mathcal{G}$).

2.2 Bayesian Networks

The formalism of Bayesian Networks (BNs) [Jensen & Nielsen, 2007; Pearl, 2014] is among the most popular knowledge representation techniques, and it has been the tool of choice for designing Intelligent Systems in domains where uncertainty is predominant.

A Bayesian Network can be built by domain experts, induced from data or as a combination of both, providing a unique experience in the creation of machine learning applications. Its graphical representation provides users
2. BACKGROUND

with a clear visualisation of a problem regarding causality and dependency and expresses uncertainty in a familiar, mathematically sound language such as probability theory. An interpretable model (open box model) reinforces the confidence of the users and stimulates its popularity. This is one of the main advantages of BNs, as many of the most popular machine learning frameworks are known to be black box models and lack interpretability.

Its applications range from decision making, automatic classification or risk analysis among others and are utilised broadly in domains such as medicine, healthcare, banking, environmental science or robotics. They have achieved commercial success as well, with many companies developing software based explicitly on this formalism.

The idea behind Bayesian networks is to efficiently represent a domain by encoding the joint probability distribution for its variables. Given a domain of categorical\(^1\) random variables \(X = \{X_1, X_2, \ldots, X_n\}\), a BN is composed by a pair \((G, \theta)\), where \(G\) is a directed acyclic graph (DAG) defined for the set of nodes \(X\). We will use \(X\) to represent both variables and nodes in the graph for the sake of clearness.

The topology of the graph (the arcs in \(E \subseteq X \times X\)) captures conditional (in)dependence relationships among the variables by means of the presence or absence of direct connections between pairs of variables. The second element of the pair, \(\theta\), encodes the quantitative information regarding the variables and the dependences encoded in the graph, concretely, for each variable \(X_i \in X\) and its set of parents in the graph \(\Pi_{X_i}\), a conditional probability distribution table (CPT) \(p(X_i \mid \Pi_{X_i})\) is stored.

This representation can be used to recover the join probability distribution through the Markov rule, which states that each variable \(X\) is conditionally independent of its non-descendant given its parents. According to this we can represent the distribution as the chain products of the corresponding

\(^1\)Probabilistic classifiers are naturally defined for categorical variables, we will not introduce any technique for dealing with continuous domain during this dissertation, so from this point we will assume that all variables in \(X\) are discrete, and therefore each \(X_i \in X\) has a set of states \(\Omega_{X_i}\).
CPT for each node:

\[ p(X_1, \ldots, X_n) = \prod_{i=1}^{n} p(X_i \mid \Pi_{X_i}) \]  

(2.1)

2.3 Learning Bayesian Networks

We can learn a Bayesian network automatically from data using algorithms to discover dependencies in a particular dataset and encoding them with the previously defined structures. Formally, we can state the problem as follows: given a dataset \( D = \{x^{(1)}, \ldots, x^{(m)}\} \) of \( m \) examples defined over the set of variables \( \mathbf{X} = \{X_1, \ldots, X_n\} \), we wish to estimate the parameters of a Bayesian network \( H = (\mathcal{G}, \theta) \).

Both components of a BN are particularly distinct and are usually learnt using an independent two-stage process. We usually learn the graph structure first and then we induce the corresponding parameters accordingly.

2.3.1 Structural Learning

There are two main approaches to learn the structure of a Bayesian network in the literature. Constraint-based [Neapolitan, 2004] builds a DAG trying to satisfy as many independence relationships in the data as possible, usually discovered by employing statistical tests such as \( \chi^2 \) or \( G^2 \).

We will focus on the second approach, known as Score-based learning, in which we employ a search algorithm to look for the fittest candidate network according to a particular scoring criterion. Our search space will be the directed acyclic graphs able to represent our domain of size \( n \). It is easy to see that the problem grows exponentially, in fact, this problem is known to be NP-hard [Chickering, 1996], so heuristic search algorithms are usually considered.

We can formally formulate this problem as the search for the DAG \( \mathcal{G} \) that defines a probability distribution, \( P_{\mathcal{G}}(\mathbf{X}) \), as close as possible to the original distribution found in the data, \( P_D(\mathbf{X}) \), as measured by the score function \( f \).
This is expressed by the following maximisation problem:

$$G^* = \arg \max_{G \in \mathbb{G}} f(G : D)$$  \hfill (2.2)

There are some popular approaches to choose from as suitable scoring criteria. The BIC [Schwarz et al., 1978] scoring metric is based on information theory and measures the log-likelihood of the graph given the data, plus a penalising factor concerning its density. A complete graph can indeed represent any probability distribution and be as good as any other structure in terms of log-likelihood alone, biasing the search algorithm towards it. The formulation of this scoring metric is as follows:

$$BIC(G \mid D) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \log_2 \left( \frac{N_{ijk}}{N_{ij}} \right) - \frac{\log_2(m)}{2} \sum_{i=1}^{n} q_i (r_i - 1)$$  \hfill (2.3)

where we are assuming an estimation of the parameters $\theta$ via maximum likelihood and thus the statistics are obtained by counting frequencies over the $n$ variables and configuration of parents, being $r_i = |\Omega_{X_i}|$ the number of states for each $X_i$ and $q_i = \prod_{X_j \in \Pi_{X_i}} r_j$ the number of configurations over its parents in $G$. $N_{ijk}$ is the number of instances in the dataset $D$ with $X_i$ in its $k$th value and $\Pi_{X_i}$ in its $j$th configuration, and $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$ the marginalization over $X_i$.

Another family of scoring metrics has a Bayesian inspiration and measure the posterior probability of a graph given the database. Developing from the Bayes theorem, only the estimation of $P(D \mid G) \cdot P(G)$ is required for maximisation. Assuming Dirichlet priors, multinomial samples and logarithms we can derive the formulation for the Bayesian Dirichlet scoring criterion (BDe)
[Heckerman et al., 1995b]:

\[
BDe(\mathcal{G} | \mathcal{D}) = \log(P(\mathcal{G})) + \sum_{i=1}^{n} \sum_{j=1}^{q_i} \log \left( \frac{\Gamma(N'_{ij})}{\Gamma(N_{ij} + N'_{ij})} \right) + \sum_{k=1}^{r_i} \log \left( \frac{\Gamma(N_{ijk} + N'_{ijk})}{\Gamma(N'_{ijk})} \right)
\]

(2.4)

A commonly used variant of this metric, the BDeu score, assumes uniform hyperparameters for ease of computation by setting \(N'_{ijk} = ESS/(r_iq_i)\), where \(ESS\) is the equivalent sample size that establishes the strength of the priors.

These metrics can be used to score a candidate network. Still, the evaluation of even a small portion of the search space requires a significant amount of computing power. Opportunely, the majority of the available scoring functions can be computed efficiently through the decomposability property. We can easily see from the formulations that the score can be computed independently for each variable \(X_i\) given its parents and then aggregated for the full network. This property allows efficient implementation of local search algorithms with specific candidate generation strategies, i.e. if we have evaluated a particular network, changing a single arc will only involve the recomputation of the variables whose parents have changed.

The Hill Climbing (HC) algorithm is perhaps one the most relevant and widely adopted options. We can configure a local search algorithm by combining a few elements. The algorithm begins the search from a starting solution, from which we generate neighbouring candidates through local changes. The candidates are evaluated, and the search continues from the network with the maximum score until no more changes are available. Usually, the search stops at a local optimum, strategies such as restarts or randomness can be used to mitigate this effect.

This algorithm can take advantage of decomposable metrics if the generation of new neighbours modify a single arc from the network at a time. This
2. BACKGROUND

way, the score can be computed by reusing the computation carried out in
previous iterations. Concretely, we consider the operations of arc addition,
deletion and reversal between each pair of variables. The basics of the HC
algorithm are shown on Algorithm 2.1.

Another important property of the HC algorithms is that, under cer-
tain conditions, it asymptotically guarantees a resulting network that is
a minimal I-map of the underlying probability distribution in \( \mathcal{D} \) [Gámez
et al., 2012]. A given DAG \( \mathcal{G} \) is an I-map of a probability distribution \( p \)
if every independence codified in the graph is present in the distribution,
\((X, Y \mid Z) \Rightarrow I_p(X, Y \mid Z)\). We consider that the I-map is minimal if no arc
can be removed from \( \mathcal{G} \) without violating this condition. This is a desirable
property in any structural learning algorithm.

2.3.2 Parametric Learning

We can measure how good a particular model \( H \) is for a given dataset \( \mathcal{D} \) by
computing the likelihood, \( L(H \mid \mathcal{D}) = P(H \mid \mathcal{D}) \). Assuming independence
among the examples given the model and taking logarithms we obtain the
log-likelihood:

\[
LL(H \mid \mathcal{D}) = \sum_{d=1}^{m} \log P(x^{(d)} \mid H) \tag{2.5}
\]

Maximum Likelihood Estimation (MLE) is the most prominent method
employed to compute the parameters \( \theta \) for a given a Bayesian network \( H \).
We will estimate \( \hat{\theta} \) as the configuration that maximizes the log-likelihood:

\[
\hat{\theta} = \arg \max_{\theta} LL(H_\theta \mid \mathcal{D}) \tag{2.6}
\]

Given the independences encoded by the model we can factorize the pre-
vious expression. Also, we can assume independence among the local param-
eters \( \theta_j \) [Buntine, 1996] and decompose the problem into smaller pieces:

\[
LL(H_\theta \mid \mathcal{D}) = \sum_{i=1}^{n} LL(\theta_i \mid \mathcal{D}) \tag{2.7}
\]


**Algorithm 2.1:** Structural learning of BNs by using a Hill Climbing algorithm [Gámez et al., 2012].

**Input:** $D$: A dataset defined over variables $X = \{X_1, \ldots, X_n\}$

**Input:** $G_0$: A DAG defined over $X$ used as the starting point for the search

**Output:** A DAG $G$

1. $G \leftarrow G_0$
2. $f_g \leftarrow f(G : D)$; // $f$: decomposable scoring metric
3. improvement $\leftarrow$ true;
4. while improvement do
5.   improvement $\leftarrow$ false;
6.   // neighbours generated by addition
7.   For each node $X_i$ and each node $X_j \notin \Pi_{g_i}X_i$, such that $X_j \to X_i$ does not introduce a directed cycle in $G$, compute the difference $\text{diff} = f(G + \{X_j \to X_i\} : D) - f_g$. Store the change which maximizes $\text{diff}$ in $\langle \text{change}_a, \text{diff}_a \rangle$;
8.   // neighbours generated by deletion
9.   For each node $X_i$ and each node $X_j \in \Pi_{g_i}X_i$, compute the difference $\text{diff} = f(G - \{X_j \to X_i\} : D) - f_g$. Store the change which maximizes $\text{diff}$ in $\langle \text{change}_d, \text{diff}_d \rangle$;
10.  // neighbours generated by reversal
11.  For each node $X_i$ and each node $X_j \in \Pi_{g_i}X_i$, such that reversing $X_j \to X_i$ does not introduce a directed cycle in $G$, compute the difference $\text{diff} = d_1 + d_2$ where $d_1$ corresponds to $f(G - \{X_j \to X_i\} : D) - f_g$ and $d_2$ corresponds to $f(G + \{X_j \to X_i\} : D) - f_g$. Store the change which maximizes $\text{diff}$ in $\langle \text{change}_r, \text{diff}_r \rangle$;
12.  // Checking if improvement
13.  Let $d^* = \max_{k=a,d,r} \text{diff}_k$ and $\text{move}^*$ its corresponding change;
14.  if $d^* > 0$ then
15.     improvement $\leftarrow$ true;
16.     $G \leftarrow$ apply $\text{move}^*$ over $G$;
17.     $f_g \leftarrow f_g + d^*$;
18.  end
19. return $G$;
2. BACKGROUND

\[ LL(\theta_i \mid \mathcal{D}) = \sum_{d=1}^{m} \log P(x_i^{(d)} \mid \Pi_i^{(d)}) \]  

(2.8)

We can maximize the previous equation by estimating the empirical probabilities from the data by simply computing the relative frequencies from the data:

\[ \hat{\theta}_{ijk} = \frac{N_{ijk}}{N_{ij}} \]  

(2.9)

Estimating the parameters from observed frequencies only can give us unreliable results, especially if there are few or zero occurrences for a particular combination of variables \( N_{ijk} \). Bayesian smoothing techniques such as Laplace correction guarantees that the estimated distribution is positive by assuming a prior distribution with uniform hyperparameters, which translates into simply adding an extra outcome for each value of the database:

\[ \hat{\theta}_{ijk} = \frac{N_{ijk} + 1}{N_{ij} + r_i} \]  

(2.10)

2.4 Markov Random Fields

Directed networks are very convenient to express dependencies such as causality, however, directionality can often force unnatural relationships among the variables for certain domains. Consider an image in which each pixel is represented by a random variable and we consider each pixel to be dependent on the value of its neighbouring pixels. Imposing a topological ordering among the different pixel seems not optimal and may introduce artificial dependencies in the model.

An alternative is to use an undirected graph, this kind of model is usually known as Markov Random Field (MRF) or Markov Network [Murphy, 2012]. This symmetric representation is often use to model domains such as spatial or relational data. The dependency model of these networks is defined by a simple graph separation rule called the global Markov Property: Given a set of nodes \( X, Y, Z \in \mathcal{G} \) we say that \( X \) is independent from \( BY \) iff \( Z \) separates
Encoding a joint probability distribution in a MRF is more complex than in a directed model, as the Markov chain rule is not applicable given the absence of a topological ordering. Instead of associating a particular conditional probability table for each nodes we associate potential functions or factors to each edge of the graph. For discrete variables, potential functions can be represented as tables of non-negative numbers, just as with CPTs. Please note that this representation shall no be confused with probability distributions and rather as a compatibility function between the different variables.

2.5 Supervised Classification with Bayesian Networks

Supervised classification is one of the most relevant problems in data mining, constituting a rich and prolific research field. This task consist on learning a model from labelled data with the purpose of assign or recover such information in new unobserved and unlabeled examples.

Formally the problem can be stated as learning a model from a dataset \(D = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}\) of labelled examples where \(y \in \Omega_Y\) is a particular label among the available ones defined by variable \(Y\). In this setting we differentiate between the set of predictor features or attributes \(X = \{X_1, \ldots, X_n\}\) and the label, also called class, variable \(Y\).

2.5.1 Bayesian Network Classifiers

From a probabilistic approach, given an example \(x\) we can recover the class by estimating \(p(y | x)\), and returning the value \(y \in \Omega_Y\), which maximizes the posterior probability. Starting from the Bayes rule:

\[
p(y | x) = \frac{p(x | y)p(y)}{p(x)} \quad (2.11)
\]

the denominator becomes constant when comparing among different values of
2. BACKGROUND

\[ \hat{y} = \arg \max_{y \in \Omega} p(y)p(x \mid y) \] (2.12)

When modelling this problem by using a Bayesian network we can increase the accuracy of posterior estimates by ensuring that all attributes in the class Markov blanket are connected directly to the class node or its children. This produces a particular model that focuses on supervised classification, often called a Bayesian network classifier (BNC) [Bielza & Larrañaga, 2014].

Perhaps the most common BNC model is the popular Naive Bayes (NB) classifier [Domingos & Pazzani, 1997], in which all the predictive features in X are considered to be independent given the class Y. The performance of the NB classifier has proven to be competitive for many problems [Rish, 2001], and thanks to its efficient training is a suitable candidate for many practitioners. This independence assumption defines a fixed structure for the model, and thus it is not necessary to induce the graph from the training data, translating into high computational savings. The graphical representation of NB classifier is illustrated in Figure 2.1.

![Figure 2.1: Naive Bayes classifier (NB)](image)

To train NB classifier we only need to induce its parameters from the data. The estimation technique plays an important role and can significantly condition the overall quality of the resulting model [Madden, 2009]. Maximum likelihood estimation from frequency counts can yield zero estimates for unlikely events, which will harm the predictive power of the models. As a result,
smoothing techniques such as Laplace may be applied to overcome this limitation. Recent proposals of sophisticated estimation techniques [Petitjean et al., 2018] demonstrate that BNCs can be overly improved with more accurate parameters, especially in high variance domains. In our experiments we will always employ smoothed Laplacian estimation to learn the models.

Training a naive Bayes classifier has linear computational complexity, given the dimensions of the dataset, $O(nm)$ and linear spatial complexity, $O(nc\bar{v})$ which is the spatial complexity of the resulting model as well. In the previous formulas, $c = \Omega_Y$ and $\bar{v}$ represent the cardinality of the class variable and the average cardinality of the predictor attributes respectively. The resulting MAP hypothesis is very efficient, being linear to the number of attributes:

$$
\hat{y} = \arg\max_{y \in \Omega_Y} \left( p(y) \prod_{i=1}^{n} p(x_i \mid y) \right)
$$

(2.13)

While some of the independence assumptions in NB cause no harm [Rish, 2001], others come at the expense of discriminative power, resulting in a highly biased classifier. To overcome this problem the NB assumption can be relaxed by allowing some additional relationships in the DAG [Bielza & Larrañaga, 2014]. The resulting models are usually called augmented naive Bayes classifier and are often preferred over general BNs [Friedman et al., 1997] for the sake of simplicity and as better approximations for discriminative posteriors. Learning these models imply an additional computation effort in the learning process of the algorithm but usually paired with a substantial improvement on the classifier performance. We will describe in depth some of the most known techniques.

2.5.1.1 Tree-Augmented Naive Bayes

The Tree Augmented NB (TAN) model [Friedman et al., 1997] relaxes the conditional independence assumption in NB by allowing each predictive attribute to depend on an extra variable in addition to the class, see an example of such structure in Figure 2.2. To select this particular dependences, a struc-
2. BACKGROUND

A natural learning algorithm is proposed in which the well known Chow and Liu method [Chow & Liu, 1968] is used to build a maximum spanning tree by means of the Conditional Mutual Information (CMI):

\[
MI(X_i, X_j \mid Y) = \sum_{y \in \Omega_Y} \sum_{x_i \in \Omega_{X_i}} \sum_{x_j \in \Omega_{X_j}} p(x_i, x_j, y) \log \left( \frac{p(x_i, x_j \mid y)}{p(x_i \mid y)p(x_j \mid y)} \right)
\]

(2.14)

\[
\begin{array}{c}
\text{} \\
\text{} \\
Y \\
X_1 & X_2 & X_3 & X_4 \\
\text{} \\
\text{} \\
\end{array}
\]

Figure 2.2: Tree-augmented naive Bayes classifier (TAN or kDB with \( k = 1 \))

Once the tree is obtained a DAG is built by arbitrarily selecting a root node and orienting the edges in topological ordering; finally the class variable is added as a common parent to all nodes. This algorithm requires a computational complexity of \( O(mn^2) \) to compute the CMI for each pair of attributes, for which it builds a three-dimensional table with spatial complexity of \( O(cn^2) \). The computational complexity for building the maximum spanning tree is \( O(n^2 \log n) \). Parametric learning of the model can reuse the counts computed for the CMI and calculate the required probability tables with \( O(nv^2) \) complexity; the resulting TAN classifier requires an storage with spatial complexity \( O(cn^2) \).

2.5.1.2 \( k \)-Dependence Estimators

The \( k \)-Dependent Estimator [Sahami, 1996] can be seen as a generalization of the previous idea, in which a number of \( k \) additional parents are allowed in the model for each predictive variable, e.g. a \( k \)DB model with \( k = 1 \) could be identical to the one pictured in Figure 2.2. This algorithm can explore a wider spectrum of distributions, starting from NB if \( k = 0 \) towards a more
densely connected Bayesian network as $k$ increases. The structure of the classifier is learned using an algorithm that can be decomposed into three different stages:

- A ranking $\sigma$ is established between the predictive attributes by means of their mutual information with the class variable, $MI(X, C)$ (eq. 2.15).

$$MI(X_i \mid Y) = \sum_{y \in Y} \sum_{x_i \in \Omega_{X_i}} p(x_i, y) \log \left( \frac{p(x_i, y)}{p(x_i)p(y)} \right)$$ (2.15)

- For each attribute $X_i$ being $X_{\sigma(i)}$ its position on the previous ranking $\sigma$, we compute the conditional mutual information $MI(\cdot, X_i \mid Y)$ (eq. 2.14) given the class for the subset of attributes preceding $X_i$ in $\sigma$: $\{X_1, \ldots, X_{i-1}\}$. Then the best $k$ attributes with the highest dependence are set as the parents of $X_i$.

- Finally the class variable $Y$ is added as a parent for all the predictive attributes.

Learning the network structure of a $k$BD classifier has a computational complexity of $O(n^2m)$ and a spatial complexity of $O(cvn^2)$ as it also builds a three-dimensional table. Calculating the CPTs for that particular network takes $O(n(m + \bar{v}^k))$ and needs a spatial capacity of $O(kn\bar{v}^k)$. Notice that this algorithm requires passing twice through the data if $k \geq 2$.

Selecting an appropriate value for $k$ can be challenging as it depends on the properties of the problem at hand. At higher values of $k$, the algorithm yields more complex models, decreasing the bias but usually increasing the variance because of overfitting. This can be a disadvantage in small data, however, some studies [Martínez et al., 2016] have demonstrated that variance can be overly reduced in the presence of large datasets along with overfitting. Approaches such as Selective $k$DB (S$k$DB) [Martínez et al., 2016] extends $k$DB by carrying out a cross validated selection between attribute subsets and values of $k$ in a single additional pass through the dataset.
2. BACKGROUND

2.5.1.3 Averaged $k$-Dependence Estimators

The A1DE classifier [Webb et al., 2005] follows a different strategy, in which structural learning is avoided by learning a restricted, full family of models. Such models are called super-parent one dependence estimators (SP1DE) and present a particular augmented NB graph in which a single attribute $X_{sp}$, designated as super-parent, is the unique shared parent for the remaining variables in addition to $Y$. A1DE can be considered as an ensemble [Dietterich, 2000] of every possible combination of SP1DE models for a given problem with $X = \{X_1, \ldots, X_n\}$ attributes for a total of $n$ models, Figure 2.3. The classifiers are combined by computing the average of their probabilities, resulting in the following MAP hypothesis:

$$
\hat{y} = \arg \max_{y \in \Omega_Y} \left( \sum_{j=1}^{n} p(y, x_j) \prod_{i=1, i \neq j}^{n} p(x_i \mid y, x_j) \right) 
$$  (2.16)

![Figure 2.3: Individual models of an A1DE classifier for $n = 4$](image)

The same idea has been extended for arbitrary sizes of super-parent compound variables as the A$k$DE classifier [Webb et al., 2011]. For larger values of $k$ the super-parent attributes are constructed as the Cartesian product of $k$ attributes for a total of $\binom{n}{k}$ SP$k$DE, increasing the predictive power of the classifier at the cost of a higher training time and space complexity. The
A1DE classifier has proven to be a good trade-off between efficiency and accuracy, outperforming the other BNCs approaches in many domains. Learning an AkDE classifier requires the computation of a $k + 2$ dimensional table to estimate the required parameters, with computational complexity of $O(mn^k)$ and spatial complexity of $O\left(\binom{n}{k}c^{(k+1)}\right)$.

As in $k$DB, the AkDE framework has the ability to adjust the bias-variance trade-off by varying the $k$ parameter. This allows us to represent from high bias/low variance classifiers, such as NB (A0DE), to lower bias but higher variance classifiers as $k$ increases [Webb et al., 2011]. This increases model space and thus the dimensionality of the parameters. Specifically, a total of $K = \binom{n}{k}$ models will be learnt, requiring the induction of $K$ $k$-dimensional joint frequency tables. The complexity of the algorithm is polynomial given the number of attributes and increases in order with the hyperparameter $k$. This is an intractable problem in the case of high dimensional data, not just due to extensive computational requirements but because of the spatial complexity of the model, quickly scaling to Gigabytes even for moderate datasets and values of $k$: (2, 3, ...).

Model selection becomes mandatory to make effective use of AkDE in practice. For this reason several approaches have been proposed in order to reduce the computational requirements of the algorithm. We can divide them into pure wrapper strategies [Zheng & Webb, 2007], based mostly on greedy search; hybrid filter-wrapper approaches [Chen et al., 2014b, 2017b], based on information theory metrics rankings and greedy wrapper model selection; and pure filter ranking algorithms [Chen et al., 2017a], based uniquely on information theory metrics and computational resources bounds. The latter approaches have been extensively tested on its application to high dimensional and large data domains.

These studies have reported that model selection not only reduces the spatial complexity but also increases the performance of the resulting ensemble. In practice, the discriminative power of each individual model can be harmed by the inherent assumptions in the naive Bayes condition and the inter-dependencies between the attributes and the super-parent sets, since a fixed BN structure will not likely be an optimal representation of the under-
lying distribution in the data.

Reducing the model space of a given AkDE classifier is by itself a problem in the exponential search space of each possible combination of $k$ attributes. The intuition behind the mentioned extensions is grounded on the assumption that the conditional mutual information of the super parent set of attributes given the class acts as a good approximation for the performance indicator for the resulting sub model. This heuristic is used as a ranking metric among the attributes and then a particular criterion is used to select and optimal cut point for the ranking:

**Attribute Selective A1DE (ASA1DE)** [Chen et al., 2014b] is a particular case in which a constrained model space is built by greedily adding a new parent or children attribute following the ordering from mutual information ranking, this creates a nested collection of possible ensemble models. These models are then evaluated by using an efficient leave one out cross validation schema that relies on the probabilistic model of A1DE to reuse the majority of the computed statistics. Finally, the best combination is selected according to the maximum performance. This approach implies performing an additional pass through the dataset, increasing the computational requirements for the training phase and thus being too expensive for larger values of $k$.

**Sample Attribute Selective A$k$DE (SASA$k$DE)** [Chen et al., 2017b] follows the previous strategy but uses only a sample of the training dataset to evaluate the nested model space, alleviating the computational requirements of the second pass through the data. The resulting model is not biased towards such sample as the parameters are learnt from the full dataset in the first pass through the data. This model still requires to learn a large number of parameters as at least a full AkDE model is induced during the process.

**Selective A$n$DE (SA$n$DE)** [Chen et al., 2017a] is a pure filter approach that bounds the number of models in the ensemble by directly setting a cut point $s$ in the attribute ranking. The selected models are those formed by the $k$–size combination of the available parents. In the experiments, the authors select the value of $s$ by calculating memory restrictions according to the size of the resulting full model. In this work, an improvement over
the ranking metric is introduced, which measures the conditional mutual information between the a candidate attribute and the class given the set of preceding attributes in the ranking. This approach is the most scalable one at this moment, being able to learn subsets of an A3DE ensemble.

2.6 Multi-dimensional Classification

While the description given for supervised classification can be applied to the majority of existing problems, there are specific cases where the problem is altered to reflect some real world scenarios that cannot be represented with a single fully observed class variable.

One of such problems is known as multi-label classification, in which several class variables are simultaneously considered and the task consists of assigning a configuration of values to all the class variables. In the multi-label setting, classes (or labels) are binary. On the contrary, multi-dimensional classification is a generalization of multi-label classification that allows class variables to have more than two values [Read et al., 2014]. There is a wide range of explored applications for multi-dimensional classification [Zhang & Zhou, 2014]: bio-informatics, document categorization, semantic scene classification, multi-fault diagnosis, etc.

One approach to solve a multi-dimensional problem is to first transform the problem into a set of single-class classification problems, and then combine the outputs to obtain a joint configuration of the class values. Transformation-based methods range from binary relevance, where no interaction among the class variables is modelled, to brute-force label power set methods, where all the class variables are aggregated into a single compound class. In-between these two extremes, we find new and/or adapted algorithms that have been developed to deal with the multi-dimensional problem, managing the interactions between the variables in a natural way. From this second family, probabilistic methods and, in particular, those based on Bayesian networks have demonstrated a convincing performance [Bielza et al., 2011].

Formally we can describe the problem as having a dataset consisting of a collection of instances \( \mathcal{D} = \{ (x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)}) \} \), where the first part
of an instance, \( \mathbf{x}^{(i)} = (x_1^{(i)}, \ldots, x_n^{(i)}) \), is a configuration of values defined over a set \( \mathbf{X} = \{X_1, \ldots, X_n\} \) of predictive attributes, while the second part, \( \mathbf{y}^{(i)} = (y_1^{(i)}, \ldots, y_l^{(i)}) \), is a configuration of values defined over a set \( \mathbf{Y} = \{Y_1, \ldots, Y_l\} \) of classes.

Our goal is to induce a multi-dimensional classifier \( h \) that maps configurations of the predictive variables to configurations of the class variables:

\[
\begin{align*}
\mathbf{h} : \bigtimes_{i=1}^{n} \Omega_{X_i} & \longrightarrow \bigtimes_{j=1}^{l} \Omega_{Y_j}, \\
(x_1, x_2, \ldots, x_n) & \mapsto (y_1, y_2, \ldots, y_l),
\end{align*}
\]  

(2.17)

where \( \bigtimes \) denotes the Cartesian product.

### 2.6.1 Evaluation

We can use different evaluation metrics to evaluate a multi-dimensional classifier (see e.g. [Bielza et al., 2011, Sec. 5]). The two most popular ones are specializations of the classical classification accuracy, measured as the proportion of correctly classified instances. Given a multi-dimensional dataset \( \mathbf{D} \) consisting of \( m \) instances, together with the predictions obtained by a multi-dimensional classifier \( h(\mathbf{x}) = (\hat{\mathbf{y}}) \), we can define:

- **Exact match** (or **global accuracy**), measures the average accuracy for each individual label independently among all examples.

\[
\text{acc}(\mathbf{D}, f) = \frac{1}{m} \sum_{i=1}^{l} \delta(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}),
\]

(2.18)

where \( \delta \) is Kronecker’s delta function.

- **Hamming** (or **mean**) **accuracy**, measures the accuracy for estimating
the exact configuration of all labels correctly for each example.

\[
H_{\text{acc}}(D, h) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{l} \sum_{j=1}^{l} \delta(y_j^{(i)}, \hat{y}_j^{(i)}).
\] (2.19)

In both cases, the higher the value the better. Obviously, acc is a harder scoring criterion than \(H_{\text{acc}}\).

From a probabilistic perspective, we can use different inference tasks [Jensen & Nielsen, 2007] to maximize each of the two scores. Concretely, computing the most probable explanation (MPE) for the class variables will maximize global accuracy, while maximizing the Hamming accuracy requires the computation of the most probable marginal assignment for each of the individual class variables.

We can find several approaches for dealing with multi-dimensional classification problems in the literature. We will review some of them, detailing a bit more those approaches used for comparison in this dissertation. A more general and extensive overview and comparison of existing algorithms can be found in Zhang & Zhou [2014] and Madjarov et al. [2012].

### 2.6.2 Transformation methods

According to Tsoumakas & Katakis [2007] this family of methods simplify the multi-dimensional classification problem into one or several single-class classification problems. Perhaps the two most adopted approaches coming from the multi-label domain are the classifiers based on label power-sets (LP) and binary relevance (BR). In their simplest form, LP-based classifiers construct a new (compound) single-class variable having as possible values all the different configurations of the class values (labels) included in the training set. This method implicitly considers the dependencies between classes, but its obvious main drawback is that it is computationally tractable only for a relatively small number of class variables.

On the other hand, BR-based methods learn a single-class classifier (or base classifier) for each class variable, \(h_i : \times_{j=1}^{n} \Omega_{X_j} \rightarrow \Omega_{Y_i}\). A solution
to the multi-dimensional problem is then found by combining the single-class outputs from the base classifiers. This method does not consider any dependencies between class variables, and it usually underperforms when considering exact accuracy. However it can still provide good predictions according to Hamming accuracy for certain domains.

More sophisticated approaches can be found in-between BR and brute-force LP-based methods. One example is RAkEL [Tsoumakas et al., 2011], which is based on training several single-class classifiers each using as class a compound variable constructed as the Cartesian product of \( k \) class variables. The number of single-class variable models is usually linear in the number of class variables, and random selection is used to choose the \( k \) variables that will form the compound variable. Later, voting is used to aggregate the \( k \)-tuples predicted, obtaining a joint configuration over all the class variables.

### 2.6.3 Chain classifiers

Chain classifiers (CC) [Read et al., 2011] are an alternative to BR that incorporate dependencies between classes while maintaining the computational efficiency of BR. In CC, an ordering \( \sigma \) is defined over the class variables. Let \( Y_{\sigma(i)} \) be the \( i \)-th class variable according to ordering \( \sigma \). As in BR, \( l \) single-class classifiers are induced, however, when learning the single-class classifier having \( Y_{\sigma(i)} \) as class, the variables \( Y_{\sigma(1)}, \ldots, Y_{\sigma(i-1)} \) are also included as predictive attributes: \( h_i : \times_{j=1}^n \Omega_{x_j} \times \times_{k=1}^{i-1} \Omega_{Y_{\sigma(k)}} \rightarrow \Omega_{Y_{\sigma(i)}} \). Therefore, the class variable in position \( i \) of \( \sigma \), depends on the class variables appearing earlier in the ordering.

As a consequence, inference over the single-class classifiers must be done sequentially by following the ordering imposed by \( \sigma \) and by using the predicted values of the previous class variables as input when performing inference. This ordering restriction is lifted in Alvares-Cherman et al. [2012]; Monta˜nes et al. [2014], where each single-class classifier uses all the other class variables as predictive attributes; during inference, the values of these additional input features are found using a separate set of base classifiers. A
probabilistic approach to chain classifiers was proposed in Dembczyński et al. [2010], which generalizes the original chain classifier by letting the single-class classifiers $h_i$ define a probability distribution over $Y_{\sigma(i)}$. In combination, the single-class classifiers define a joint distribution over all class variables, for which the original chain classifier can be seen as a deterministic approximation. The cost of this generalization is a corresponding increase in model complexity, which requires the use of approximate inference algorithms for domains with even a moderate number of class variables.

2.6.4 Adaptation methods

These are methods directly modify/adapt existing single-class classification algorithms to accommodate multiple classes, e.g. based on decision trees, nearest neighbours, support vector machines, etc. See Tsoumakas & Katakis [2007] for an overview.

2.6.5 Multi-dimensional Bayesian Networks classifiers

Multi-dimensional Bayesian networks classifiers (MBCs) [Bielza et al., 2011; de Waal & van der Gaag, 2007; van der Gaag & de Waal, 2006] model the problem under the Bayesian network formalism. As in the case of single-class classification where we use BNCs, the models employed in multi-dimensional classification are specialized to discriminate the label configuration, meaning that the class nodes will have a special position in the topology of the graph. We usually split the structure into three subgraphs:

1. The **class subgraph**, which codifies dependencies between the class nodes.

2. The **feature subgraph** which codifies dependencies between the features nodes.

3. The **bridge subgraph** which codifies dependencies from the class nodes to the feature nodes.
A range of models can be found, usually defined by the complexity allowed for each subgraph: trees and polytrees [de Waal & van der Gaag, 2007; Rodríguez & Lozano, 2008; van der Gaag & de Waal, 2006], $k$-dependence limited models [Rodríguez & Lozano, 2008] or general BN structures [Bielza et al., 2011] among others. Another taxonomy of methods can be established with respect to the search strategy used to guide the learning process, filter and wrapper approaches have been explored in Bielza et al. [2011], while skeleton-based ones are proposed in Borchani et al. [2012, 2013] based on Markov blankets and in Zaragoza et al. [2011a] using mutual information.

In Corani et al. [2014] a slightly different approach which is of interest for this dissertation is introduced. This approach concentrates the search effort in the class and bridge subgraphs, while assuming an empty feature subgraph, that is, features are conditionally independent given the classes. For the class subgraph, restricted topology structures (naive Bayes and forest augmented networks) are considered, while for the bridge subgraph an exact score+search method is used considering all possible parents sets for each feature. The accuracy of the obtained classifiers is slightly worse than the ensemble proposed by the same authors [Antonucci et al., 2013], but inference is much faster.

The method proposed in Antonucci et al. [2013] assumes that the input features are conditionally independent given the class labels, hence focus is on learning structures over the class labels. In contrast, Hong et al. [2014] considers a mixture of tree structures over the class labels, but each of the mixture trees being conditioned on the input features.

### 2.6.6 Ensembles

As in single-class classification, ensembles of multi-dimensional classifiers have shown potential to improve performance compared to single-classifiers. This is, for example, the case of an ensemble of CC, where each member of the ensemble uses a different (usually random) ordering [Read et al., 2011]. In the particular case of using ensembles of MBCs, recent studies [Antonucci et al., 2013; Sucar et al., 2014; Zaragoza et al., 2011a] explore the idea of using
as many members in the ensemble as there are class variables. In Sucar et al. [2014]; Zaragoza et al. [2011b] an undirected tree structure is first learned for the class variables, and then each class variable is set as root and the resulting topological ordering is used as a CC in the ensemble. The resulting configuration of class values is obtained by voting. In Antonucci et al. [2013] no structural learning is done over the classes, and instead a naïve Bayes structure among the class variables is used, but with a different root node for each member of the ensemble. In contrast to previous approaches, the resulting configuration is obtained by probabilistic inference.

Finally, we would like to mention two recent works which show the degree of maturity research on MBCs has achieved. In Varando et al. [2014] a theoretical study is carried out to analyse the expressive power of binary relevance and chain classifiers, proving that chain classifiers provide more expressive models than the binary relevance method when the same type of BAN (Bayesian network augmented naïve Bayes) classifier is used as the base classifier. On the other hand, Ramírez-Corona et al. [2014] addressed the problem of Hierarchical Multi-label Classification (HMC), which addresses domains for which there exists a hierarchical structure among the classes.

2.7 Notation Summary

The following table is a summary of the most relevant notation terms that we will use during this dissertation.
2. BACKGROUND

<table>
<thead>
<tr>
<th>$X$</th>
<th>Set of random variables in the problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_i$</td>
<td>A particular random variable</td>
</tr>
<tr>
<td>$x_i$</td>
<td>A state for the variable $X_i$</td>
</tr>
<tr>
<td>$\Omega_{X_i}$</td>
<td>Domain of the variable $X_i$</td>
</tr>
<tr>
<td>$r_i =</td>
<td>\Omega_{X_i}</td>
</tr>
<tr>
<td>$x_{[i,j,\ldots,q]}$</td>
<td>A subset of attributes from $x$ with indexes</td>
</tr>
<tr>
<td>$\mathcal{D}_k$</td>
<td>A subset of the training data</td>
</tr>
<tr>
<td>$x = (x_1, \ldots, x_n)$</td>
<td>An unlabelled example</td>
</tr>
<tr>
<td>$x = (x_1, \ldots, x_n, y)$</td>
<td>A labelled example</td>
</tr>
<tr>
<td>$x = (x_1, \ldots, x_n, y_1, \ldots, y_l)$</td>
<td>A multi-dimensional labelled example</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of predictor attributes</td>
</tr>
<tr>
<td>$m$</td>
<td>Number of examples in the training data</td>
</tr>
<tr>
<td>$Y$</td>
<td>Class variable in a supervised classification problem</td>
</tr>
<tr>
<td>$\Omega_Y = {y_1, \ldots, y_c}$</td>
<td>Domain of the class variable</td>
</tr>
<tr>
<td>$c$</td>
<td>Number of classes in a supervised classification problem</td>
</tr>
<tr>
<td>$l$</td>
<td>Number of class variables in a multi-dimensional problem</td>
</tr>
<tr>
<td>$\bar{v}$</td>
<td>Average number of values per attribute</td>
</tr>
<tr>
<td>$H = {G, \theta}$</td>
<td>A BN configured by a DAG $\mathcal{G}$ and a set of parameters $\theta$</td>
</tr>
<tr>
<td>$\Pi_{\mathcal{G},X_i}$</td>
<td>Set of parents for the variable $X_i$ encoded by $\mathcal{G}$</td>
</tr>
<tr>
<td>$q_i = \prod_{X_j \in \Pi_{X_i}} r_j$</td>
<td>Number of configurations of $X_i$ over its parents in a BN</td>
</tr>
<tr>
<td>$N_{ijk}$</td>
<td>Number of instances with $X_i$ in its $k$th value and $\Pi_{X_i}$ in its $j$th configuration</td>
</tr>
<tr>
<td>$N_{ij}$</td>
<td>Number of instances with the parent set of $X_i$, $\Pi_{X_i}$ in its $j$th configuration</td>
</tr>
</tbody>
</table>

Table 2.1: Notation summary
Chapter 3
Computing Paradigms for Large-scale Problems

3.1 Introduction

Supercomputers and high performance computing (HPC) has been a core technology in computational sciences as well as a fundamental tool for many other disciplines such as quantum mechanics, weather forecasting, climate research, oil and gas exploration, molecular modelling, genomics and any other example that requires high computing power to run simulations or other intensive applications.

Generally speaking, supercomputers [Buyya et al., 1999] are systems with high level performance compared to a general-purpose computer. While we measure the power of a commodity processor in millions of instructions per second (MIPS), the most powerful supercomputers are currently measured in thousands of trillions of floating point operations per second (TLOPS) \(^1\). Current research trends are exploring the paths towards exascale machines [Bergman et al., 2008].

Such a vast amount of computing power requires complex programming paradigms in order to manage multiple processors in parallel. These frame-

\(^1\)The Top500 organisation maintains an update list of the most powerful supercomputers in the world, reporting Summit at Oak Ridge National Laboratory as the most advanced with 122.3 petaflops at the time of this writing.
works have many challenges as usually algorithms cannot be directly translated into a parallel programs. In addition, many of these programs require sharing input data or intermediate computations that require specific synchronization techniques, as data is stored in physical devices that might not be directly connected to the processing units. Frameworks such as OpenMP [Dagum & Menon, 1998] or MPI [Gropp et al., 1999] have been the usual choices for the most common shared memory or message passing architectures.

Although HPC is a prolific field of research, commercial computing systems also known as commodity hardware has been one of the most relevant economic and research areas for technological companies over the last decades. Moore’s law [Moore, 1998] predicted that the capabilities of microprocessors would experiment exponential grow starting in the 1970s’. Integration technology became more and more sophisticated and commodity hardware grew proportionally to the point that, in the span of 30 years the computer inside a common mobile device became more powerful that some of the supercomputers being used back then. However, the exponential grow of integration technology has slowed down [Bergman et al., 2008] since the early 2000s’ and innovation has shifted towards multi-core processors and other architectures such as GPUs which require even more specialization of the underlying computing framework to exploit on-chip parallelism.

Influenced by the advent of advanced computers, other technologies such as the Internet bloomed and created new challenges and requirements for computing technologies. While many of the traditional HPC tasks were focused on expensive computations such as complex simulations, data intensive applications were starting to be considered very valuable, such as Google’s PageRank algorithm [Page et al., 1999]. The complexity of the algorithm is driven by the amount of data we are willing to process, the target at the time was Internet which had a massive scale that no single machine was able to process.

Even though PageRank was designed with parallelism in mind, there were other scalability costs despite computing speed that were crucial in the design of their scalable computing architecture such as data durability.
and component failure. Google File System (GFS) [Ghemawat et al., 2003] is a modern approach to scalable data storage and processing, aimed at supporting data intensive tasks. This is one of the main focuses on computing with large-scale data that drifts from the usual HPC perspective of simply adding a massive amount of computing power.

3.2 The MapReduce Paradigm

The MapReduce framework [Dean & Ghemawat, 2004] builds on top of GFS to provide the first computing paradigm oriented to large-scale data processing. It is a high level paradigm aimed at simplify the most complex parts of dealing with the architecture of clusters and provides the programmer an interface consisting only on two abstractions, a map and a reduce function. Using MapReduce we can abstract from the parallelization and distribution of the data and the management of fault tolerance on the underlying hardware.

A master/worker architecture is proposed in which the former orchestrates a series of small tasks that are distributed and processed by the worker nodes. The original data is split into homogeneous partitions or chunks and a single map task is spawned for each one. The map stage transforms each partition of the data independently and returns an intermediate result that is shuffled along the cluster in preparation for the reduce stage. Ideally, transformations may have aggregated and filtered the data locally and now is much smaller. In the last stage the data from all map tasks is aggregated using the reduce function and the final value is returned.

Although this definition might seem simple, the framework is very expressive and a wide range of applications can be implemented under these restrictions. A MapReduce job can exploit data locality in a distributed file system such as GFS with high resiliency, as it can repeat any failed tasks without recomputing or synchronizing any other split or data.

A common representation for the intermediate data in a MapReduce job is defined by tuples of the form \((key, value)\). The initial input for each map task is usually given as an input partition of the original data, associated to
an arbitrary key. Each map task processes its chunk of data by applying a defined function to it, which generates a number of new intermediate \( \langle \text{key, value} \rangle \) output pairs. The system collects, shuffles and sorts this intermediate output pairs by their corresponding keys, and sends the matching pairs sharing the same key as the input data of a number reduce tasks. This input consists on a pair formed by the common key and a list containing the associated values. Then, the reduce step is performed, in which all the sorted pairs are processed by applying a function to combine the grouped values; as a result, a final output is generated, usually consisting on new formatted \( \langle \text{key, value} \rangle \) pair. Figure 3.1 shows a detailed flow of a MapReduce procedure.

A very didactic example is the word count algorithm, in which we can obtain the number of occurrences of all unique words in a raw text file. Given \( F \) as an arbitrary text file, we start by splitting it in into \( k \) independent partitions \( F_1 \) to \( F_k \). Then each partition is the input of a map function which generates a key-value pair \((w_i, 1)\), where \( w_i \) is a particular word in the text for each \( w \in \text{tokenize}(F) \) and \( \text{tokenize} \) is a function that receives raw text and splits it into an array of words. For each map task we will obtain a collection of tuples that will be shuffled and sorted by their corresponding key, namely the word. The output of this shuffle stage would be a key-value pair \((w_i, c_i)\)
where $c_i$ is a numeric array containing the individual counts for each word. Lastly, the reduce stage will just compute the tuple $(w_i, \sum c_i)$ by adding the individual counts. At the end we will have one tuple for each unique word in the document representing the total occurrences for such word. A common improvement called combination may be included in the map stage by applying a local reduce to the output, prior to the shuffle procedure. In this particular problem combination will aggregate multiple occurrences of the same word in the same data chunk. The algorithm is illustrated in Figure 3.2.

### 3.3 The Big Data Ecosystem

MapReduce is often used as both the name of the paradigm and the original implementation by Google. However, a few years after the success of MapReduce a community effort created an open source implementation known as Apache Hadoop [White, 2012]. Hadoop evolved quickly and became the de facto standard for large-scale data computing, being adopted by almost any relevant company and startup. This can be considered as a paradigm shift in technology, as other components such as hardware started adapting to the
new requirements.

Big data [John Walker, 2014; Manyika et al., 2011; McAfee et al., 2012] is a term that is used by both industry and scientists to reference problems which require a massive amount of data and very scalable computational approaches. The increasing amount of digital data that is available enables new problems and create new challenges everyday, i.e. technologies such as Social Networks, the Internet of Things or the impact of globalization and telecommunications [Chen et al., 2014b].

A broad ecosystem of libraries and frameworks has been created around Apache Hadoop [White, 2012]. Among those we can highlight machine learning as one of the key interests for the community, with early efforts such as Apache Mahout [Owen et al., 2011]. Machine learning is a data demanding task and scalability is a key factor for many algorithms. Unfortunately, MapReduce is not a general framework and not every algorithm is suitable for adaptation, for this reason, particular techniques have obtained more popularity on this new paradigm.

### 3.3.1 In-memory Distributed Computing

As with microprocessors, data storage has evolved exponentially over the years, especially since the advent of big data technologies. Back in 1987, a 4GB/120Kg IBM HD averaged a cost of $129k while a 128GB/5g USB stick has a price of approximately $30 nowadays, resulting in a difference factor of a million. Main memory scalability has improved as well, computers had around 1MB of memory when relational databases were designed while nowadays we can find scientific computers in the scale of terabytes.

One of the pitfalls of Apache Hadoop is that intermediate computations must be spilled to disk, therefore creating a bottleneck as mechanical disk have less speed than processors and main memory. A great effort was made to implement in-memory capabilities into MapReduce in order to allow out-of-core computations and iterated algorithms.

The Apache Spark [Zaharia et al., 2010] framework can be seen as the next generation of distributed computing frameworks. Apache Spark introduces
an in-memory data structure called *Resilient Distributed Datasets* (RDD) [Zaharia et al., 2012] which allows the practitioner to define additional operations over the data without strictly sticking to map and reduce functions. This abstraction allows for a wider range of applications on massive datasets and more advanced hardware.

While Hadoop was a generic platform for computation, Spark provides several high level interfaces for data processing such as statistics, structured data and machine learning [Meng et al., 2016]. Spark is the most extended data processing framework and has been used extensively in the literature to push forward the research of scalable machine learning approaches for large-data domains.
3. COMPUTING PARADIGMS FOR LARGE-SCALE PROBLEMS
Chapter 4

Balancing Efficiency and Accuracy in Structural Learning

We have seen that heuristic search can be employed to success at structural learning despite dealing with an exponential search space. Hill Climbing is perhaps one of the most tested method in the literature, and from the experience we observe two different problems: local optimum solutions and large scale domains for which even this algorithm does not scale. A particular approach, called FastCHC (Fast Constrained Hill Climbing), has achieved a consistent improvement upon the efficiency of the original algorithm, obtaining models with slightly lower quality but maintaining their theoretical properties. In this chapter we propose three different modifications for the most scalable version of FastCHC to improve the quality of its output. We will explore a relaxed version of the imposed constraints to introduce diversification in the search process without harming the final outcome. The aim of these new approaches is to adjust the trade-off between efficiency and accuracy of the algorithm, maintaining its theoretical computational complexity. We perform an intensive experimental evaluation of the proposed modifications along with an comprehensive comparison between the original algorithms and the modifications covering several scenarios with quite large datasets.†

†This chapter is based on the following papers: Arias et al. [2013, 2015b]
4. EFFICIENCY TRADE-OFF IN STRUCTURAL LEARNING

4.1 Introduction

Over the years there has been continuous effort to learn the structure of Bayesian networks from increasingly larger datasets. High dimensional datasets are present in multiple domains, e.g. micro-array gene expression data ranging in the thousands of variables, these pose a considerable challenge in order to be modelled with a BN. We can find several scalable algorithms in the literature, a great deal of them specifically based on local-search approaches [Friedman et al., 1999; Tsamardinos et al., 2006]. These methods usually restrict the search space in different ways in order to perform a more efficient search process. The most common strategy is based on a two-step process which first detects constraints based on the relationships between the variables, and then perform an intensified and more efficient local search [Tsamardinos et al., 2006]. However, there are other approaches such as the CHC algorithm [Gámez et al., 2011], which is a one-step constrained algorithm that progressively restricts the search space while performing an iterated local search without needing a previous step.

This new algorithm provides a significant efficiency improvement when compared with most state-of-the-art algorithms and especially with the unconstrained Hill Climbing approach. However, restricting the search space normally implies a loss of quality in the models obtained. In this chapter, we propose and evaluate three modifications to the FastCHC algorithm in order to solve this accuracy loss. Our new approaches are aimed at improving the resulting networks without modifying the computational complexity of the algorithm, and in order to achieve this we relax the constraints imposed by the original algorithm, adding diversification to the search process and allowing the algorithm to visit additional solutions. The proposed algorithms are parametrized and allow us to adjust the trade-off between the efficiency of the algorithm and the quality of the models, trying to provide better solutions without excessively decreasing the efficiency advantage.

4.2 Constrained Hill Climbing Methods

The CHC algorithm [Gámez et al., 2011] performs a progressive restriction of the neighbourhood during a Hill Climbing search process. The algorithm keeps what the authors call, Forbidden Parents Sets (FP) for each node, so in the neighbourhood generation step for the node $X_i$, any node $X_j \in FP(X_i)$ is not
considered as a suitable parent of $X_i$ and the algorithm avoids its evaluation, saving a large number of computations.

In order to include a node in the $FP$ set the algorithm uses the value of the score metric as a sort of conditional independence test when evaluating local changes, so that when the difference in score between the current structure and the new one resulting from applying the considered operation, $(diff = f(X_i, \Pi_{\text{New}}(X_i)) - f(x_i, \Pi_{\text{Current}}(X_i)))$ does not reveal a gain in the structure, the $FP$ sets are updated consequently:

- adding $X_j \to X_i$. If $diff = f(X_i, \Pi(X_i) \cup \{X_j\}) - f(X_i, \Pi(X_i)) < 0$ then $\{X_j\}$ is added to $FP(X_i)$ and vice versa.
- deleting $X_j \to X_i$. If $diff = f(X_i, \Pi(X_i) \setminus \{X_j\}) - f(X_i, \Pi(X_i)) > 0$ then $\{X_j\}$ is added to $FP(X_i)$ and vice versa.
- reversal of $X_j \to X_i$. Decompose as deleting($X_j \to X_i$)+adding($X_i \to X_j$) and use the previous two rules to update the $FP$ sets.

This progressive neighbourhood restriction is based on the concept of the locally consistent scoring criterion [Chickering, 2002], which states that $I(X_i, X_j \mid \Pi(X_i))$ if and only if $f(X_i, \Pi(X_i) \cup \{X_j\}) - f(X_i, \Pi(X_i)) < 0$. The usual scores, such as BDe, K2, MDL or BIC, are locally consistent [Chickering, 2002]. Due to the local consistency of the usual metrics, we can (asymptotically) assume that the differences computed from the metric $f$ can be used as conditional independence tests over the dataset $D$.

From the property in the previous paragraph, we can note that when the Hill Climbing algorithm considers an addition operation, $X_j \to X_i$, and a negative difference ($diff < 0$) is obtained with respect to the previous step, then (asymptotically) we can assume that $I(X_i, X_j \mid \Pi(X_i))$, and, because of the symmetry of conditional independence, the undirected link $X_i - X_j$ can be restricted, so we no longer have to test the addition of $X_j$ as parent of $X_i$, and $X_i$ as parent of $X_j$ as the subset of forbidden parents $FP$ is consequently updated for each variable. When Hill Climbing considers a deletion operation, and a positive difference is obtained ($diff > 0$), then again (asymptotically) we can assume that $I(X_i, X_j \mid \Pi(X_i) \setminus \{X_j\})$ and update the forbidden parent set of $X_i$ and $X_j$ accordingly. An inversion operation can be seen as deleting
4. EFFICIENCY TRADE-OFF IN STRUCTURAL LEARNING

\(X_j \rightarrow X_i \) + adding \(X_i \rightarrow X_j\). Algorithm 4.1 outlines the CHC algorithm, adding the described FP sets.

This algorithm is more efficient than unconstrained Hill Climbing, however, some of its theoretical properties do not hold. More concretely, we cannot guarantee that the procedure will return a minimal I-map. For this reason, the CHC* algorithm is proposed, which runs both CHC followed by HC using the intermediate solution as an starting point. This way the minimal I-map is guaranteed by the HC stage.

Performing an unconstrained stage reduces the performance of the whole CHC* algorithm, as an alternative we can iterate CHC multiple times [Gámez et al., 2011], chaining the output of an iteration as the initial solution for the next one. In addition we must start each iteration with an empty set of FP, this way the algorithm ends when it is unable to perform any change at the beginning of an iteration and, because no constraints are being used, it has the same stopping criteria as an unconstrained Hill Climbing algorithm, thus returning a minimal I-map.

In Gámez et al. [2012], the FastCHC algorithm is introduced, which returns a minimal I-map in a single iteration. The key idea behind this approach is to correct wrongly discovered relationships in the graph during the search process. For that, it releases some constraints every time it performs an addition operation on the network, allowing the solution to become an I-map in further steps. In particular, after the algorithm adds an arc \(X_i \rightarrow X_j\), all the nodes from the neighbourhoods of the head and tail nodes \(adj(X_i)\) and \(adj(X_j)\) are removed from their respective FP sets: \(FP(X_i) = FP(X_i) - adj(X_j)\) and \(FP(X_j) = FP(X_j) - adj(X_i)\). In Algorithm 4.2 this modification is introduced by adding lines 15 to 20 to the basic CHC algorithm (Algorithm 4.1).

In Figure 4.1, we compare the efficiency/performance trade-off between Hill Climbing and FastCHC for the BNn200p2m5 network, parameters and datasets are specified in Section 4.4. We can confirm that FastCHC is much more efficient than the unconstrained approach, at the expense of score loss.

4.3 Proposal

We have seen that the main disadvantage of constrained algorithms is that they experiment a loss of quality in the resulting networks when compared with the
Algorithm 4.1: Structural learning of BNs by using a Constrained Hill Climbing (CHC) algorithm [Gámez et al., 2012]

**Input:** $D$: A dataset defined over variables $V = \{X_1, \ldots, X_n\}$

**Input:** $G_0$: A DAG defined over $V$ used as the starting point for the search

**Output:** A DAG $G$ being the graphical part of network

\[ G \leftarrow G_0; \]
\[ f_0 \leftarrow f(G : D); \quad // f: decomposable scoring metric \]
\[ \text{For each } X_i \text{ do } FP(X_i) = \emptyset; \]
\[ \text{improvement } \leftarrow \text{true}; \]
\[ \text{while } \text{improvement } \text{do} \]
\[ \quad \text{improvement } \leftarrow \text{false}; \]
\[ \quad // \text{neighbors generated by addition} \]
\[ \quad \text{For each node } X_i \text{ and each node } X_j \notin (\Pi_5(X_i) \cup FP(X_i)) \text{ such that } X_j \rightarrow X_i \text{ does not introduce a directed cycle in } G, \text{ compute the difference } \]
\[ \quad \text{diff} = f(G + \{X_j \rightarrow X_i\} : D) - f_0. \text{ If } \text{diff} < 0 \text{ then add } \{X_j\} \text{ to } FP(X_i) \text{ and add } \{X_i\} \text{ to } FP(X_j). \text{ Store the change which maximizes diff in } \]
\[ \quad \langle \text{change}_a, \text{diff}_a \rangle; \]
\[ \quad // \text{neighbors generated by deletion} \]
\[ \quad \text{For each node } X_i \text{ and each node } X_j \in \Pi_5(X_i), \text{ compute the difference } \]
\[ \quad \text{diff} = f(G - \{X_j \rightarrow X_i\} : D) - f_0. \text{ If } \text{diff} > 0 \text{ then add } \{X_j\} \text{ to } FP(X_i) \text{ and add } \{X_i\} \text{ to } FP(X_j). \text{ Store the change which maximizes diff in } \]
\[ \quad \langle \text{change}_d, \text{diff}_d \rangle; \]
\[ \quad // \text{neighbors generated by reversal} \]
\[ \quad \text{For each node } X_i \text{ and each node } X_j \in \Pi_5(X_i) \text{ such that reversing } X_j \rightarrow X_i \]
\[ \quad \text{does not introduce a directed cycle in } G, \text{ compute the difference } \]
\[ \quad \text{diff} = d_1 + d_2 \text{ where } d_1 \text{ corresponds to } f(G - \{X_j \rightarrow X_i\} : D) - f_0 \text{ and } d_2 \]
\[ \quad \text{corresponds to } f(G + \{X_j \rightarrow X_i\} : D) - f_0. \text{ If } d_1 > 0 \text{ or } d_2 < 0 \text{ then add } \]
\[ \quad \{X_j\} \text{ to } FP(X_i) \text{ and add } \{X_i\} \text{ to } FP(X_j). \text{ Store the change which maximizes diff in } \]
\[ \quad \langle \text{change}_r, \text{diff}_r \rangle; \]
\[ \quad // \text{Checking if improvement} \]
\[ \quad \text{Let } d^* = \max_{k=a,d,r} \text{diff}_k \text{ and move* its corresponding change;} \]
\[ \quad \text{if } d^* > 0 \text{ then} \]
\[ \quad \quad \text{improvement } \leftarrow \text{true}; \]
\[ \quad \quad G \leftarrow \text{apply move* over } G; \]
\[ \quad \quad f_0 \leftarrow f_0 + d^*; \]
\[ \quad \text{end} \]
\[ \text{return } G; \]
Algorithm 4.2: Structural learning of BNs by using a Fast Constrained Hill Climbing (FastCHC) algorithm [Gámez et al., 2012]

\[\textbf{Input:}\ D:\ \text{A dataset defined over variables } V = \{X_1, \ldots, X_n\}\]
\[\textbf{Input:}\ G_0:\ A \text{ DAG defined over } V \text{ used as the starting point for the search}\]
\[\textbf{Output:}\ A \text{ DAG } G \text{ being the graphical part of network } \mathcal{B}\]

\begin{algorithm}
1 \text{if } \text{change}_a \neq 0 \text{ then} \\
2 \text{improvement } \leftarrow \text{true}; \\
3 G \leftarrow G_0; \\
4 f_5 \leftarrow f(G : D); \quad \text{//}: \text{ decomposable scoring metric}\]
5 \text{For each } X_i \text{ do } FP(X_i) = \emptyset; \\
6 \text{improvement } \leftarrow \text{false}; \\
7 \text{while improvement do} \\
8 \text{//neighbors generated by addition} \\
9 \text{For each node } X_i \text{ and each node } X_j \notin (\Pi_5(X_i) \cup FP(X_i)) \text{ such that } X_j \to X_i \text{ does not introduce a directed cycle in } G, \text{ compute the difference} \\
10 \text{diff} = f(G + \{X_j \to X_i\} : D) - f_5. \text{ If } \text{diff} < 0 \text{ then add } \{X_j\} \text{ to } FP(X_i) \text{ and add } \{X_j\} \text{ to } FP(X_j). \text{ Store the change which maximizes } \text{diff} \text{ in } \langle \text{change}_a, \text{diff}_a \rangle; \\
11 \text{//neighbors generated by deletion} \\
12 \text{For each node } X_i \text{ and each node } X_j \in \Pi_5(X_i), \text{ compute the difference} \\
13 \text{diff} = f(G - \{X_j \to X_i\} : D) - f_5. \text{ If } \text{diff} > 0 \text{ then add } \{X_j\} \text{ to } FP(X_i) \text{ and add } \{X_j\} \text{ to } FP(X_j). \text{ Store the change which maximizes } \text{diff} \text{ in } \langle \text{change}_d, \text{diff}_d \rangle; \\
14 \text{//neighbors generated by reversal} \\
15 \text{For each node } X_i \text{ and each node } X_j \in \Pi_5(X_i) \text{ such that reversing } X_j \to X_i \text{ does not introduce a directed cycle in } G, \text{ compute the difference} \\
16 \text{diff} = d_1 + d_2 \text{ where } d_1 \text{ corresponds to } f(G - \{X_j \to X_i\} : D) - f_5 \text{ and } d_2 \text{ corresponds to } f(G + \{X_j \to X_i\} : D) - f_5. \text{ If } d_1 > 0 \text{ or } d_2 < 0 \text{ then add} \\
17 \text{diff} \text{ in } \langle \text{change}_r, \text{diff}_r \rangle; \\
18 \text{//Checking if improvement} \\
19 \text{Let } d^* = \max_{k=a,d,r}\text{diff}_k \text{ and move* its corresponding change; } \\
20 \text{if } d^* > 0 \text{ then} \\
21 \text{improvement } \leftarrow \text{true;} \\
22 G \leftarrow G_0; \\
23 f_5 \leftarrow f_5 + d^*; \\
24 \text{if move* is adding } X_j \to X_i \text{ then} \\
25 FP(X_i) = FP(X_i) \setminus \text{NB}_j \\
26 \text{For all } X_a \in \text{NB}_j \text{ do } FP(X_a) = FP(X_a) \setminus \{X_i\} \\
27 FP(X_j) = FP(X_j) \setminus \text{NB}_i \\
28 \text{For all } X_b \in \text{NB}_i \text{ do } FP(X_b) = FP(X_b) \setminus \{X_j\}\]
29 \text{return } G;
\end{algorithm}
original unconstrained Hill Climbing algorithm [Gámez et al., 2012]. In this section we propose three modifications to the FastCHC algorithm in order to adjust the trade-off between the efficiency and accuracy of the constrained algorithm.

With these modification we pursue two different strategies. The first one is to relax the constraints imposed by the FastCHC algorithm, introducing a diversification factor in the search process. This is done by either releasing or not including some of the already discovered forbidden parents, thus letting the algorithm visit additional solutions that otherwise would remain unexplored. In the second one we try to identify the constraints that are less likely to generate errors, that is, constraints with a high confidence with respect to the data. Experimentally, we have noted that erroneously included constraints, via forbidden parents, make the algorithm degrade excessively. In any case, the proposed modifications maintain the theoretical properties of FastCHC as the original definition is more restrictive. This guarantees that the modifications will obtain local optima solutions satisfying the minimal I-map property as well.
4. EFFICIENCY TRADE-OFF IN STRUCTURAL LEARNING

4.3.1 Releasing Constraints from Variable Neighbourhood Levels

Our first proposal relaxes the constraints by releasing some of the forbidden parents dynamically at each iteration of the algorithm. We allow the algorithm to re-evaluate previously constrained arcs from the neighbourhood of the nodes involved in the last change performed. The intuition behind this decision is that their score is more likely to have changed since the evaluation performed before adding them to the forbidden parent sets.

In FastCHC, we already release constrains when adding new arcs, this modification extends that behaviour to deletion and reversal operations. The original proposal requires this relaxation of the $FP$ constrains in order to guarantee the minimal I-map condition. We are assuming a new meaning as a diversification technique, with the purpose of allowing the algorithm to explore additional solutions which could have a better score. In addition, we will release constraints from a wider neighbourhood to extend the unconstrained search space.

More formally, we include a new hyperparameter $L \in \mathbb{N}$ which represents the levels of neighbourhood from which we will release constrains, i.e., when a change involving nodes $X_i$ and $X_j$ is performed, a value $L = 1$ means that any node adjacent to $X_j$ ($adj^1(X_j)$) will be removed from $FP(X_i)$ and vice versa; for a level $L = 2$, any node $X_{jp} \in adj^1(X_j)$ and its respective adjacent nodes ($adj^2(X_j)$) will be removed from $FP(X_i)$ and vice versa; for the general case, $L = n$, any node in $adj^n(X_j)$ will be removed from $FP(X_i)$ and vice versa.

The results of this modification when $L = 1$ are similar to the original FastCHC definition, which releases the same constraints but only after performing addition moves. We have noted from the experimentation that, as $L$ increases, the algorithm obtains more accurate solutions with a minimum loss of efficiency. In addition, incrementing $L$ past a particular value will not improve the network accuracy and will just increase the computational requirements of the algorithm.

4.3.2 Limiting the Forbidden Parent Set Size

The main disadvantage of the previous approach is that it is difficult and unintuitive to tune the hyperparameter $L$ by hand, especially for large values when the behaviour of the algorithm becomes unpredictable. Our second proposal, from
now on FastCHCMod, releases constraints dynamically during the search process using the score metric value of the forbidden relationships.

More concretely, we impose a bound $S$ for the maximum number of $FP$ constraints that can be simultaneously stored in the $FP$ sets, so when the number of constraints reaches the limit some of them must be released in order to include new ones and keeping the number of constraints at $S$. Given that the algorithm constraints undirected arcs, they can be added and released from the $FP$ sets in pairs, i.e. if variable $X_i$ is added to $FP(X_j)$, $X_j$ will also be added to $FP(X_i)$, so we count both directions as one when computing the maximum number of constraints. As a result, $S$ refers to half the size of the sum of all the $FP$ set sizes:

$$\frac{1}{2} \sum_{i=1}^{n} \#FP(X_i) \leq S \quad (4.1)$$

This modification requires two design decisions to be defined:

- a suitable approximation for $S$, the maximum number of $FP$ constraints to be maintained.
- an update policy to determine which constraints must be kept in the $FP$ sets.

Using absolute values for $S$ might not be optimal, as the number of constraints that are discovered during a search process depends on the dimensionality of the dataset and the density of the graph. We will define the maximum number of $FP$ kept during a particular run of FastCHC as $S_{max}$, Figure 4.2 shows the distribution of this value for datasets with varying number of variables (described in Section 4.4). We can then define $S$ as a proportion of the maximum number of $FP$ for a dataset $S = \alpha S_{max}$, where $\alpha \in [0, 1]$ is a new hyperparameter of the learning algorithm. Since obtaining $S_{max}$ is not feasible without performing a full execution of the algorithm, we must find a suitable approximation that we could compute without increasing the complexity of the search.

As we mentioned in Section 4.2, FastCHC performs its first iterations very efficiently when compared to Hill Climbing (Figure 4.1). The explanation is that a large volume of constraints are added during this iteration, when the algorithm scores all possible arcs between all variables over the empty network. In fact, a large percent of the total $FP$ are added during this first step, we will refer to this
quantity to as $S_0$. Figure 4.3 shows the difference between the size of the $FP$ sets after the first iteration of the algorithm ($S_0$) and at the end of it $S_{max}$. The average ratio for the available datasets is 0.74, with a few exceptions. Since there is not a big difference between $S_{max}$ and $S_0$, we can use the latter value as an optimistic approximation that should meet our requirements. This way we can compute the bounded size $S$ of the $FP$ sets as a proportion of $S_0$, which we will compute during the first iteration of the algorithm:

$$S = \alpha \cdot S_0$$

In summary, we will perform the first iteration identically as in FastCHC, and use the number of forbidden parents discovered to set up $S$. From this point, the regular FastCHC algorithm is executed, but every time the $FP$ sets must be updated and some of the constraints need to be released in order to satisfy the imposed limit. An update policy must be defined in order to select the constrains that will be released once the sets reach their maximum size. For that, we will keep a list including the constraints ordered by their score at the time of being included (as they are forbidden they will not be updated any more). When the amount of constraints exceeds the limit, we will remove the one with the highest
score. We follow the intuition that this particular arc will be closer to a positive difference and therefore is more likely to be considered as a good candidate given the updated graph.

This modification does not introduce a significant efficiency penalty, given that the cost of the operations required to keep the bounded forbidden parents sets and the ordered list have a computational complexity of $O(n^2)$ in the worst case, so the complexity is still drawn from the exponential cost of the score metric computation. Having said that, these modifications make the algorithm less efficient as they normally imply a higher number of calls to the score metric function and several more iterations than FastCHC.

We can interpret the hyperparameter $\alpha$ as a balance between efficiency and the quality of the models. A value of $\alpha$ closer to 1.0 will keep the algorithm behaving much like FastCHC, whereas a value of $\alpha$ closer to 0.0 will keeps the algorithm’s behaviour closer to the Hill Climbing approach, while retaining the speed-up advantage of the first iteration of the original constrained algorithm.

This approach can seem similar to a Tabu Search (TS) [Glover, 1989], but in practice follows a completely different strategy. TS builds on top of HC and maintains a forbidden set of operations, however they remain being updated, as
4. EFFICIENCY TRADE-OFF IN STRUCTURAL LEARNING

the algorithms computes an aspiration criterion that can re-enable the movements after a number of iterations, in addition TS is a short-term memory algorithms and the restrictions expire after a number of iterations. This imply that TS is more demanding computationally than HC, while FastCHC\textsubscript{Mod2} is in fact more efficient as we do not explore the majority of the forbidden moves again.

4.3.3 Releasing Constraints Based on the Confidence in Conditional Independence Tests

The modifications proposed scale up the FastCHC algorithm in terms of accuracy by relaxing the restrictions imposed by using different approaches. However, its efficiency decreases because of the limitations imposed on the search space, even if we carefully manage the forbidden parent set as the previous proposals do. Experimentally, we have noticed two aspects with respect to the behaviour of the algorithm. The first one is the sensibility of the algorithms to wrongly included constraints in the forbidden parents set, and secondly, that these errors are more likely to happen in the later stages of the algorithm, once the size of the parent sets has increased. For these reasons, we aim to select the appropriate set of constraints to be released in order to minimize the efficiency loss.

We have defined two different strategies that rely on some of the BDeu score metric properties to release restrictions dynamically during the search process. In FastCHC\textsubscript{Mod1}, we release constraints related to the recent changes performed in the network, using a strategy which relies on the decomposability properties of the score metric. Given that the algorithm only re-evaluates the local score for the head node of the last modified arc, the forbidden parents from its neighbourhood have a higher chance of improving its score. On the other hand, FastCHC\textsubscript{Mod2} is focused on a quantitative approach regarding the score metric, believing that the higher the score assigned to the change, the greater the probability it will have of becoming a better candidate in the future.

Now, we introduce a third approach by interpreting the score metric as a conditional independence test, the theoretical basis of CHC algorithms, and their confidence given the limited amount of data. Furthermore, we pursue a simple strategy to manage the forbidden parent sets in order to maintain the algorithm’s original efficiency. It is well known that, with discrete variables, the confidence of the conditional independence test decreases drastically as its complexity increases.
that is, the number of variables involved in the test and the number of parameters to be estimated are larger. In general, we need exponentially more data as the number of variables increases in the tests in order to maintain the confidence.

This modification, from now on FastCHC\textsubscript{Mod3}, tries to determine which of the discovered independences are more likely to be wrong because of their limited confidence and should not be included as a restriction on the search space. As mentioned in Section 4.2, CHC algorithms make use of the BDeu score in order to check whether $I(X_i, X_j | \Pi(X_i))$ is true given the difference ($diff$) between the score of the current network and the one resulting from adding or removing an arc $X_j \rightarrow X_i$ between these two nodes. As the score metric is computed locally for $X_i$, the confidence depends on the size of its current parent set $\Pi(X_i)$, and the whole size of the CPT to be estimated, therefore also the error induced by the independence test.

Consequently, the absence of arcs in the network in an early stage of the algorithm will result in performing marginal independence tests $I(X_i, X_j | \emptyset)$, which have presumably low error and a high level of confidence. However, as arcs are added to the network, the algorithm will perform conditional independence $I(X_i, X_j | \Pi(X_i))$ tests which are likely to have a higher error as the number of parents for the conditioned variable increases, so the number of parameters. This modification tries to incorporate this concept by using an approximate criterion such as limiting the number of conditioning variables in the tests performed. We add a new hyperparameter $P \in \mathbb{N}$ which fixes a threshold to the maximum number of parents for the node $X_i$ required to add a new restriction to the FP sets when a negative difference is calculated for that node using the BDeu score. More formally, given the addition or deletion of $X_j \rightarrow X_i$ we will only include new constraints if $diff < 0$ and $|\Pi(X_i)| < P$; i.e. for a value of $P = 0$ only marginal independences will be taken into account in order to add new forbidden parents.

### 4.4 Experimental Evaluation

In this section we present a comparative study of the three different modifications proposed for the FastCHC algorithm. We have performed an empirical evaluation for each approach using several hyperparameter configurations and databases.
4. EFFICIENCY TRADE-OFF IN STRUCTURAL LEARNING

4.4.1 Algorithms

Our study includes the original constrained FastCHC algorithm and the Hill Climbing approach for reference purposes. We have evaluated several instances for each one of the three modifications presented in this paper by choosing different configurations for their parameters:

- FastCHC\text{Mod1} has been evaluated for values $L = 2, 3, 5, 8$ and $12$, as preliminary experiments show that setting $L$ to higher values would not improve the performance of the algorithm.

- FastCHC\text{Mod2} has been configured for different values of the $\alpha$ parameter in $[0, 1]$; in particular, we have selected $\alpha = 0.4, 0.6$, and $0.8$.

- FastCHC\text{Mod3} has been evaluated for values $P = 0, 1$, and $2$, as higher values do not provide good confidence levels for conditional independence tests.

In addition, we have included two state-of-the-art algorithms for Bayesian networks structural learning, namely GES [Chickering, 2002] and MMCH [Tsamardinos et al., 2006], to provide representative comparison results. We have run our own GES implementation based on the tetrad \(^1\) project and for the MMHC algorithm we have taken the MMPC implementation from the author’s original package called Causal Explorer [Tsamardinos et al., 2006], setting the available parameters to the default configuration, and then used the skeleton learned to constrain our Hill Climbing algorithm for a fair comparison with our implementation.

4.4.2 Implementation and Running Environment

The methods have been implemented in Java using the ProGrMo library for dataset and graph structures management [Gámez et al., 2010]. The score metric used is the Bayesian Equivalent Uniform (BDeu) with an equivalent sample size of 10 and all other parameters set up as in related work [Gámez et al., 2012]. We also take advantage of an internal cache [Gámez et al., 2011] to improve the efficiency of the score metric implementation, saving the result of every score computation using the probability family as a hash key in order to re-use it later in the execution. This achieves high computational savings especially in larger domains. All the runs

\(^1\)http://www.phil.cmu.edu/projects/tetrad/
of these algorithms were conducted on a dedicated server with Pentium Xeon 3.0 Ghz, 64 bit architecture, 8 Gb RAM memory.

4.4.3 Performance Indicators

We consider two kinds of factors as performance indicators to compare the different algorithms: the quality of the obtained network, which is given by the value of the scoring metric (BDeu), and the efficiency of the algorithm, which is given by the number of score function computations carried out by each algorithm (calls). As the execution time depends on both the implementation and the specification of the computer on which the algorithm is executed, we consider the score function calls as being independent of these factors and as having a direct correspondence with CPU time requirements. However we offer an execution time comparison between the HC, FastCHC and its modifications, as all of them share the same implementation platform; the results obtained can give us an idea of the computation time magnitude for these approaches.

4.4.4 Experiments

We have carried out an extensive set of experiments using publicly available networks which represent problems from real life applications in a range of different domains; also, they have been used widely to evaluate several state-of-the-art BN learning algorithms and can be found frequently in the literature. We have obtained the networks from the Bayesian network repository hosted on the bnlearn R package website\(^1\).

Although these networks offer high diversity with respect to dimensionality, ranging from 20 to 724 variables, we wished to extend our experiments by adding some big synthetic networks. We expect these problem to provide a nice example on the capabilities of the proposed methods when solving high dimensional problems. For this purpose, we have generated a set of random Bayesian networks with different degrees of difficulty [Alonso-Barba et al., 2013]. A summary of the properties for all these networks can be found in Table 4.1. For each database we have generated 5 samples of 5000 instances each.

\(^1\)http://www.bnlearn.com/bnrepository/
4. EFFICIENCY TRADE-OFF IN STRUCTURAL LEARNING

Table 4.1: Main characteristics of the networks used in the experiments.

<table>
<thead>
<tr>
<th>Network</th>
<th># Var</th>
<th># Edges</th>
<th>Max Parents</th>
<th>Domain</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarm</td>
<td>37</td>
<td>46</td>
<td>4</td>
<td>Medicine</td>
<td>Beinlich et al. [1989]</td>
</tr>
<tr>
<td>Andes</td>
<td>223</td>
<td>338</td>
<td>6</td>
<td>Education</td>
<td>Conati et al. [1997]</td>
</tr>
<tr>
<td>barley</td>
<td>48</td>
<td>84</td>
<td>4</td>
<td>Agriculture</td>
<td>Kristensen &amp; Rasmussen [2002]</td>
</tr>
<tr>
<td>Child</td>
<td>20</td>
<td>25</td>
<td>2</td>
<td>Medicine</td>
<td>Cowell et al. [1999]</td>
</tr>
<tr>
<td>Diabetes</td>
<td>413</td>
<td>602</td>
<td>2</td>
<td>Medicine</td>
<td>Andreassen et al. [1991]</td>
</tr>
<tr>
<td>Hallfinder</td>
<td>56</td>
<td>66</td>
<td>4</td>
<td>Meteorology</td>
<td>Jensen &amp; Jensen [1996]</td>
</tr>
<tr>
<td>Hepar2</td>
<td>70</td>
<td>1236</td>
<td>6</td>
<td>Medicine</td>
<td>Onisko et al. [1998]</td>
</tr>
<tr>
<td>Insurance</td>
<td>27</td>
<td>52</td>
<td>3</td>
<td>Insurance</td>
<td>Binder et al. [1997]</td>
</tr>
<tr>
<td>Link</td>
<td>724</td>
<td>1125</td>
<td>3</td>
<td>Genetics</td>
<td>Jensen [1997]</td>
</tr>
<tr>
<td>Mildew</td>
<td>35</td>
<td>46</td>
<td>3</td>
<td>Agriculture</td>
<td>Jensen &amp; Jensen [1996]</td>
</tr>
<tr>
<td>Munin1</td>
<td>189</td>
<td>282</td>
<td>3</td>
<td>Medicine</td>
<td>Andreassen et al. [1989]</td>
</tr>
<tr>
<td>Pathfinder</td>
<td>135</td>
<td>200</td>
<td>130</td>
<td>Medicine</td>
<td>Heckerman &amp; Bharat [1992]</td>
</tr>
<tr>
<td>Pigs</td>
<td>441</td>
<td>592</td>
<td>2</td>
<td>Genetics</td>
<td>Jensen &amp; Kong [1999]</td>
</tr>
<tr>
<td>Water</td>
<td>32</td>
<td>66</td>
<td>5</td>
<td>Water Treatment</td>
<td>Jensen et al. [1989]</td>
</tr>
<tr>
<td>Win95PTS</td>
<td>76</td>
<td>112</td>
<td>7</td>
<td>Troubleshooting</td>
<td>Heckerman et al. [1995a]</td>
</tr>
<tr>
<td>BNn100p1m2</td>
<td>100</td>
<td>99.6</td>
<td>5.0</td>
<td>Generated</td>
<td>-</td>
</tr>
<tr>
<td>BNn100p1m5</td>
<td>100</td>
<td>98.6</td>
<td>4.9</td>
<td>Generated</td>
<td>-</td>
</tr>
<tr>
<td>BNn100p2m2</td>
<td>100</td>
<td>197.5</td>
<td>7.8</td>
<td>Generated</td>
<td>-</td>
</tr>
<tr>
<td>BNn100p2m5</td>
<td>100</td>
<td>197.5</td>
<td>7.9</td>
<td>Generated</td>
<td>-</td>
</tr>
<tr>
<td>BNn200p1m2</td>
<td>200</td>
<td>190.7</td>
<td>5.4</td>
<td>Generated</td>
<td>-</td>
</tr>
<tr>
<td>BNn200p1m5</td>
<td>200</td>
<td>191.6</td>
<td>5.3</td>
<td>Generated</td>
<td>-</td>
</tr>
<tr>
<td>BNn200p2m2</td>
<td>200</td>
<td>392.7</td>
<td>8.2</td>
<td>Generated</td>
<td>-</td>
</tr>
<tr>
<td>BNn200p2m5</td>
<td>200</td>
<td>393.0</td>
<td>8.4</td>
<td>Generated</td>
<td>-</td>
</tr>
</tbody>
</table>

4.4.5 Empirical Results

Table 4.2 summarizes the results from running all algorithms over the 23 datasets, these have obtained as the average of executions over 5 independent data samples. We have aggregated the results over all datasets for better readability\(^1\), this way the values reported correspond with the ratio between each algorithm and a baseline method, in this case the unconstrained Hill Climbing algorithm. The ratio for the score metric calls has been computed using the mean of every individual execution \(\text{Calls(Model}_i)/\text{Calls(Model}_\text{HC})\) for each of the other algorithm \(i\). The ratio for the BDeu score\(^2\) has been computed by using \(\exp((\text{BDeu(Model}_i) - \text{BDeu}(\text{HC}))/m)\). In addition, we have included the standard deviation in order to check for outliers, as ratios are very sensitive to networks with particular structures that bias the learning process.

As we can see, FastCHC is the most efficient algorithm; MMHC results are

\(^{1}\)Detailed results are included in Section 4.7, where Tables 4.5 and 4.6 show the BDeu score metric value and calls for each algorithm and database respectively. In addition detailed results for execution time can be found in Figure 4.8.

\(^{2}\)This ratio can be interpreted statistically as the ratio between the probability that \(\text{Model}_i\) and model \(\text{HC}\) assign to the next data sample.
not stable and depend on the dataset, it being one of the most efficient algorithms for some datasets and the worst for others, especially for the largest ones. This variability in the algorithm’s behaviour results in a high deviation with respect to the score metric calls ratio shown in Table 4.2, which greatly decreases its averaged performance. This behaviour shows that this algorithm does not scale correctly in all domains. If we look at the detailed results in Section 4.7 we can observe that there are some databases for which MMHC performance is really poor. Even so, FastCHC is generally better and steady over all domains, thus being a more scalable approach. On the other hand, the GES and Hill Climbing algorithms are the less efficient ones, respectively, whereas they obtain the best scores.

If we compare our proposal with FastCHC, we can confirm how the modifications improve the score results with a minimum loss of efficiency. When comparing the modifications among themselves, FastCHC_{Mod1} is more efficient than FastCHC_{Mod2}, but, the latter can be scaled up by increasing $\alpha$, giving a better idea of how to tune the hyperparameter by hand. Also, in FastCHC_{Mod1} for values of $L \geq 8$ the algorithm starts to obtain a worse trade-off and thus its scalability will be limited. It seems that FastCHC_{Mod2} with $alpha = 0.4$ obtains the best BDeu score ratio, but as we can see it has also a high deviation, which is due to atypical values obtained from the diabetes network, as can be verified in Table 4.5. With respect to FastCHC_{Mod3}, we can observe that relying only on marginal independence tests, by setting up $P = 0$, gives the algorithm the best improvement in accuracy while maintaining a very good trade-off, as its efficiency is close to that of the original FastCHC algorithm.

In Figure 4.4 we provide a graphical representation of this comparison, where we can confirm the expected behaviour of the modifications. FastCHC is displayed in the bottom left corner as the most efficient but least accurate algorithm and Hill Climbing in the upper right corner, being the least efficient algorithm which recovers the best networks. Regarding the modifications, it is possible to distinguish their different behaviour and the influence of each hyperparameter configuration. GES has been removed from this graphic and will not be included from this point onward in the following results because, although it obtains the most accurate results, it has proven to be the least efficient algorithm and its results are beyond the scale of this scalability study. MMHC has also been removed from this plot as the odd results for the conflictive datasets will not allow its general behaviour to be fairly represented on the plot.
Table 4.2: BDeu Score and Score Metric Calls ratio computed for each algorithm with respect to Hill Climbing. Averaged over all 23 datasets. Standard deviation for the ratios is shown in brackets. Results highlighted in bold represent ratios with high deviation that include atypical values that do not generalize the behaviour of the algorithm.
4.4.6 Combining Marginal Tests with other Modifications

The previous results show that \( \text{FastCHC}_{Mod3} \) outperforms the other approaches when we fix the \( P \) parameter to 0. This configuration only considers forbidden parents by relying in scores that have been computed for a node with an empty parent set. If we increase \( P \), the algorithm accepts constraints discovered by performing higher order independence tests, thus inducing critical errors that dramatically reduce the accuracy of the network. This result endorses the experiments carried out in Section 4.3 where we observed that approximately seventy percent of the total constraints are discovered in the first iteration of the algorithm (Figure 4.3) by performing marginal independence tests from the empty graph.

---

**Figure 4.4:** Trade-off between score metric calls and score metric for the first set of experiments. The legend shows the averaged ratio (calls, score) relative to the Hill Climbing.
It is feasible to combine the first two modifications with the third one, instead of using FastCHC as the base algorithm. We have carried out a second set of experiments using this new configuration: FastCHC_{Mod3} with \( P = 0 \), plus releasing constraints from a variable neighbourhood (FastCHC_{Mod3+1}) and limiting the \( FP \) set size (FastCHC_{Mod3+2}); for each modification we have selected the same configuration for hyperparameter \( L \) and \( \alpha \) as in the previous experiments.

Again, a summarized\(^{1}\) comparison can be found in Table 4.3 which uses the same methodology as for Table 4.3. We can observe how this new approach scales the algorithm similarly to the previous experiment, but obtains much better results. In Figure 4.5 we have included a new graphical comparison in which we can confirm the expected behaviour of these new configurations. Given these results, the combination of modifications 3 ans 2 FastCHC\_{Mod3+2} achieve better performance than FastCHC\_{Mod3+1} in general (instances D-F) achieve better performance than FastCHC\_{Mod3+1} in general (instances H-K).

However, notice that the improvement range is much smaller than before.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>BDeu Score Ratio</th>
<th>Algorithm</th>
<th>Metric Calls Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>HillClimbing</td>
<td>1.0000 (0.000)</td>
<td>FastCHC</td>
<td>0.3637 (0.161)</td>
</tr>
<tr>
<td>FastCHC_{Mod3+2a=0.4}</td>
<td>0.9025 (0.223)</td>
<td>FastCHC_{Mod3P=0}</td>
<td>0.4360 (0.213)</td>
</tr>
<tr>
<td>FastCHC_{Mod3+2a=0.6}</td>
<td>0.8658 (0.254)</td>
<td>FastCHC_{Mod3+1L=2}</td>
<td>0.4567 (0.214)</td>
</tr>
<tr>
<td>FastCHC_{Mod3+1L=12}</td>
<td>0.8412 (0.302)</td>
<td>FastCHC_{Mod3+1L=3}</td>
<td>0.4867 (0.212)</td>
</tr>
<tr>
<td>FastCHC_{Mod3+1L=8}</td>
<td>0.8378 (0.301)</td>
<td>FastCHC_{Mod3+2a=0.8}</td>
<td>0.5130 (0.187)</td>
</tr>
<tr>
<td>FastCHC_{Mod3+1L=5}</td>
<td>0.8162 (0.303)</td>
<td>FastCHC_{Mod3+1L=5}</td>
<td>0.5373 (0.204)</td>
</tr>
<tr>
<td>FastCHC_{Mod3+2a=0.8}</td>
<td>0.7875 (0.319)</td>
<td>FastCHC_{Mod3+1L=8}</td>
<td>0.5687 (0.191)</td>
</tr>
<tr>
<td>FastCHC_{Mod3+1L=3}</td>
<td>0.7763 (0.324)</td>
<td>FastCHC_{Mod3+1L=12}</td>
<td>0.5789 (0.185)</td>
</tr>
<tr>
<td>FastCHC_{Mod3+1L=2}</td>
<td>0.7524 (0.333)</td>
<td>FastCHC_{Mod3+2a=0.6}</td>
<td>0.5921 (0.158)</td>
</tr>
<tr>
<td>FastCHC_{Mod3P=0}</td>
<td>0.7400 (0.337)</td>
<td>FastCHC_{Mod3+2a=0.4}</td>
<td>0.6768 (0.128)</td>
</tr>
<tr>
<td>FastCHC</td>
<td>0.5510 (0.396)</td>
<td>HillClimbing</td>
<td>1.0000 (0.000)</td>
</tr>
<tr>
<td>MMHC</td>
<td>0.4266 (0.348)</td>
<td>MMHC</td>
<td>(1.8166 (2.546))</td>
</tr>
</tbody>
</table>

Table 4.3: BDeu Score and Score Metric Calls ratio computed for each algorithm with respect to Hill Climbing. Averaged over all 23 datasets. Standard deviation for the ratios is shown in brackets. Results highlighted in bold represent ratios with high deviation that include atypical values that do not generalize the behaviour of the algorithm.

\(^{1}\)Detailed results are included in Section 4.7, where Tables 4.7 and 4.8 show the BDeu score metric value and calls for each algorithm and database respectively.
4.4.7 Statistical Tests

We have extended the previous comparison by performing statistical tests to compare the quality of the networks, we follow the methodology suggested in Demšar [2006]; García & Herrera [2008]. We have performed a Friedman rank test [Friedman, 1940] followed by a post-hoc analysis using Holm’s procedure [Holm, 1979], to correct the familywise error when comparing the algorithms. Table 4.4 shows the average rankings for the different algorithms regarding BDeu score as well as the corrected $p-$value for comparing each algorithm with the best one, the control procedure, which in this case is Hill Climbing. With a 95% confidence level, we reject the rank test hypothesis that all algorithms obtain equivalent models in quality for all 23 datasets. Individual comparisons show that several configurations of FastCHC are statistically significant to HillClimbing, highlighted entries...
in Table 4.4 denote the corrected $p-$values for which the hypothesis is not rejected by the test procedure.

A visual representation of this comparison is shown in Figure 4.6, where the distribution of ranks and non-rejected hypotheses are represented jointly. We can confirm that HillClimbing is not always the best algorithm with a inconsistent rank distribution. The equivalent modifications perform evenly among the 23 datasets, and there is not a clear pattern one of the over performing the rest.

Additional insights can be extracted by comparing these approaches on their efficiency. For that we have computed the same ranking distribution regarding score metric calls\(^1\), as can be seen in Figure 4.7. As we might expect, the modifications that proved to be statistically equivalent to HC are at the tail of the ranking distribution, but always before HillClimbing. From these configuration, the moderated ones such as FastCHC\(_{Mod3+2\alpha=0.6}\) obtains a good ranking distribution, and shows a good balance between efficiency and accuracy.

<table>
<thead>
<tr>
<th>method</th>
<th>rank</th>
<th>$p-$value</th>
<th>win</th>
<th>tie</th>
<th>loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>HillClimbing</td>
<td>2.76</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FastCHC(_{Mod3+2\alpha=0.4})</td>
<td>3.67</td>
<td><strong>3.5053e-01</strong></td>
<td>13</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>FastCHC(_{Mod3+2\alpha=0.6})</td>
<td>4.39</td>
<td><strong>1.9100e-01</strong></td>
<td>14</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>FastCHC(_{Mod3+1L=12})</td>
<td>4.61</td>
<td><strong>1.7653e-01</strong></td>
<td>16</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>FastCHC(_{Mod3+1L=8})</td>
<td>4.96</td>
<td><strong>9.9073e-02</strong></td>
<td>17</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>FastCHC(_{Mod3+1L=5})</td>
<td>5.67</td>
<td>1.4483e-02</td>
<td>17</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>FastCHC(_{Mod3+1L=12})</td>
<td>6.00</td>
<td>5.5593e-03</td>
<td>16</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>FastCHC(_{Mod3+1L=3})</td>
<td>6.78</td>
<td>2.7442e-04</td>
<td>19</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>FastCHC(_{Mod3+1L=2})</td>
<td>7.91</td>
<td>1.1035e-06</td>
<td>19</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>FastCHC(_{Mod3})</td>
<td>8.59</td>
<td>2.3118e-08</td>
<td>19</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>FastCHC</td>
<td>10.65</td>
<td>7.1074e-15</td>
<td>23</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.4: Statistical analysis for BDeu score results for the second set of experiments from Table 4.7.

The rank columns shows the averaged rank computed for the Friedman test and $p-$value shows the corrected statistic by the Holm procedure. Highlighted entries represent rejected hypotheses. Win/tie/loss records show the comparison of the control algorithm with each other, i.e. the second row shows that HC obtained higher scores 13 times, equivalent 7 times and worse 3 times when compared to FastCHC\(_{Mod3+2\alpha=0.4}\).

\(^1\)Notice that statistical tests are not performed for score metric calls as the rankings are overly dominated by HillClimbing and FastCHC being always the worst and best algorithm respectively.
4.5 Conclusions

We have proposed three modifications for the constrained local search FastCHC algorithm in order to obtain higher quality solutions closer to the state-of-the-art algorithms while maintaining an efficient algorithm. Each modification is based on different properties of the scoring metric and has been designed in order to maintain the overall complexity of the original algorithm.

Each modification introduce a different hyperparameter that can be tuned to adjust the behaviour of the algorithm balancing its efficiency and quality trade-off. We have performed an extensive empirical evaluation using different configurations for each approach. In the case of FastCHC\textsubscript{Mod1}, the $L$ hyperparameter could be difficult to set manually as it normally reaches a maximum value beyond which it would not obtain any improvements. On the other hand, FastCHC\textsubscript{Mod2} hyperparameter is bounded to the interval $[0,1]$ and gives us a good intuition of how it

Figure 4.6: Ranks computed for the Friedman test regarding BDeu Score. Shadowed boxes refer to non rejected hypotheses.
4. EFFICIENCY TRADE-OFF IN STRUCTURAL LEARNING

Figure 4.7: Ranks computed for the Friedman test regarding score metric calls. Shadowed boxes refer to non rejected hypotheses.

affects the algorithm’s behaviour: with values closer to 0 it performs similarly to FastCHC and with values closer to 1 it performs similarly to Hill Climbing. Finally, experimental results show that the FastCHC\textsubscript{Mod3} hyperparameter $P$ reaches an optimal value of 0 when the algorithm only takes into account marginal independence tests.

We have combined this specific modification with the other two, obtaining better results with respect to the aforementioned accuracy/efficiency trade-off. In addition, statistical tests reveal that some of the combined modifications instances obtain comparable accuracy results with respect to Hill Climbing with minimum efficiency loss. In particular, the best approach is the combination of FastCHC\textsubscript{Mod3} with $P = 0$ and FastCHC\textsubscript{Mod2} with values of $\alpha$ ranging from 0.6 to 0.8.

In summary, FastCHC has proven to be a suitable algorithm for Bayesian network structural learning as it compares well with state-of-the-art algorithms
such as MMHC, which was not comparable in terms of efficiency in our evaluation. The results obtained show that FastCHC is more efficient in the general case than MMHC, and especially at larger domains; moreover, MMHC is unstable when learning from some high dimensional domains which is not a desired property for scalability purposes. In addition, FastCHC has obtained significantly better accuracy results, and we have demonstrated that the proposed modifications can scale up the algorithm in terms of accuracy obtaining results closer to the original Hill Climbing with big computational savings.

In further work, we could study extensions to the FastCHC$\text{Mod}_3$ modification by refining the constraints imposed by the $P$ parameter using a more elaborated criterion: taking into account not only the size of the parent set but also the size of the resulting CPT and the amount of available data for the independence test. This would increase the computational cost of the tests, but would allow the algorithm to detect new constraints in further steps of the search process and to release some of the early discovered constraints that allow the unconstrained Hill Climbing to obtain better results in the later stages of the learning process.

### 4.6 Reproducibility and Extended Results

Available code and data for further use of the algorithms presented in this chapter as well as instructions to reproduce the experiments can be found on [http://simd.albacete.org/supplements/FastCHC.html](http://simd.albacete.org/supplements/FastCHC.html).

### 4.7 Detailed Results

This sections includes detailed results for the presented experiments. We preferred to include them at the end of the chapter for the sake of readability, but we invite the reader to check the individual results for each database, as they hold valuable insights regarding the behaviour of the exposed algorithms.
Table 4.6: Detailed results for score metric calls from the first set of experiments. The results highlighted in bold are the best for the corresponding network. Italic results marked in the MMHC column indicate databases with no scalable efficiency. Empty cells correspond to interrupted executions due to wall-time restrictions.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>HC</th>
<th>MMHC</th>
<th>GES</th>
<th>FastCHC</th>
<th>FastCHC_{Mod=1} (L = 2)</th>
<th>FastCHC_{Mod=1} (L = 3)</th>
<th>FastCHC_{Mod=1} (L = 5)</th>
<th>FastCHC_{Mod=4} (L = 8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>alarm</td>
<td>3391</td>
<td>3714</td>
<td>5731</td>
<td>1376</td>
<td>1484</td>
<td>1813</td>
<td>1871</td>
<td>2001</td>
</tr>
<tr>
<td>andow</td>
<td>11253</td>
<td>12018</td>
<td>NA</td>
<td>206845</td>
<td>30395</td>
<td>30724</td>
<td>35225</td>
<td>41013</td>
</tr>
<tr>
<td>BNN100p1m2</td>
<td>21665</td>
<td>13274</td>
<td>4210</td>
<td>63864</td>
<td>65766</td>
<td>6921</td>
<td>7835</td>
<td>8692</td>
</tr>
<tr>
<td>BNN100p1m5</td>
<td>21317</td>
<td>13273</td>
<td>36970</td>
<td>55984</td>
<td>59881</td>
<td>6148</td>
<td>6779</td>
<td>7444</td>
</tr>
<tr>
<td>BNN100p2m2</td>
<td>32426</td>
<td>19446</td>
<td>45010</td>
<td>71615</td>
<td>8077</td>
<td>10450</td>
<td>15999</td>
<td>17216</td>
</tr>
<tr>
<td>BNN100p2m5</td>
<td>27439</td>
<td>18568</td>
<td>36172</td>
<td>59633</td>
<td>6350</td>
<td>7114</td>
<td>9846</td>
<td>11564</td>
</tr>
<tr>
<td>BNN200p1m2</td>
<td>81764</td>
<td>49573</td>
<td>178075</td>
<td>23453</td>
<td>23761</td>
<td>24364</td>
<td>26115</td>
<td>28731</td>
</tr>
<tr>
<td>BNN200p1m5</td>
<td>82162</td>
<td>47591</td>
<td>129341</td>
<td>21376</td>
<td>21996</td>
<td>22965</td>
<td>24600</td>
<td>25716</td>
</tr>
<tr>
<td>BNN200p2m2</td>
<td>127266</td>
<td>64130</td>
<td>197025</td>
<td>24758</td>
<td>26715</td>
<td>31317</td>
<td>43359</td>
<td>60109</td>
</tr>
<tr>
<td>BNN200p2m5</td>
<td>107639</td>
<td>56285</td>
<td>152983</td>
<td>21692</td>
<td>22974</td>
<td>30804</td>
<td>43859</td>
<td>66500</td>
</tr>
<tr>
<td>alarm</td>
<td>3391</td>
<td>3714</td>
<td>5731</td>
<td>1376</td>
<td>1484</td>
<td>1813</td>
<td>1871</td>
<td>2001</td>
</tr>
<tr>
<td>andow</td>
<td>11253</td>
<td>12018</td>
<td>NA</td>
<td>206845</td>
<td>30395</td>
<td>30724</td>
<td>35225</td>
<td>41013</td>
</tr>
<tr>
<td>BNN100p1m2</td>
<td>21665</td>
<td>13274</td>
<td>4210</td>
<td>63864</td>
<td>65766</td>
<td>6921</td>
<td>7835</td>
<td>8692</td>
</tr>
<tr>
<td>BNN100p1m5</td>
<td>21317</td>
<td>13273</td>
<td>36970</td>
<td>55984</td>
<td>59881</td>
<td>6148</td>
<td>6779</td>
<td>7444</td>
</tr>
<tr>
<td>BNN100p2m2</td>
<td>32426</td>
<td>19446</td>
<td>45010</td>
<td>71615</td>
<td>8077</td>
<td>10450</td>
<td>15999</td>
<td>17216</td>
</tr>
<tr>
<td>BNN100p2m5</td>
<td>27439</td>
<td>18568</td>
<td>36172</td>
<td>59633</td>
<td>6350</td>
<td>7114</td>
<td>9846</td>
<td>11564</td>
</tr>
<tr>
<td>BNN200p1m2</td>
<td>81764</td>
<td>49573</td>
<td>178075</td>
<td>23453</td>
<td>23761</td>
<td>24364</td>
<td>26115</td>
<td>28731</td>
</tr>
<tr>
<td>BNN200p1m5</td>
<td>82162</td>
<td>47591</td>
<td>129341</td>
<td>21376</td>
<td>21996</td>
<td>22965</td>
<td>24600</td>
<td>25716</td>
</tr>
<tr>
<td>BNN200p2m2</td>
<td>127266</td>
<td>64130</td>
<td>197025</td>
<td>24758</td>
<td>26715</td>
<td>31317</td>
<td>43359</td>
<td>60109</td>
</tr>
<tr>
<td>BNN200p2m5</td>
<td>107639</td>
<td>56285</td>
<td>152983</td>
<td>21692</td>
<td>22974</td>
<td>30804</td>
<td>43859</td>
<td>66500</td>
</tr>
<tr>
<td>alarm</td>
<td>3391</td>
<td>3714</td>
<td>5731</td>
<td>1376</td>
<td>1484</td>
<td>1813</td>
<td>1871</td>
<td>2001</td>
</tr>
<tr>
<td>andow</td>
<td>11253</td>
<td>12018</td>
<td>NA</td>
<td>206845</td>
<td>30395</td>
<td>30724</td>
<td>35225</td>
<td>41013</td>
</tr>
<tr>
<td>BNN100p1m2</td>
<td>21665</td>
<td>13274</td>
<td>4210</td>
<td>63864</td>
<td>65766</td>
<td>6921</td>
<td>7835</td>
<td>8692</td>
</tr>
<tr>
<td>BNN100p1m5</td>
<td>21317</td>
<td>13273</td>
<td>36970</td>
<td>55984</td>
<td>59881</td>
<td>6148</td>
<td>6779</td>
<td>7444</td>
</tr>
<tr>
<td>BNN100p2m2</td>
<td>32426</td>
<td>19446</td>
<td>45010</td>
<td>71615</td>
<td>8077</td>
<td>10450</td>
<td>15999</td>
<td>17216</td>
</tr>
<tr>
<td>BNN100p2m5</td>
<td>27439</td>
<td>18568</td>
<td>36172</td>
<td>59633</td>
<td>6350</td>
<td>7114</td>
<td>9846</td>
<td>11564</td>
</tr>
<tr>
<td>BNN200p1m2</td>
<td>81764</td>
<td>49573</td>
<td>178075</td>
<td>23453</td>
<td>23761</td>
<td>24364</td>
<td>26115</td>
<td>28731</td>
</tr>
<tr>
<td>BNN200p1m5</td>
<td>82162</td>
<td>47591</td>
<td>129341</td>
<td>21376</td>
<td>21996</td>
<td>22965</td>
<td>24600</td>
<td>25716</td>
</tr>
<tr>
<td>BNN200p2m2</td>
<td>127266</td>
<td>64130</td>
<td>197025</td>
<td>24758</td>
<td>26715</td>
<td>31317</td>
<td>43359</td>
<td>60109</td>
</tr>
<tr>
<td>BNN200p2m5</td>
<td>107639</td>
<td>56285</td>
<td>152983</td>
<td>21692</td>
<td>22974</td>
<td>30804</td>
<td>43859</td>
<td>66500</td>
</tr>
</tbody>
</table>
Figure 4.8: Approximate execution times for HC, FastCHC and each modification on each dataset.
Table 4.7: Detailed results for BDeu score calls from the second set of experiments. The results highlighted in bold are the best for the corresponding network.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>HC</th>
<th>MMDU</th>
<th>FastCHC</th>
<th>FastCHC</th>
<th>FastCHC</th>
</tr>
</thead>
<tbody>
<tr>
<td>alarm</td>
<td>-53383.14</td>
<td>-53383.14</td>
<td>-53383.14</td>
<td>-53340.94</td>
<td>-53340.94</td>
</tr>
<tr>
<td>andes</td>
<td>-46862.03</td>
<td>-46862.03</td>
<td>-46862.03</td>
<td>-46862.03</td>
<td>-46862.03</td>
</tr>
<tr>
<td>barley</td>
<td>-261460.34</td>
<td>-261460.34</td>
<td>-261460.34</td>
<td>-261460.34</td>
<td>-261460.34</td>
</tr>
<tr>
<td>BNn100p1m2</td>
<td>-214228.50</td>
<td>-214228.50</td>
<td>-214231.20</td>
<td>-214086.74</td>
<td>-214156.05</td>
</tr>
<tr>
<td>BNn100p1m5</td>
<td>-429321.68</td>
<td>-429212.24</td>
<td>-429212.24</td>
<td>-429501.73</td>
<td>-429434.88</td>
</tr>
<tr>
<td>BNn100p2m2</td>
<td>-216411.64</td>
<td>-216360.76</td>
<td>-216360.76</td>
<td>-216207.36</td>
<td>-217307.42</td>
</tr>
<tr>
<td>BNn100p2m5</td>
<td>-455528.19</td>
<td>-455142.96</td>
<td>-455104.47</td>
<td>-455691.36</td>
<td>-455371.71</td>
</tr>
<tr>
<td>BNn200p1m2</td>
<td>-437182.38</td>
<td>-437075.69</td>
<td>-436881.69</td>
<td>-436000.29</td>
<td>-436715.62</td>
</tr>
<tr>
<td>BNn200p1m5</td>
<td>-873201.36</td>
<td>-873006.44</td>
<td>-872898.66</td>
<td>-872226.61</td>
<td>-872361.47</td>
</tr>
<tr>
<td>BNn200p2m2</td>
<td>-446460.09</td>
<td>-445432.09</td>
<td>-445463.33</td>
<td>-444669.00</td>
<td>-447944.59</td>
</tr>
<tr>
<td>BNn200p2m5</td>
<td>-923422.66</td>
<td>-921977.79</td>
<td>-921911.68</td>
<td>-921984.22</td>
<td>-921980.91</td>
</tr>
<tr>
<td>child</td>
<td>-61743.12</td>
<td>-61743.12</td>
<td>-61743.12</td>
<td>-61743.12</td>
<td>-61743.12</td>
</tr>
<tr>
<td>diabetes</td>
<td>-1124113.20</td>
<td>-1246011.95</td>
<td>-1124113.20</td>
<td>-1124380.43</td>
<td>-1251748.05</td>
</tr>
<tr>
<td>hailfinder</td>
<td>-249745.31</td>
<td>-249745.31</td>
<td>-249745.31</td>
<td>-249745.31</td>
<td>-249745.31</td>
</tr>
<tr>
<td>hepar2</td>
<td>-163916.90</td>
<td>-163916.90</td>
<td>-163916.90</td>
<td>-163902.19</td>
<td>-163910.93</td>
</tr>
<tr>
<td>insurance</td>
<td>-67140.02</td>
<td>-67140.02</td>
<td>-67140.02</td>
<td>-67130.83</td>
<td>-67131.91</td>
</tr>
<tr>
<td>link</td>
<td>-1406120.95</td>
<td>-1406077.68</td>
<td>-1406077.68</td>
<td>-1400434.48</td>
<td>-1404874.90</td>
</tr>
<tr>
<td>munin1</td>
<td>-202346.99</td>
<td>-202299.56</td>
<td>-202285.69</td>
<td>-202532.15</td>
<td>-202747.71</td>
</tr>
<tr>
<td>pathfinder</td>
<td>-90839.06</td>
<td>-90839.06</td>
<td>-90839.06</td>
<td>-90841.47</td>
<td>-90839.06</td>
</tr>
<tr>
<td>pigs</td>
<td>-1668056.82</td>
<td>-1667966.85</td>
<td>-1667966.85</td>
<td>-1667913.75</td>
<td>-1667939.85</td>
</tr>
<tr>
<td>win95pts</td>
<td>-46253.89</td>
<td>-46235.01</td>
<td>-46235.01</td>
<td>-46366.03</td>
<td>-46477.00</td>
</tr>
</tbody>
</table>

Footnotes:

- HC
- MMDU
- FastCHC
Table 4.8: Detailed results for score metric calls from the second set of experiments. The results highlighted in bold are the best for the corresponding network.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Mod</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>HC</td>
<td>3</td>
<td>3361</td>
</tr>
<tr>
<td>MMHC</td>
<td>3</td>
<td>3714</td>
</tr>
<tr>
<td>FastCHC</td>
<td>3</td>
<td>1376</td>
</tr>
<tr>
<td>FastCHC</td>
<td>1</td>
<td>1761</td>
</tr>
<tr>
<td>FastCHC</td>
<td>2</td>
<td>1822</td>
</tr>
<tr>
<td>FastCHC</td>
<td>2</td>
<td>1915</td>
</tr>
<tr>
<td>alarm</td>
<td>3</td>
<td>33</td>
</tr>
<tr>
<td>andes</td>
<td>1120</td>
<td>23</td>
</tr>
<tr>
<td>barley</td>
<td>6067</td>
<td>92</td>
</tr>
<tr>
<td>BNn100p1m2</td>
<td>21665</td>
<td>13254</td>
</tr>
<tr>
<td>BNn100p1m5</td>
<td>21317</td>
<td>13273</td>
</tr>
<tr>
<td>BNn100p2m2</td>
<td>32426</td>
<td>19446</td>
</tr>
<tr>
<td>BNn100p2m5</td>
<td>27439</td>
<td>18368</td>
</tr>
<tr>
<td>BNn200p1m2</td>
<td>81764</td>
<td>49573</td>
</tr>
<tr>
<td>BNn200p1m5</td>
<td>82162</td>
<td>47591</td>
</tr>
<tr>
<td>BNn200p2m2</td>
<td>127226</td>
<td>64130</td>
</tr>
<tr>
<td>BNn200p2m5</td>
<td>107639</td>
<td>56285</td>
</tr>
<tr>
<td>child</td>
<td>898</td>
<td>2371</td>
</tr>
<tr>
<td>diabetes</td>
<td>4271</td>
<td>856508</td>
</tr>
<tr>
<td>hailfinder</td>
<td>6748</td>
<td>30775</td>
</tr>
<tr>
<td>hepar2</td>
<td>10892</td>
<td>19227</td>
</tr>
<tr>
<td>insurance</td>
<td>2254</td>
<td>4495</td>
</tr>
<tr>
<td>link</td>
<td>13003</td>
<td>314895</td>
</tr>
<tr>
<td>mildew</td>
<td>2474</td>
<td>3456</td>
</tr>
<tr>
<td>munin1</td>
<td>99479</td>
<td>215123</td>
</tr>
<tr>
<td>pathfinder</td>
<td>39051</td>
<td>10258</td>
</tr>
<tr>
<td>pigs</td>
<td>535910</td>
<td>6740754</td>
</tr>
<tr>
<td>water</td>
<td>3021</td>
<td>1696</td>
</tr>
<tr>
<td>win95pts</td>
<td>19034</td>
<td>19205</td>
</tr>
</tbody>
</table>

The results highlighted in bold are the best for the corresponding network.

Table 4.8: Detailed results for score metric calls from the second set of experiments.
Chapter 5

Capturing Class Interaction in Multi-dimensional Classification

Each multi-dimensional classification technique has its own weaknesses and strengths. Label transformation methods are efficient but lack a solid interpretation for the interaction among classes, resorting to heuristic approaches such as majority voting. On the other hand, Bayesian classifiers struggle finding an efficient representation to combine every class and attribute, ending with complex models and costly inference processes. In this chapter, we present a new framework for multi-dimensional classification that captures the pairwise interactions between class variables in an unique way by using the best of both worlds. We propose a two stage algorithm that encodes pairwise interactions between the classes using label power sets with single-class supervised models and then combines the individual predictions by inducing a Markov random field that is subsequently used for multi-dimensional inference. Our proposal leads to a general framework supporting a wide range of base classifiers in the first stage as well as different inference methods in the second one. We will define the basic framework and its main properties, as well as strategies for ensuring the scalability for this approach. We include a detailed experimental evaluation based on a range of publicly available databases, where we analyse the overall performance of the framework and the scalability.
5. FACTOR-BASED MULTI-DIMENSIONAL CLASSIFIER

strategies proposed.†

5.1 Introduction

Multi-dimensional classification is often considered as a non-standard classification problem [Bielza et al., 2011], where the "standard" approaches refer to those applied to supervised classification in which a single-class variable is targeted. Problems with more than single-class variables such as multi-label and multi-dimensional ones are seen as broader and more complex to solve and usually build on existing single-class solutions to extend its definition to the multi-dimensional setting. By adapting single-class methods such as in binary relevance or label transformation methods we usually have to resort to heuristic and less robust representations of the data. On the other hand, probabilistic classifiers such as Bayesian network classifiers with more than a single-class variable do not retain the same restrictions and inference becomes a complex and costly task. Our approach proposes a novel hybrid representation of the problem, using both a divide and conquer strategy to transform the multi-dimensional setting into a many single-class problems and a robust probabilistic model for the distributions among classes that focuses on efficient inference.

The framework is composed by two different stages and can be positioned between the transformation-based classifiers and the family of multi-dimensional probabilistic graphical models (PGM-based classifiers) 1. In the first stage we follow a transformation-based approach and learn a single-class classifier for each pair of class variables in the domain. The framework does not prescribe a particular type of classifier, but only requires that the outcome of the classifier should be a weighted distribution over the (compound) class values. Standard probabilistic classifiers meet this criterion. In the second stage a Markov random field (MRF) is constructed based on the results from the first stage. The MRF models the dependencies between the class variables, and thereby connects the framework to the class of multi-dimensional PGM-based classifiers. Subsequent classification is achieved by performing inference in the induced MRF.

†This chapter is based on the following papers: Arias et al. [2014, 2016]

1We refer to PGM based classifiers in order to accommodate a wider range of graphical models [Guo & Gu, 2011] in addition to the more common approaches based on Bayesian networks.
We describe this new approach as a framework because it would not be completed without the selection of a base classifier to be included in the first stage. This configuration is very flexible as it presents many free configuration options to choose. The most relevant is the selection of the classifier algorithm to be applied in the first stage. Additionally, preprocessing can be done separately for each single-class classifier, allowing one to take advantage of state-of-the-art algorithms for supervised discretization and feature selection. Finally, different types of MRF-based inference algorithms can be used for the subsequent classification, the choice of method can therefore depend on the complexity of the model (exact or approximate inference) and the score to be maximized (calculation of marginal probabilities or a most probable explanation).

Furthermore, the general method scales with the computational resources available, as the single-class classifiers in the first stage can be learned independently and chosen to be scalable as well [Madsen et al., 2014]. Nevertheless, naively dealing with all pairs of class variables imposes a strong limitation on the number of class variables the algorithm can handle. We therefore outline strategies for scaling up the algorithm to datasets having a large number of class variables. Experiments carried out over a collection of benchmark datasets confirm the feasibility of the approach and show that the proposed method significantly outperforms or is comparable to the straw-men methods included in the comparison.

We would like to remark that the proposal does not fit into the so-called pair-wise multi-label approach, which interprets the labels of the instances as preferences and whose goal is to obtain a ranking among the labels [Hüllermeier et al., 2008] and not a joint configuration of class values. Furthermore, although our approach trains several classifiers in the first phase, it does not strictly fall into the class of ensemble methods either, as each single-class classifier only provides a partial answer to the multi-dimensional problem.

5.2 A General Framework for Multi-Dimensional Classification

The proposed framework falls between the family of MBCs classifiers [Bielza et al., 2011; de Waal & van der Gaag, 2007; van der Gaag & de Waal, 2006] and label transformation-based classifiers [Tsoumakas & Katakis, 2007], by combining the
results from a collection of classifiers learned for each possible pairwise interaction between the class variables. The legal types of classifiers are restricted to those classifiers that for a given instance $\mathbf{x}$ can provide a factor $\phi_{ij} | \mathbf{x} : \Omega_{Y_i} \times \Omega_{Y_j} \rightarrow \mathbb{R}^+$ for each class pair $Y_i$ and $Y_j$ such that the greater the value the higher the dependence between the class states. We shall refer to these classifiers as base classifiers. Given the class-pair factors produced by the base classifiers for an instance $\mathbf{x}$, we pose the problem of doing multi-dimensional classification as an inference problem in the pairwise Markov random field (MRF) induced by the factors. Specifically, for $l$ class variables, the pairwise Markov random field specified by the factors $\phi_{ij} | \mathbf{x}$ defines a joint distribution

$$P(Y_1, \ldots, Y_l | \mathbf{x}) = \frac{1}{Z} \prod_{i \neq j} \phi_{ij} | \mathbf{x}(Y_i, Y_j),$$  \hspace{1cm} (5.1)

where

$$Z = \sum_{Y_1, \ldots, Y_l} \prod_{i \neq j} \phi_{ij} | \mathbf{x}(Y_i, Y_j)$$  \hspace{1cm} (5.2)

is the partition function.$^1$ Based on this specification, we perform classification by doing inference in the MRF model. Thus, for global accuracy we look for the most probable explanation (MPE) in the MRF

$$\hat{y} = \arg \max_{y=(y_1, \ldots, y_m)} \frac{1}{Z} \prod_{i \neq j} \phi_{ij} | \mathbf{x}(Y_i, Y_j)$$

$$= \arg \max_{y=(y_1, \ldots, y_m)} \prod_{i \neq j} \phi_{ij} | \mathbf{x}(Y_i, Y_j),$$  \hspace{1cm} (5.3)

and for Hamming accuracy we consider the most probable class variable configurations separately:

$$\hat{y}_k = \arg \max_{y_k} \sum_{Y_1, \ldots, Y_l \neq k} \prod_{i \neq j} \phi_{ij} | \mathbf{x}(Y_i, Y_j).$$  \hspace{1cm} (5.4)

In summary, given a collection of base classifiers, multi-dimensional classification of an instance $\mathbf{x}$ consists of two steps:

$^1$We abuse notation slightly and use variable summation to denote summation over states of a variable.
1. For each pair of class variables, $Y_i$ and $Y_j$, employ the corresponding base classifier to find a factor $\phi_{ij|x}$ that for each configuration $(y_i, y_j)$ encodes the dependence between $y_i$ and $y_j$ for instance $x$.\(^1\)

2. Recover the configuration of classes for $x$ by performing inference in the pairwise Markov random field defined by the factors $\phi_{ij|x}$ found in step 1.

The overall framework is flexible in the sense that it can accommodate several different types of base classifiers (e.g., probabilistic classifiers, neural networks, etc.). Hence, we say that a particular choice of base classifier instantiates the framework, and in what follows we shall refer to the instantiated framework as a \textit{factor-based multi-dimensional classifier (FMC)}. Given the context of this dissertation, we will focus on probabilistic base classifiers, and for ease of exposition we will refer to naive Bayes classifiers in our examples, even though other base classifiers can be used as well. For each pair of class variables $Y_i$ and $Y_j$ we will induce a naive Bayes classifier (NB), where the state space of the class variable is the Cartesian product of the states in $Y_i$ and $Y_j$. With this type of base classifier, the factors $\phi_{ij|x}$ in the FMC correspond to the posterior probabilities $P(Y_i, Y_j | x)$. The relationship between the NB base classifiers and the induced MRF is illustrated in Figure 5.1 for a domain with three attributes $\{X_1, X_2, X_3\}$ and three class variables $\{Y_1, Y_2, Y_3\}$.

Comparing the proposed framework to multi-dimensional Bayesian network classifiers (MBCs) [Bielza et al., 2011; Borchani et al., 2012; de Waal & van der Gaag, 2007] we see that the induced MRF plays the role of the class-subgraph in the MBC. Similarly, the feature subgraph and the bridge subgraph are captured by the base classifiers that, in addition, also allow each class pair to employ different types of preprocessing and encode different dependency structures. The proposed method also shares some similarities with the method in Read et al. [2014], which performs multi-dimensional classification by constructing super-classes based on a partitioning of the class variables. This partitioning is based on (indirect) measurements of conditional class dependencies [Zhang & Zhang, 2010], and for each identified subset a distinct multi-dimensional classifier is learned. A consequence of the class partitioning is that class variables in different super-classes are as-

\(^1\)Clearly, considering all pairs of base classifiers can be computationally intensive, and strategies for learning base classifiers for only a subset of the possible class variable pairs are explored in Section 5.3.
Figure 5.1: FMC structure example with naive Bayes as base classifier. The joint probabilities calculated using the NB base classifiers serve as factors in the MRF, which is in turn used for finding the class variables.
sumed independent\(^1\). In comparison, dependencies between class variables in the FMC are encoded in the Markov random field, which is directly obtained based on pairwise class interactions and provides a flexible way of encoding the dependences without having to assume (marginal) independences between classes. In the FMC framework, the Markov random field has the role of a secondary structure used for combining pairwise class predictions. A related approach is explored in Wang et al. [2014], where a Bayesian network structure is used for capturing dependencies among the classes. The Bayesian network is learned directly from the class variables in the data set, and is subsequently used to combine previously obtained class estimates (found using existing multi-dimensional classifiers).

### 5.2.1 Preprocessing Capabilities

Learning an FMC is, in principle, equivalent to learning independent base classifiers for each class variable pair. Not only does this support scalable learning methods to be devised, but it also allows for different preprocessing techniques to be deployed for the different base classifiers; this includes discretization of continuous variables as well as feature subset selection (FSS). Consequently, the discretization of a continuous feature variable \(X_i\) appearing in two different base classifiers can produce different outcomes if one, e.g., employs a supervised discretization technique like MDL-based discretization [Fayyad & Irani, 1993]. Similarly, different feature subsets can also be selected for each base classifiers using a supervised method such as the Correlation-Based-Feature subset selection algorithm (CFS) [Hall, 1999]. This enables the FMC to take a more fine-grained class context into account when performing preprocessing as compared to other MBCs approaches that rely on a single fixed set of feature variables. Furthermore, since preprocessing is only applied at the level of the base classifiers, standard supervised preprocessing techniques, such as the ones previously mentioned, can be directly employed (a property that is typically not available for other multi-dimensional classifiers).

We have analysed to what extent the flexibility of applying an individual preprocessing process to each base classifier is exploited by the learned classifiers. Table 5.1 shows the results for running supervised MDL discretization and CFS selection for the datasets analysed in Section 5.4. CFS is configured with best

\(^1\)To make the method more robust to variations in the training data, in particular given the relation to the identified dependency structures/class partitioning, Read et al. [2014] also considers ensembles of super-class classifiers.
5. FACTOR-BASED MULTI-DIMENSIONAL CLASSIFIER

Table 5.1: Effect of the individual discretization feature selection techniques applied in a FMC classifier.

<table>
<thead>
<tr>
<th></th>
<th>birds</th>
<th>CAL500</th>
<th>CLEF14</th>
<th>emotions</th>
<th>enron</th>
<th>genbase</th>
<th>medical</th>
<th>scene</th>
<th>tmc2007</th>
<th>yeast</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of class pairs</td>
<td>171</td>
<td>15051</td>
<td>36</td>
<td>15</td>
<td>1378</td>
<td>351</td>
<td>990</td>
<td>15</td>
<td>231</td>
<td>91</td>
</tr>
<tr>
<td>Avg. no. of disc</td>
<td>4.78</td>
<td>27.36</td>
<td>33.25</td>
<td>6.36</td>
<td>-</td>
<td>-</td>
<td>12.30</td>
<td>-</td>
<td>15.13</td>
<td>-</td>
</tr>
<tr>
<td>Avg. no. of FSS</td>
<td>116</td>
<td>1171</td>
<td>36</td>
<td>15</td>
<td>1264</td>
<td>344</td>
<td>818</td>
<td>15</td>
<td>231</td>
<td>88</td>
</tr>
</tbody>
</table>

The data sets enron, genbase, and medical do not contain any continuous variables so discretization is not performed. Note that the results in the table are independent of the particular choice of base classifier.

first search procedure for these experiments. The first row shows the total number of class pairs for each dataset, which corresponds with the number of base classifiers to be learned. An independent execution of each the preprocessing methods will be conducted for each of these datasets. The second row shows the average number of different discretizations for the continuous variables produced by the discretization algorithm for each base classifier. The third row shows the number of different feature subsets selected for each base classifier on each dataset. Both values have been averaged among all base classifiers for each dataset. From the table we see that FMC can indeed obtain different discretizations and feature subset selections when multiple runs of the preprocessing pipeline are performed for each base classifier. It seems that the effect of this process varies among the different datasets, where comparatively minor variation is seen in the number of different discretizations for the datasets birds and emotions, whereas larger variation is seen for the remaining data sets. Regarding FSS, with the exception of the CAL500 dataset, we see that we almost always obtain a different feature subset for each base classifier.

5.3 Scalable Learning and Inference

We can evaluate the computational complexity of the FMC algorithm separately for each stage of the algorithm:

- The computational complexity of both learning and inference using all pairwise base classifiers is quadratic given the number of class variables. Con-
cretely, if we consider all the class pairs, we have a total of $l(l - 1)/2$ base classifiers. This stage of the algorithm is also influenced by the complexity of the base classifier, $T(n)$, which is dominated by the number of predictive attributes. The complete complexity of the first stage is given by $\mathcal{O}(l^2)\cdot T(n)$. For instance, learning a naive Bayes classifier is linear given the number of predictive attributes $n$ and so the overall complexity of the first stage of the algorithm would be $\mathcal{O}(l^2n)$. However, employing other base classifiers may increase the complexity of this stage of the algorithm.

- Performing exact inference in the induced MRF is NP-hard [Dagum & Luby, 1993; Smyth, 1997]. The complexity of inference in a fully connected graph is exponential in the number of class variables, making inference infeasible for domains with even a moderate number of class variables. The computational complexity is, however, independent of the number of feature variables. Since the number of feature variables is often far larger than the number of class variables (see Table 5.2), in practice we have observed that the computation time used by the two stages of the algorithm are comparable. This is also confirmed by the experimental results in Section 5.4.

One effective strategy to reduce the complexity of the first stage is to exploit that the overall learning setting is embarrassingly parallel. We can easily see that most types of standard base classifiers can be learned independently, although one could envision complex classifiers that share substructures across class variable pairs. For the second stage of the algorithm, one might consider a suitable approximate inference algorithm when exact inference is not feasible.

However, even when using a parallel computing architecture and approximate inference algorithms, the dimensionality of the pairwise interaction between a moderate number of classes can increase the complexity of both stages beyond tractable limits. One immediate approach to overcome this difficulty is to only consider a restricted subset of class variable pairs for which base classifiers should be learned. This strategy will not only reduce the complexity of the first stage of the framework, but will also reduce the complexity of the resulting MRF model. In addition, it is possible that selecting a subset of class pairs may improve the quality of the results, as pruning edges from the MRF model may reduce the noise introduced by edges representing spurious class dependencies.

A straightforward strategy for selecting this candidate subset of class pairs
5. FACTOR-BASED MULTI-DIMENSIONAL CLASSIFIER

could be to first measure the dependence/affinity between every class pair and then make a search-based selection. We propose two strategies that are based on measuring the empirical mutual information between all class variable pairs, see 2.15.

Our first proposal starts building a maximum spanning tree using the Chow-Liu algorithm [Chow & Liu, 1968] over the class variables. This will be referred to as the CLtree algorithm in the remainder of the chapter. This approach reduces the complexity of the first learning stage to $O(kl \cdot T(n))$, as the tree will always be composed of $l - 1$ edges. In addition, the resulting MRF is guaranteed to be connected.

Another strategy is to greedily select (according to the mutual information) the best subset of $k$ class pairs with $k$ being proportional to the number of classes $l$ in the dataset. This latter strategy, referred to as best$k$, would learn more complex models than the CLtree approach while still retaining linear computational complexity $O(kl \cdot T(n))$ for the learning stage. However, this algorithm does not guarantees a connected graph and some classes will not have any dependencies; the algorithm can deal with this by learning a single-class classifier for each of these singleton variables and then set up the obtained distributions as marginal node potentials of the disconnected variables in the MRF. Both approaches will improve the scalability of the second stage as the resulting model will be less denser than the fully connected one, although still exponential.

The main drawback of this approach is that the empirical mutual information that guides the selection strategies only captures marginal dependencies among a pair of classes. This could lead to unintentionally ignoring important dependencies, and thus fail to learn and include crucial base classifiers along with the corresponding edges in the MRF. For that reason, we propose another strategy in which we first compute the Markov blankets (MBs) for each class over the space of class variables and then include all the dependencies in the MRF model. We have chosen the HITON-MB [Aliferis et al., 2003] algorithm to learn the MBs of the classes from the data, as it is a well known algorithm that has been used successfully in related works [Borchani et al., 2012], for example, to learn the class subgraph of an MBC.

The HITON-MB algorithm computes the Markov blanket for a given variable by evaluating several candidate subsets of variables by using a $G^2$ independence test. This algorithm must be parameterized by fixing a maximum number of
elements to be tested in each possible subset of dependent variables; in the experiments, we have selected a maximum subset size of 3 as this value has shown good performance in the literature [Aliferis et al., 2003]. The new search procedure starts by running HITON-MB for each class variable and learning its MB. Next, all class pairs appearing in the resulting MBs are included in the model. This strategy, which we will refer to as hiton, selects a variable number of pairs given the dependencies found in the dataset and not according to a fixed-size set or hyperparameter, so the scalability gain will vary depending on the problem. The resulting models are therefore expected to obtain superior results at the expense of efficiency.

Lastly, we propose a different learning strategy in which we perform a forward greedy search algorithm guided by the BDeu scoring metric for directed graphical models. A consequence of the search procedure is that we will obtain a directed structure over the class variables, which must be translated into an undirected structure. A trivial yet effective strategy is to drop the directions on the arcs. Since the BDeu metric is decomposable, we can efficiently compute the score of a model as the sum of the local scores for the different nodes, which, in turn, only depends on the node in question and its parent set. We will refer to this last technique as FSBDeu, which is outlined in Algorithm 5.1. The process can be separated into three different phases guided by heuristic strategies with the purpose of reducing the number of evaluations and to constrain the problem to enable a scalable search.

1. (Lines 1 to 3) First, the algorithm defines a topological ordering \( \sigma \) over the class variables by computing the score difference between the marginal BDeu score and the conditional BDeu score for each class pair; the position of a variable in the ordering is determined by the sum of the score differences computed for all class pairs in which the variable is involved. This step contains most of the computational burden of the procedure as it requires a quadratic number of evaluations given the number of classes.

2. (Lines 4 to 11) Afterwards, the candidate variable pairs are generated according to the ordering \( \sigma \): For each pair of classes \( Y_i \) and \( Y_j \) the direction of the arc is established by the relative position of \( Y_i \) and \( Y_j \) in \( \sigma \) with the preceding variable becoming the tail of the arc. This ordering guarantees that there will be no directed cycles in the resulting model even if all possible
5. FACTOR-BASED MULTI-DIMENSIONAL CLASSIFIER

pairs are added.

3. (Lines 12 to 20) Finally, a greedy forward search is performed: The pairs are sorted by their score and are tested once following the computed order. A pair \((Y_i, Y_j)\) is introduced in the model if the score difference resulting from adding \(Y_i\) to \(\Pi_{Y_j}\) is positive. When updating the parent sets, the corresponding BDeu scores are updated as well. The final structure is obtained by ignoring the directions of the arcs from the resulting model (notice that the parent sets are only considered to measure BDeu score).

5.4 Experimental Evaluation

In the first part of this section we evaluate the scalability proposals described in the previous section, in order to analyse the introduced scalability improvements and how the quality varies compared to the full pairwise framework. Our objective is to discover which of the proposed strategies entails the most significant improvements regarding the trade-off between efficiency and quality. Secondly, we perform a comparison between our selected models and a collection of representative state-of-the-art multi-dimensional classifiers taken from the literature.

5.4.1 Experimental Set up

We evaluate our approach using different instantiations of the proposed framework by taking two well-known Bayesian network classifiers as base classifiers: naive Bayes (NB), being a simple classifier with light computational and memory requirements, and A1DE, which is a more expressive classifier but more computationally intensive at the same time.

We have built a prototype of the proposed framework which consists of two different implementations for each of the algorithm stages. The first stage, learning the pairwise base classifiers, has been implemented using the Mulan [Tsoumakas et al., 2010] library for multi-dimensional dataset management as well as the Weka library [Hall et al., 2009] for learning. For the second stage we use the UGM\(^1\) Matlab package for performing inference over the induced MRF. The experiments

\[^1\text{http://www.di.ens.fr/~mschmidt/Software/UGM.html}\]
Algorithm 5.1: Forward greedy search with BDeu score model selection for FMC.

**Input:** $D_Y$: Projection of the training data over classes $Y = \{Y_1, \ldots, Y_l\}$

**Output:** A collection $P$ of selected class pairs variables $(Y_i, Y_j)$

//Compute the score difference for each class pair.
1 for each $Y_i, Y_j \in Y, j > i$ compute $\text{diff}_{Y_i,Y_j} = BDeu(Y_j | Y_i) - BDeu(Y_j | \emptyset)$;

//Establish a topological ordering among the classes according to the sum of all class pair scores in which the class is involved
2 for each $Y_i \in Y$ compute $\text{sumDiff}_{Y_i} = \sum_{j=1, j \neq i}^{l} (\text{diff}_{Y_i,Y_j})$;
3 $\sigma \leftarrow \text{order the variables } Y_i \in Y \text{ by maximizing } \text{sumDiff}_{Y_i}$;

//Create the candidate pairs according to the topological ordering
4 $P \leftarrow \emptyset$;
5 for $Y_i, Y_j \in Y, j > i$ do
6     if $\sigma_{Y_i} < \sigma_{Y_j}$ then
7         Add $(Y_i, Y_j)$ to $P$
8     else
9         Add $(Y_j, Y_i)$ to $P$
10    end
11 end

//Sort the pairs according to the score differences
12 $P \leftarrow \text{order each } (Y_i, Y_j) \in P \text{ by maximizing } \text{diff}_{Y_i,Y_j}$;

//Initialize empty parent sets for each class
13 for $Y_i \in Y$: $\Pi(Y_i) \leftarrow \emptyset$;

//Perform a forward search over the ordered pairs
14 for $(Y_i, Y_j) \in P$ do
15     if $BDeu(Y_j, | \Pi(Y_j) \cup \{Y_i\}) - BDeu(Y_j, | \Pi(Y_j)) > 0$ then
16         $\Pi(Y_j) \leftarrow \Pi(Y_j) \cup \{Y_i\}$
17     else
18         Remove $(Y_i, Y_j)$ from $P$
19     end
20 end
21 return $P$
were conducted on a dedicated Linux server with a Pentium Xeon 3.0 Ghz processor and 16GB of RAM.

We use the two metrics described in Section 2.6, Hamming accuracy \(H_{\text{acc}}\) and global accuracy \(\text{acc}\) as performance indicators. By using both measures we can provide a clear overview of the behaviour for each classifier as well as capture the difficulty of the problem. We will consider \(H_{\text{acc}}\) as an easier metric to optimize than \(\text{acc}\) especially for complex domains, so we expect the latter to be a more difficult criterion to optimise.

The experiments have been carried out using a collection of publicly available datasets, most of them taken from the Mulan repository\(^1\). We have selected datasets with a moderate number of classes and attributes, as it is infeasible to measure global accuracy for domains with a large number of classes. Additionally, we have included the CLEF14 dataset in the experiments\(^2\). This dataset comes from a real world challenge in the field of computer vision [Martinez-Gomez et al., 2014] and contains seven binary classes as well as one multi-dimensional variable with 10 states. The characteristics of the datasets used in the experiments can be found in Table 5.2.

<table>
<thead>
<tr>
<th>Database</th>
<th>Classes</th>
<th>Features</th>
<th>Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Birds</td>
<td>19</td>
<td>260</td>
<td>645</td>
</tr>
<tr>
<td>CAL500</td>
<td>174</td>
<td>68</td>
<td>502</td>
</tr>
<tr>
<td>Emotions</td>
<td>6</td>
<td>72</td>
<td>593</td>
</tr>
<tr>
<td>Enron</td>
<td>53</td>
<td>1001</td>
<td>1702</td>
</tr>
<tr>
<td>Genbase</td>
<td>27</td>
<td>1186</td>
<td>662</td>
</tr>
<tr>
<td>Medical</td>
<td>45</td>
<td>1449</td>
<td>978</td>
</tr>
<tr>
<td>Scene</td>
<td>6</td>
<td>294</td>
<td>2407</td>
</tr>
<tr>
<td>TMC2007</td>
<td>22</td>
<td>500</td>
<td>28596</td>
</tr>
<tr>
<td>Yeast</td>
<td>14</td>
<td>103</td>
<td>2417</td>
</tr>
<tr>
<td><strong>CLEF14</strong></td>
<td>7x2/1x10</td>
<td>360</td>
<td>9500</td>
</tr>
</tbody>
</table>

Table 5.2: Multi-label/Multi-dimensional datasets used in the experiments.

Since the standard version of the A1DE classifier cannot handle numeric variables we have preprocessed the datasets by discretizing the continuous variables.\(^3\)

---

\(^1\)http://mulan.sourceforge.net/datasets.html
\(^2\)http://www.rovit.ua.es/dataset/vidrilo.html
\(^3\)We have conducted preliminary experiments using NB with Gaussian distributions as well as variants of the A1DE classifier capable of dealing with continuous variables [Flores
This has been done by applying the procedure mentioned in Section 5.2.1, where we take advantage of the structure of the model and apply supervised discretization and posterior feature subset selection to each individual base classifier. In particular, we have preprocessed the data by discretizing the numerical features using MDL and selecting features with CFS.

### 5.4.2 Comparison Between Pruning Strategies

In Tables 5.3 and 5.4 we report on the results regarding \( \text{acc} \) and \( H_{\text{acc}} \) respectively, for the full pairwise model and for each of the scalability strategies. The results have been obtained by performing a 10-fold cross validation for each of the datasets; the results highlighted in bold are the best for the corresponding datasets. At first sight, we can observe that there is a clear difference between configurations using each base classifiers, with A1DE obtaining superior results over NB as expected. However, if we compare the approaches among themselves when using the same base classifier we can not observe a significant difference in quality between the full pairwise model and the pruned ones regarding any of the accuracy measures. To extend our comparison, we have performed statistical tests for both global accuracy and Hamming accuracy according to the procedure described by Demšar [2006].

A Friedman test [Friedman, 1940] comparing all the pruning approaches using A1DE as base classifier does not reject the null hypothesis that all classifiers are equivalent regarding global accuracy with \( p \)-value = 0.1046 and a 5% significance level. However, in the case of Hamming accuracy the same test is rejected with \( p \)-value = 0.0108. A post-hoc test with a 5% significance level using the Holm correction [Holm, 1979] shows that the \( CLtree \) approach obtains statistically worse results in case of Hamming accuracy when compared to the other approaches with \( p \)-value = 0.0095. Figure 5.2 shows the distribution of the mean ranks computed for the Friedman test for each of the approaches, confirming that the \( CLtree \) has the overall worst performance and that \( FSBDeu, hiton \), and the full pairwise model have comparable results.

We conclude that most of the pruning approaches obtain results that are comparable to the full pairwise model, especially in the case of global accuracy. Next
Table 5.3: Results on global accuracy for various pruning strategies in FMC.
Grouped by the choice of base classifier, A1DE (above) and NB (below). Highlighted results show the best result for the corresponding dataset. Notice that owing to the large number of class variables in the CAL500 dataset, it is infeasible to provide accurate predictions of the full vector and thus the exact accuracy is very low.

Table 5.4: Results on Hamming accuracy for various pruning strategies in FMC. Highlighted results show the best result for the corresponding dataset.
Figure 5.2: Average rank obtained for each pruning strategy in FMC. Divided by score metric, acc (above) and $H_{acc}$ (below). The values correspond to the mean rank among each dataset as computed for the Friedman test.
we will study these results in terms of efficiency and general scalability. Figures 5.3 and 5.4 show the number of pairs selected for each method and dataset. In these plots we see that the full pairwise approach builds the largest models, and thus it is more inefficient in both stages of the algorithm, especially for datasets with a high number of classes. On the other hand, the Chow-Liu approach, CLtree, builds the least dense models, followed by FSBDeu, bestk with $k = 2l$, and hiton. In order to measure the overall complexity of each pruning strategy we must take into account the size of the resulting model as well as the complexity of the pruning procedure itself. In this case, hiton is by far the most costly strategy, given the exponential nature of the HITON-MB algorithm. On the other hand, bestk, CLtree and FSBDeu consist on greedy search and have linear complexity given the number of class pairs.

As a conclusion, we propose the FSBDeu approach as the most promising strategy as it combines both an efficient learning scheme and the ability to learn less dense models when compared to hiton and bestk. In addition, this strategy does not define any additional hyperparameters so it is more intuitive to use.

We would like to highlight an important detail regarding the results for the CLEF14 dataset. If we look at the results obtained for the acc score metric in Table 5.3 we can see that the CLtree approach obtains the best results for this dataset followed by FSBDeu, whereas the full pairwise model obtains the worst ones. This behaviour reflects that the underlying dependence model among the classes in this dataset can be accurately described by a tree structure, thus fully connected dependency networks risk the potential addition of noise in the prediction. In this case, sparser models such as the ones induced by FSBDeu are a better representation for this dataset, so the strategy is improving efficiency and predictive power at the same time.

We performed an additional experiment to compare the complexity of both stages of the algorithm by measuring their contributions to the overall runtime. Figure 5.5 shows the runtime breakdown for the full pairwise, FSBDeu and CLtree approaches for all the datasets: The first column corresponds to the training time for the first stage of the algorithm; the second column shows the time for classifying the test instances using all base classifiers; finally the third column corresponds to the time used by the second stage, where inference is performed on the MRF for each test instance (see also the discussion below). All results are expressed in seconds with logarithmic scale and have been obtained by averaging the runtime.
Figure 5.3: Number of pairs selected by each pruning strategy in FMC.

Figure 5.4: Proportion between selected pairs and classes in the dataset for each pruning strategy in FMC.
5. FACTOR-BASED MULTI-DIMENSIONAL CLASSIFIER

Figure 5.5: Breakdown of the runtime in seconds for the different stages in the FMC framework.
based on 10-fold cross validation and using A1DE as base classifier. As we can observe, in practice, the first stage is less efficient than the second one, especially when a pruning strategy is being used. There are only two exceptions to this observation: emotions, which is a very small dataset and the training time is therefore correspondingly low, and CAL500, which features a large number of labels and instances, thus making the inference on the MRF more complex. It should be noted that the tests are based on the A1DE classifier, and the observed differences would therefore be reduced by using the NB instead.

Additionally, we could consider different inference algorithms to obtain the final predictions from the MRF. The results reported to this point have been obtained using approximate inference. Specifically, we have used the loopy belief propagation algorithm [Murphy et al., 1999], which is included in the aforementioned Matlab package. Performing approximate inference over a MRF introduces a significant improvement in the computational requirements of the algorithm at the expense of a potential detriment of the quality of the predictions.

For that reason, we have conducted additional experiments running exact inference for some of the smaller datasets and pruned models. Table 5.5 shows the results obtained for accuracy using the full pairwise model as well as the FSBDeu and CLtree pruning algorithms for both approximate and exact inference. From the results we see that there is no clear difference when it comes to the final predictions using either exact or approximate inference. However, when it comes to the complexity of the procedures the differences are significant. Figure 5.6 shows the runtime results for the previous experiment, revealing that, as expected, the approximate inference algorithm is many times faster than the exact approach. Notice that for this application we are not looking to retrieve accurate probability distributions, as we are just computing the MAP configuration for each one of the individual classes.

5.4.3 Comparison with State-of-the Art Classifiers

In order to contrast the performance of our proposal we have replicated our experimental evaluation to include three well known state-of-the art multi-dimensional classifiers, all of which have public implementations available: Binary-Relevance (BR), Ensembles of Classifier Chains (ECC), and RAkEL. All of them must be instantiated with a particular base classifier as well, for which we have selected the
5. FACTOR-BASED MULTI-DIMENSIONAL CLASSIFIER

<table>
<thead>
<tr>
<th>Dataset</th>
<th>full-A1DE</th>
<th>FSBDeu</th>
<th>CLtree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LBP exact</td>
<td>LBP exact</td>
<td>LBP exact</td>
</tr>
<tr>
<td>birds</td>
<td>0.9521</td>
<td>0.9521</td>
<td>0.9490</td>
</tr>
<tr>
<td></td>
<td>0.9490</td>
<td>0.9488</td>
<td>0.9488</td>
</tr>
<tr>
<td>CLEF14</td>
<td>0.8548</td>
<td>0.8543</td>
<td>0.8647</td>
</tr>
<tr>
<td></td>
<td>0.8579</td>
<td>0.8668</td>
<td>0.8520</td>
</tr>
<tr>
<td>emotions</td>
<td>0.7960</td>
<td>0.7960</td>
<td>0.7987</td>
</tr>
<tr>
<td></td>
<td>0.7987</td>
<td>0.7892</td>
<td>0.7892</td>
</tr>
<tr>
<td>enron</td>
<td>0.9511</td>
<td>-</td>
<td>0.9508</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>0.9493</td>
<td>0.9493</td>
</tr>
<tr>
<td>genbase</td>
<td>0.9986</td>
<td>0.9982</td>
<td>0.9985</td>
</tr>
<tr>
<td></td>
<td>0.9985</td>
<td>0.9983</td>
<td>0.9983</td>
</tr>
<tr>
<td>medical</td>
<td>0.9892</td>
<td>-</td>
<td>0.9888</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>0.9909</td>
<td>0.9887</td>
</tr>
<tr>
<td>scene</td>
<td>0.9117</td>
<td>0.9114</td>
<td>0.9117</td>
</tr>
<tr>
<td></td>
<td>0.9114</td>
<td>0.8985</td>
<td>0.8985</td>
</tr>
<tr>
<td>tmc2007</td>
<td>0.9329</td>
<td>0.9323</td>
<td>0.9329</td>
</tr>
<tr>
<td></td>
<td>0.9329</td>
<td>0.9317</td>
<td>0.9303</td>
</tr>
<tr>
<td>yeast</td>
<td>0.7939</td>
<td>0.7938</td>
<td>0.7937</td>
</tr>
<tr>
<td></td>
<td>0.7937</td>
<td>0.7872</td>
<td>0.7872</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>full-A1DE</th>
<th>FSBDeu</th>
<th>CLtree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LBP exact</td>
<td>LBP exact</td>
<td>LBP exact</td>
</tr>
<tr>
<td>birds</td>
<td>0.4968</td>
<td>0.4968</td>
<td>0.4877</td>
</tr>
<tr>
<td></td>
<td>0.4877</td>
<td>0.4781</td>
<td>0.4781</td>
</tr>
<tr>
<td>CLEF14</td>
<td>0.3701</td>
<td>0.3699</td>
<td>0.4134</td>
</tr>
<tr>
<td></td>
<td>0.4134</td>
<td>0.4257</td>
<td>0.3483</td>
</tr>
<tr>
<td>emotions</td>
<td>0.3017</td>
<td>0.3017</td>
<td>0.3253</td>
</tr>
<tr>
<td></td>
<td>0.3253</td>
<td>0.2900</td>
<td>0.2900</td>
</tr>
<tr>
<td>enron</td>
<td>0.1269</td>
<td>-</td>
<td>0.1263</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>0.1263</td>
<td>0.1263</td>
</tr>
<tr>
<td>genbase</td>
<td>0.9623</td>
<td>0.9501</td>
<td>0.9608</td>
</tr>
<tr>
<td></td>
<td>0.9608</td>
<td>0.9578</td>
<td>0.9578</td>
</tr>
<tr>
<td>medical</td>
<td>0.6431</td>
<td>-</td>
<td>0.6339</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>0.6360</td>
<td>0.6360</td>
</tr>
<tr>
<td>scene</td>
<td>0.6581</td>
<td>0.6568</td>
<td>0.6581</td>
</tr>
<tr>
<td></td>
<td>0.6568</td>
<td>0.5928</td>
<td>0.5928</td>
</tr>
<tr>
<td>tmc2007</td>
<td>0.2440</td>
<td>0.2390</td>
<td>0.2473</td>
</tr>
<tr>
<td></td>
<td>0.2430</td>
<td>0.2336</td>
<td>0.2336</td>
</tr>
<tr>
<td>yeast</td>
<td>0.1850</td>
<td>0.1850</td>
<td>0.1994</td>
</tr>
<tr>
<td></td>
<td>0.1994</td>
<td>0.1787</td>
<td>0.1787</td>
</tr>
</tbody>
</table>

Table 5.5: Comparison between exact and approximate inference in an FMC. Hamming accuracy (above) and global accuracy (below) for the selected models and datasets.

same base classifiers as for our original proposal (naive Bayes and A1DE). We have chosen the recommended hyperparameter configuration from the literature: ECC has been configured to learn a ensemble chain of 10 models, and RA$k$EL follows the recommended configuration from Tsoumakas et al. [2011] of 2$l$ models with $k = 3$, which corresponds with triplets of class combinations in the power set. We have used the open implementation from the Mulan library for the experiments.

Tables 5.6 and 5.7 show the results for global accuracy and Hamming accuracy, respectively, for each of the classifiers instantiated with both base classifiers. We have included several configurations of the FMC classifier: full pairwise, FMC, and BDeu-pruned, FMC-BDeu, for the sake of comparison. We have also included the same preprocessing pipeline described for our framework. Notice that the three straw-men methods considered are also transformation-based classifiers that can apply separate preprocessing algorithms for each base classifier. However, we have
observed that, in some domains, using feature selection decreases the performance of the multi-dimensional classifiers when using A1DE as base classifier, especially in terms of global accuracy. For the emotions and yeast datasets we have therefore replaced the results in the previous tables with the corresponding results obtained without using feature selection. This was, however, not feasible for the data sets enron, medical, and genbase, as the algorithms are not able to handle these datasets using the full set of features.

This result can be attributed to internal details of these multi-dimensional classifiers. In the case of transformation methods such as Binary Relevance or RAkEL, the final output is computed by relying only on the individual class predictions of the base classifiers. Thus, any change in accuracy of these models, either positive or negative, can result in significant variance in the prediction for the individual classes. Notice that our FMC classifier uses the computed probability distributions from the probabilistic base classifiers instead of the individual predictions, so it is less sensitive to variance in the accuracy of the base classifiers.

In general, we can observe that our proposed method obtains superior results when compared to the other state-of-the-art classifiers. To extend the comparison
5. FACTOR-BASED MULTI-DIMENSIONAL CLASSIFIER

<table>
<thead>
<tr>
<th>Classifier</th>
<th>birds</th>
<th>CAL500</th>
<th>CLEF14</th>
<th>emotions</th>
<th>enron</th>
<th>genbase</th>
<th>medical</th>
<th>scene</th>
<th>tmc2007</th>
<th>yeast</th>
</tr>
</thead>
<tbody>
<tr>
<td>full-A1DE</td>
<td>0.4968</td>
<td>0.0000</td>
<td>0.3683</td>
<td>0.3017</td>
<td>0.1269</td>
<td>0.9623</td>
<td>0.6431</td>
<td>0.6597</td>
<td>0.2340</td>
<td>0.1850</td>
</tr>
<tr>
<td>full-NB</td>
<td>0.4658</td>
<td>0.0000</td>
<td>0.1338</td>
<td>0.2831</td>
<td>0.1105</td>
<td>0.9562</td>
<td>0.6227</td>
<td>0.5052</td>
<td>0.2320</td>
<td>0.1486</td>
</tr>
<tr>
<td>FSBDeu-A1DE</td>
<td>0.4877</td>
<td>0.0000</td>
<td>0.4134</td>
<td>0.3253</td>
<td>0.1263</td>
<td>0.9608</td>
<td>0.6339</td>
<td>0.6581</td>
<td>0.2473</td>
<td>0.1994</td>
</tr>
<tr>
<td>FSBDeu-NB</td>
<td>0.4315</td>
<td>0.0000</td>
<td>0.1755</td>
<td>0.2933</td>
<td>0.1181</td>
<td>0.9562</td>
<td>0.6227</td>
<td>0.5069</td>
<td>0.2336</td>
<td>0.1684</td>
</tr>
<tr>
<td>BR-A1DE</td>
<td>0.4564</td>
<td>0.0000</td>
<td>0.2286</td>
<td>*0.2377</td>
<td>0.0000</td>
<td>0.9155</td>
<td>0.1124</td>
<td>0.5451</td>
<td>0.1350</td>
<td>0.1117</td>
</tr>
<tr>
<td>BR-NB</td>
<td>0.4348</td>
<td>0.0000</td>
<td>0.0814</td>
<td>0.2495</td>
<td>0.1093</td>
<td>0.9758</td>
<td>0.5797</td>
<td>0.3079</td>
<td>0.2113</td>
<td>0.1039</td>
</tr>
<tr>
<td>ECC-A1DE</td>
<td>0.4566</td>
<td>0.0000</td>
<td>0.1463</td>
<td>*0.2019</td>
<td>0.0000</td>
<td>0.9185</td>
<td>0.1154</td>
<td>*0.5588</td>
<td>0.1476</td>
<td>*0.1572</td>
</tr>
<tr>
<td>ECC-NB</td>
<td>0.4562</td>
<td>0.0000</td>
<td>0.1033</td>
<td>0.3001</td>
<td>0.1234</td>
<td>0.9728</td>
<td>0.6205</td>
<td>0.3403</td>
<td>0.2206</td>
<td>0.1643</td>
</tr>
<tr>
<td>RAKEL-A1DE</td>
<td>0.0312</td>
<td>0.0000</td>
<td>0.3746</td>
<td>*0.3236</td>
<td>-</td>
<td>-</td>
<td>0.0143</td>
<td>*0.5376</td>
<td>0.1412</td>
<td>*0.1270</td>
</tr>
<tr>
<td>RAKEL-NB</td>
<td>0.4564</td>
<td>0.0000</td>
<td>0.1221</td>
<td>0.2983</td>
<td>0.1051</td>
<td>0.9698</td>
<td>0.6032</td>
<td>0.4674</td>
<td>0.2127</td>
<td>0.1275</td>
</tr>
</tbody>
</table>

Table 5.6: Global accuracy for the literature classifiers and FMC. Highlighted results show the best results for the corresponding dataset. Results marked with a '*' has been obtained without using feature selection. Missing results correspond with experiments that could not be finished due to limitations in the framework.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>birds</th>
<th>CAL100</th>
<th>CLEF14</th>
<th>emotions</th>
<th>enron</th>
<th>genbase</th>
<th>medical</th>
<th>scene</th>
<th>tmc2007</th>
<th>yeast</th>
</tr>
</thead>
<tbody>
<tr>
<td>full-A1DE</td>
<td>0.9521</td>
<td>0.8635</td>
<td>0.8545</td>
<td>0.7960</td>
<td>0.9511</td>
<td>0.9986</td>
<td>0.9892</td>
<td>0.9122</td>
<td>0.9329</td>
<td>0.7939</td>
</tr>
<tr>
<td>full-NB</td>
<td>0.9449</td>
<td>0.8619</td>
<td>0.7621</td>
<td>0.7835</td>
<td>0.9309</td>
<td>0.9984</td>
<td>0.9888</td>
<td>0.8853</td>
<td>0.9302</td>
<td>0.7770</td>
</tr>
<tr>
<td>FSBDeu-A1DE</td>
<td>0.9490</td>
<td>0.8638</td>
<td>0.8647</td>
<td>0.7987</td>
<td>0.9508</td>
<td>0.9985</td>
<td>0.9888</td>
<td>0.9117</td>
<td>0.9329</td>
<td>0.7937</td>
</tr>
<tr>
<td>FSBDeu-NB</td>
<td>0.9347</td>
<td>0.8616</td>
<td>0.7798</td>
<td>0.7846</td>
<td>0.9361</td>
<td>0.9983</td>
<td>0.9884</td>
<td>0.8858</td>
<td>0.9299</td>
<td>0.7792</td>
</tr>
<tr>
<td>BR-A1DE</td>
<td>0.9472</td>
<td>0.8503</td>
<td>0.8364</td>
<td>0.7797</td>
<td>*0.9364</td>
<td>0.9965</td>
<td>0.9755</td>
<td>*0.9024</td>
<td>0.9148</td>
<td>*0.7593</td>
</tr>
<tr>
<td>BR-NB</td>
<td>0.9322</td>
<td>0.8545</td>
<td>0.7245</td>
<td>0.7796</td>
<td>0.9312</td>
<td>0.9990</td>
<td>0.9878</td>
<td>0.8470</td>
<td>0.9267</td>
<td>0.7572</td>
</tr>
<tr>
<td>ECC-A1DE</td>
<td>0.9478</td>
<td>0.8503</td>
<td>0.8047</td>
<td>0.7897</td>
<td>*0.9366</td>
<td>0.9968</td>
<td>0.9756</td>
<td>*0.9048</td>
<td>0.9167</td>
<td>*0.7727</td>
</tr>
<tr>
<td>ECC-NB</td>
<td>0.9442</td>
<td>0.8526</td>
<td>0.7432</td>
<td>0.7909</td>
<td>0.9324</td>
<td>0.9989</td>
<td>0.9888</td>
<td>0.8548</td>
<td>0.9267</td>
<td>0.7812</td>
</tr>
<tr>
<td>RAKEL-A1DE</td>
<td>0.8427</td>
<td>0.6520</td>
<td>0.8827</td>
<td>*0.8027</td>
<td>0.7308</td>
<td>0.7656</td>
<td>*0.9361</td>
<td>0.8965</td>
<td>0.9157</td>
<td>*0.7691</td>
</tr>
<tr>
<td>RAKEL-NB</td>
<td>0.9403</td>
<td>0.8620</td>
<td>0.7606</td>
<td>0.7908</td>
<td>0.9420</td>
<td>0.9989</td>
<td>0.9884</td>
<td>0.8809</td>
<td>0.9276</td>
<td>0.7684</td>
</tr>
</tbody>
</table>

Table 5.7: Hamming accuracy for the literature classifiers and FMC. Highlighted results show the best results for the corresponding dataset. Results marked with a '*' has been obtained without using feature selection.
Table 5.8: Statistical tests comparing the state-of-the-art algorithms to FMC. The table shows the ranking and \( p \)-value for global accuracy (left) and Hamming accuracy (right). Boldfaced results correspond to rejected hypotheses.

we have performed statistical tests following the same procedure employed in the previous subsection. The Friedman test with a 5% significance level rejects the hypothesis that all classifiers are equivalent with a \( p \)-value of \( 2.6848 \times 10^{-5} \) in the case of global accuracy and with a \( p \)-value of \( 1.0862 \times 10^{-5} \) for Hamming accuracy. Following this result we performed a post-hoc test using Holm’s procedure with a 5% significance level. The results can be found in Table 5.8 together with the ranking computed for the Friedman test; this test compares all methods with the approach that has the highest mean rank (FMC-full-A1DE for \( acc \) and FMC-FSBDeu-A1DE for \( H_{acc} \)) as control.

This post-hoc test reveals that both the full and pruned FMC classifier instantiated with A1DE are the best classifiers in general, obtaining a significant difference regarding their mean rank value. The classifiers are equivalent among themselves and statistically superior to the others for both \( acc \) and \( H_{acc} \).

Finally we have evaluated the scalability for each approach by measuring their execution time for the previous experiments. Figure 5.7 show the averaged runtime obtained from cross validation for the training and inference stages of each algorithm. This comparison confirms that the full FMC model is by far the most complex and inefficient model. However, when applying a pruning strategy such as FSBDeu, the execution time is comparable to the other referenced methods for almost all the evaluated domains regarding training time which is an order of magnitude higher than inference.
Figure 5.7: Comparison of execution times between FMC and state-of-the-art classifiers.
Time in seconds for training (above) and inference (below). Notice that each plot has a different scale, and that training is an order of magnitude higher than inference.
5.5 Conclusions

The FMC multi-dimensional classifier has proved to be a solid alternative to the existing state-of-the-art multi-dimensional classifiers. We have obtained relevant empirical evidence of superior results from the perspective of both performance and quality. On the other hand, the framework is based on sound theoretical properties and retains one of the most important features of other probabilistic approaches such as MBCs. The underlying probability distribution that can be built from the models can improve the efforts of model interpretation beyond the abilities of the usual heuristic approaches that are usually employed for this kind of problem.

A potential concern about the framework is its scalability. We have outlined several strategies to address this problem based on structural learning algorithms which have demonstrated to be very useful in other applications of probabilistic graphical models. The experiments show that the complexity of our approach can be drastically reduced while maintaining competitive results.

5.6 Reproducibility and Extended Results

Available code and data for further use of the algorithms presented in this chapter as well as instructions to reproduce the experiments can be found on http://simd.albacete.org/supplements/FMC.html.
Chapter 6

Distributed Learning of BNCs

Implementing scalable algorithms is key to push forward machine learning towards more complex problems. High performance computing and scaling-up existing computing platforms has been the traditional answer to large domains and datasets. However, with the advent of the big data revolution the problem has shifted towards specialized brute-force approaches such as MapReduce and scaling-out commodity computers. Such frameworks provide a restricted programming model that has made existing parallel implementations and strategies obsolete. Not every algorithm can be translated to this new paradigm or benefit from the same improvements. We must explore the portability for each family of methods in order to discover the most promising ones. In this chapter we will explore how Bayesian network classifiers fit in this distributed approach by providing a definition and implementation for this family of models in MapReduce. We will explore the scalability improvements that this framework provides both from a theoretical and empirical point of view by performing an extensive evaluation on several large scale problems found in the literature.†

6.1 Introduction

Scalability is a key feature of any machine learning technique. We constantly find problems in industry and science that can not be solved using conventional computational tools. This is especially true when dealing with real-world data in

†This chapter is based on the following papers: Arias et al. [2015a, 2017]
the context of modern technologies such as telecommunications, social networks, internet of things or even fields like genomics [Chen et al., 2014a; Manyika et al., 2011; McAfee et al., 2012].

The big data technological ecosystem allows the collection of huge quantities of information that are not always suitable to be processed. This is especially true in the case of machine learning algorithms, which have always suffered from the curse of dimensionality and are well known for requiring large amounts of computing power. In such cases, we can resort to statistical data reduction techniques or to scale up the architecture and the algorithms. The latter can be achieved thanks to new powerful technologies that have been developed in that same ecosystem.

In Chapter 3 we discussed how big data frameworks such as Apache Hadoop and Apache Spark have defined a new computing paradigm both from the architectural and algorithmical point of view. Using MapReduce on commodity hardware is aimed at scalability on multiple dimensions such as size of the data as well as resiliency and speed. Practitioners solve problems in which data is ingested and delivered at different rates depending on the business objectives and must employ tools that allow keeps that rate during the whole process. Such flexibility is present even at the hardware level, with platforms such as Cloud Computing [Armbrust et al., 2010] that provide elastic architectures where computational resources can be added on demand.

We must adopt this new definition of scalability in machine learning as well. Scalable algorithms that are aimed at big data must define elastic learning and inference processes to fit into the same data processing pipeline along with every piece of technology. This has been supported by the community by actively releasing libraries and frameworks in these programming paradigms, following the same philosophy and engineering principles. Among such libraries we can highlight the early Apache Mahout for Hadoop [Owen et al., 2011] and Apache Spark MLlib [Meng et al., 2016].

Neither of these libraries explore an implementation of Bayesian network classifiers for this paradigm. Following the scope of this dissertation, we will discuss and define the adaptation of this family of models to this technology. Computing probability distributions from distributed data is embarrassingly parallel [Herlihy & Shavit, 2011] and can be naturally translated into a MapReduce program. The computational complexity of the algorithms defined in Section 2.5 can be reduced to a shared pattern that can be implemented as a generic framework under MapRe-
duce. We will discuss this core implementation and provide details to adapt it for each one of the presented techniques.

We will introduce and evaluate a software implementation of this proposal, materialized in the spark-bnc Apache Spark package discussed in 6.5. We will study the scalability of this approach for a range of large-scale problems in presence of different computing architectures and resources in order to provide clear insights of its behaviour and scalability properties.

### 6.2 Distributed Bayesian Network Classifiers

Any parallel implementations should start with a computational complexity analysis of the algorithms to discover potential optimization strategies. It is important to identify the limitations of the underlying technology as well. In MapReduce, the main bottlenecks appear when data is transported, especially when reading or writing from disks or when transferring large amounts of data through the cluster’s network. We must design our algorithms to minimize this data movement in order to maximize the parallelism even when using a main memory based framework such as Apache Spark.

In Section 2.5 we saw that some BNCs require structural learning (TAN, kDB) while others avoid this step by using a fixed structure (NB, AkDE). An algorithm that performs structural and parametric learning separately often requires multiple passes through the data, i.e. kDB when $k > 1$. This can be avoided by reusing the same statistics computed from the first stage into the second one such as in TAN, if it is not possible then the data is read multiple times introducing a considerable overhead, especially in distributed paradigms such as MapReduce. This problem is aggravated, even becoming intractable, if the data is too large to fit or to be kept in main memory. In such cases, a single pass though the data can learn a classifier without the need of storing the data itself, this is known as out-of-core learning and is a demonstrated property of many BNCs [Martínez et al., 2016]. A comprehensive summary on the complexity of the different BNCs can be found on Table 6.1.

The complexity of every BNC is driven by the computation of multi-dimensional contingency tables from the training data. Structural learning such as in kDB or TAN requires the estimation of information theory metrics such as mutual information and conditional mutual information (Equations 2.15, 2.14) for every pair
6. DISTRIBUTED LEARNING OF BNCS

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Time</th>
<th>Space</th>
<th>Model</th>
<th>Data Passes</th>
</tr>
</thead>
<tbody>
<tr>
<td>NB</td>
<td>$O(nm)$</td>
<td>$O(nw^c)$</td>
<td>$O(lnw)$</td>
<td>1</td>
</tr>
<tr>
<td>TAN</td>
<td>$O(n^2m + n^2 \log n + n)$</td>
<td>$O(lw^2n^2)$</td>
<td>$O(lnw^2)$</td>
<td>1</td>
</tr>
<tr>
<td>kDB</td>
<td>$O(n^2m + n^2 \log n + knm)$</td>
<td>$O(lw^2n^2)$</td>
<td>$O(lnw^k)$</td>
<td>2</td>
</tr>
<tr>
<td>A$k$DE</td>
<td>$O(mn^k)$</td>
<td>$O(lwn^k)$</td>
<td>$O(ln^k\bar{v}^{k+1})$</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6.1: Computational and spatial complexity of BNCs.

of variables and the class, which in turn requires the computation of a contingency table among such variables. In parametric learning, estimating a CPT for a given node following a frequentist approach such as MLE (Equation 2.9) requires the computation of a contingency table for such attribute, its parents and the class.

This pattern can be found on all of the complexity formulas in Table 6.1, where computing a contingency table is expressed as a complexity of $O(mn^k)$, where $m$ is the cost of reading the complete training data and $k + 1$ is the dimensionality of such table: i.e., estimating the parameters for a NB classifier requires learning a 2-dimensional table involving each attribute ($k = 1$) and the class, whereas for the A$k$DE classifier a 3-dimensional table is required, involving each pair of attributes ($k = 2$) and the class.

In general, these contingency tables can be represented by histograms or frequency counts distributions among the different configurations of a given set of attributes $S \subseteq X \cup \{Y\}$. We will compute the frequency of each occurrence for the joint set of states $\times \Omega_X \setminus \{X_i \in S \}$ for the attributes in the subset $S = \{X_1, \ldots, X_r, Y\}$, where the contingency table, $\#D(S)$, stores the number of occurrences of each possible configuration defined in $\times \Omega_X$ in a given dataset $D$.

Learning a particular type of BNC requires computing a given number of contingency tables, defined for a collection of sets of variables $\Psi = \{S_1, \ldots, S_v\}$; e.g., learning a Naive Bayes classifier requires estimating the counts for each attribute and the class, and thus the sets of variables from which estimating the counts will be $\Psi = \{S_i = \{X_i, Y\} | \forall X_i \in X\}$. A pattern can be defined as a general framework to cover all the proposed BNC algorithms. From now on, we will pose our problem in such general view, with the purpose of defining a common procedure to learn any of the described models; later on, we would be able to instantiate this framework to match every particular algorithm. This general procedure is based on computing the contingency tables for a given set of attribute combinations $\Psi$.  

98
that would be configured according to the particular model selected.

Computing such contingency tables for a large volume of data is a problem that falls naturally on the MapReduce paradigm. We can follow two different strategies to address the problem of computing the joint frequencies of any given set of attribute/class combinations in parallel:

- **Horizontal parallelism** is the most straightforward approach which targets the \( m \) dimension of the dataset. It starts by splitting the training dataset into different chunks and distributing them across the available computing nodes. From there, the contingency tables are computed partially for each chunk of data and combination of attributes \( S \in \Psi \). Finally, the partial distributions can be collected and aggregated to recover the full distribution over the whole dataset.

- **Vertical parallelism** [Madsen et al., 2014] distributes the computation among the different subsets in \( \Psi \). This strategy is meant for high dimensional domains where there is a large number of attributes. In these cases, a simple horizontal distribution of the dataset would imply a large overhead when collecting the different partial counts, so vertical parallelism can be used even combined with the previous approach. This is done by computing each contingency table as a separate parallel task. This strategy involves a vertical distribution of the dataset according to the different subsets \( S \in \Psi \); and, as the subsets may overlap, this involves a complex combinatorial problem in order to minimize data replication and to assure the balance of tasks across the different cluster nodes.

### 6.2.1 Horizontal Parallelism in MapReduce

According to the previous definition, our problem can be reduced to estimating the contingency tables for a particular number of attribute subsets \( \Psi = \{S_1, \ldots, S_v\} \) over a training dataset \( \mathcal{D} \). A horizontal strategy starts by partitioning the data into chunks \( \{\mathcal{D}_1, \ldots, \mathcal{D}_h\} \mid h \leq m \) that will be distributed into, ideally, \( h \) map tasks. Each of these tasks computes a partial contingency table \( \#_i(S_j) \) for the available local chunk \( \mathcal{D}_i \), and for each subset \( S_j \in \Psi \). Then, each map task emits a collection of key-value pairs \( \langle S_j, \#_i(S_j) \rangle \) for each subset, associating them with their corresponding distribution. A graphical representation of this process is illustrated in Figure 6.1.
6. DISTRIBUTED LEARNING OF BNCS

![Graphical diagram of the horizontal MapReduce algorithm.](image)

In the reduce stage, these pairs will be grouped by their keys (representing a subset of attributes) and sent to the corresponding reduce tasks, where the ideal number of tasks is \( v \), one for each subset. The reduce stage aggregates the partial distributions for the different attribute subsets in parallel and emits a new key-value pair containing the corresponding subset and its full contingency table computed for the complete dataset.

The previous framework can be configured to match any of the described BNCs by providing a particular range of attribute subsets for \( \Psi \).

### 6.2.1.1 Distributed Naive Bayes

Being the most simple of the classifiers in our study, we will start by characterizing the previous MapReduce algorithm for the NB learning algorithm. As this model does not require a previous structural learning stage, we will just define the estimation via MLE of the parameters. Concretely, we need to set the topology of the multi-dimensional contingency tables to be computed by correctly identifying \( \Psi \); in this case, the NB classifier encodes the conditional probabilities for each attribute and class, thus defining a 2-dimensional table with complexity represented by the subsets \( \Psi^1 = \{S_i = \{X_i, Y\} \forall X_i \in X\} \).

### 6.2.1.2 Distributed \( \text{AkDE} \)

Owing to the lack of structural learning in the \( \text{AkDE} \) classifier, we can proceed exactly as with the NB model. However, the number of parameters to compute will depend on the value of \( k \), which affects the dimension of the contingency...
tables that will be estimated. Concretely, the dimension of such tables is 3 for A1DE with $\Psi^2 = \{S = \{X_i, X_j, Y\} \forall X_i, X_j \in X\}$ and is a 4-dimensional table with $\Psi^3 = \{S = \{X_i, X_j, X_k, Y\} \forall X_i, X_j, X_k \in X\}$ in the case of A2DE.

6.2.1.3 Distributed TAN

The case of the TAN classifier is particular as it performs a structural learning stage before estimating the parameters. However, this second stage can reuse the statistics computed for the structural learning part, obtaining the parameters of the final model without performing an additional pass through the data. In order to induce the structure of the model, this algorithm builds a 3-dimensional table identically to A1DE with $\Psi^2$ to compute the CMI statistics. Then, those same statistics are used to build the tree model by using the maximum spanning tree (MST) algorithm.

At this stage we have already compressed the data and removed the $m$ component from the problem, thus the complexity of the MST algorithm is $n^2$ which is independent on the number of instances in the data. In practice, the cost of this operation should not have a real impact on the overall performance of the algorithm when the size of the dataset is large; however, given that this stage is sequential, the combination of a large number of attributes in a small dataset can introduce an overhead in the execution time of the learning algorithm.

After the structure of the classifier is induced, the parameters can be estimated from the same contingency tables computed to obtain the CMI. This stage would be efficient as well as it is also independent from the size of the data.

6.2.1.4 Distributed $k$DB

Learning a $k$DB model involves two different passes through the data when $k > 1$, one during the structural learning stage and a second pass to estimate the parameters. Unlike the TAN algorithm, the statistics computed during the first step can not be reused to estimate the CPTs, as the cardinality of the latter would be higher for values of $k > 1$. The structural learning complexity is identical to the described for the TAN classifier, in which 3-dimensional contingency tables are computed for $\Psi^2$ in order to obtain the CMI for each pair of attributes and the class. Once the metrics have been computed, the graph structure is induced by the algorithm described in 2.5.1.2, which again turns out to be sequential and
independent on $m$ and thus its complexity can be neglected when compared to the main computation.

In an hypothetical case of high dimensionality, such structural algorithm could be computed using MapReduce as well:

1. Starts from the previous MapReduce job, which outputs a range of key-value pairs where key represent a pair of attributes $S = \{X_i, X_j\}$ and the value corresponds with the contingency table $\#X_i, X_j, Y$.

2. **Map stage:** For each pair of attributes $\{X_i, X_j\}$ the mutual information with the class variable is computed, which defines the partial ordering between the variables in $\sigma$. The variable with the higher dependence will have a preceding order, and thus would be considered as a candidate parent for the other attribute so a key-value pair is emitted $\langle X_c, (X_p, MI(X_c \mid X_p, Y)) \rangle$ in which the key represents the child variable and the value is formed by the candidate parent associated with its CMI value computed from the joint distribution.

3. **Reduce stage:** Each task will receive a list of the candidate parents form a given attribute $X_i$ associated with their respective score. The procedure is very simple, as each task must select the best $k$ parents according to the algorithm’s configuration.

After the structural learning stage, it is necessary to estimate the parameters of the model. However, this step would require a second pass through the data when $k > 1$, as new contingency tables for a different set of attribute combinations will be required. More concretely, we need to estimate a CPT of each node in the model so, in this case, the complexity of the process is linear regarding the number of attributes $n$. For each node we must obtain the frequency counts of each attribute given the different combinations of its parent in the induced structure; this is described by using our proposed framework for the set of attributes $\Psi = \{S_i = \{X_i\} \cup pa(X_i) \forall X_i \in X\}$.

### 6.2.2 Vertical parallelism in MapReduce

The horizontal parallel strategy implies that each task would compute a partial contingency table for all the combinations in $\Psi$. In terms of spatial complexity
this is equivalent to holding the models’ full set of parameters in memory for each task, this would require a large amount of memory to be allocated for each task if the model is high dimensional. Adding vertical parallelism would alleviate this constraint by allowing each task to target only a partial number of sets from Ψ. The strategy can be used as an standalone parallelization technique or combined with the previous one. In the vertical approach, a number of map tasks would target the same chunk $D_i$ to compute a partial contingency table for a different subset of the combinations in Ψ. Figure 6.2 shows an example in which vertical parallelism is added to the horizontal basic scheme proposed in figure 6.1, notice that the shuffle and reduce processes are identical to the previous definition. In an extreme case with no horizontal partitions, the reduce stage can be suppressed, as the contingency tables estimated during the map stage would no longer be partial.

This strategy will change the way in which the tasks and data are distributed, by requiring less partitioning of the data and a different configuration of map tasks. A practical example of how this approach could be effectively configured on a real cluster is provided in Section 6.3.

As opposed to horizontal parallelism where data partitioning is trivial, the main drawback of vertical parallelism is that finding optimal data partitions for different subsets of the elements in Ψ turns out to be a very complex combinatorial problem, in which the data must be divided by reducing replication optimally. Different strategies have been proposed to solve this problem [Madsen et al., 2014] in which smart statistical techniques are used in order to guarantee a balance according to the amount of data replicated for each vertical map task. In this work, we will assume a trivial strategy in which the data is not partitioned but replicated entirely, so the different tasks can access the variables they need for their computations. This indeed adds a notable overhead to the execution as a large amount of redundant data would be read from disk and transported through the network.

### 6.3 Empirical Evaluation

We have conducted a series of experiments to evaluate the performance of the different algorithms for varying levels of parallelism, while combining the proposed horizontal and vertical strategies. The aim of these experiments is to provide a clear overview of the different behaviour for the parallel algorithms to endorse the
validity of our proposal and to provide a reference to future practitioners.

We present two series of experiments, the first one based on synthetic data, which will allow us to evaluate the improvement in a wide range of scalability problems. Finally, we present a series of experiments using real-world data obtained from challenges to show the performance of the algorithms in particular domains. In both cases we report several metrics regarding the execution time of the algorithms when different computational resources are present. The accuracy of the classifiers will not be evaluated in this work, as the algorithms are identical to their corresponding sequential version, and thus the resulting models will maintain its discriminative power.

### 6.3.1 Execution Environment

Our experimental set-up consists on a one master/six slave nodes cluster equipped with dual Intel Xeon E5-2609v3 1.90GHz hexacore processors and 64GB of RAM each one. Each worker node is running the HDFS file system on 4x1TB disks and managed by Cloudera CDH 5.5 distribution.

The main MapReduce framework employed for our implementation is Apache Spark 1.6.0 using standalone deploy, however, we will also provide a reference im-
plementation of the AkDE algorithm in Apache Hadoop to compare the efficiency of an in-disk against an in-memory computing framework. We will launch experiments using different configurations of the cluster by allocating multiple amount of resources in order to test the behaviour of the algorithms for different architecture layouts.

6.3.2 Synthetic Data Experiments

Conducting experiments on synthetic data will provide an intensive study on the scalability of the proposed framework. These experiments use an incremental synthetic model which allow us to obtain different problems covering a wide spectrum of the scalability challenge, on both the number of attributes and the number of instances. For that, we define a random synthetic Bayesian network model from which we will sample a series of datasets with different properties.

More concretely, we have constructed a random model by using a modified version of the procedure described in Alonso-Barba et al. [2013]. In our case, we will focus on creating a model that matches a supervised classification problem; more concretely, we will construct an augmented naive Bayes model over an indefinite set of variables \( \{Y\} \cup \{X_1, \ldots, X_n\} \), in which all attributes are dependent on the class and on a random set of parents. To set such dependencies, a topological ordering is arbitrarily selected over the variables and an average of \( p \) parents are randomly selected for each node. The parameters are then obtained accordingly to the given structure by following uniform distributions. A graphical representation of one possible model that follows this example is shown on Figure 6.3. We have generated a range of models with different number of attributes and several samples with incremental number of instances for each one of them, details are shown in Table 6.2.

We use synthetic data because we are not evaluating the classification performance of the models, as we assume that it would be equivalent to the performance in traditional domains that has been tested many times in the literature. Such performance in these specific domains should be compared with relevant data and additional indicators, this has been left for a future work intentionally.

Figure 6.4 shows the summarized performance for each learning algorithm over the synthetic samples when using the proposed Apache Spark implementation with 64 parallel tasks. The most noticeable result is perhaps the huge difference
6. DISTRIBUTED LEARNING OF BNCS

Figure 6.3: Structure of a random synthetic network generated where the number of attributes is indefinite.

<table>
<thead>
<tr>
<th>m</th>
<th>200</th>
<th>400</th>
<th>600</th>
<th>800</th>
</tr>
</thead>
<tbody>
<tr>
<td>4M</td>
<td>1.6GB</td>
<td>3.2GB</td>
<td>4.8GB</td>
<td>6.4GB</td>
</tr>
<tr>
<td>8M</td>
<td>3.2GB</td>
<td>6.4GB</td>
<td>9.6GB</td>
<td>12.8GB</td>
</tr>
<tr>
<td>16M</td>
<td>6.4GB</td>
<td>12.8GB</td>
<td>19.2GB</td>
<td>25.6GB</td>
</tr>
<tr>
<td>32M</td>
<td>12.8GB</td>
<td>26.6GB</td>
<td>38.4GB</td>
<td>51.2GB</td>
</tr>
</tbody>
</table>

Table 6.2: Properties of the synthetic datasets generated for the study. The number of variables and instances increases with different proportions. The random model has been created by assuring an average of 3 parents for each model, with discrete variables with 3 states.
Figure 6.4: Results for each of the 4 learning algorithms and each synthetic dataset. The y-axis shows the execution time of the algorithms, notice that each scale is unique for each plot. The x-axis shows the different number of samples \(m \cdot 10^6\) for the given problem while the number of attributes \(n\) is separated into the different charts.

between Naive Bayes and the other classifiers, showing that an algorithm with linear complexity is not comparable with the others having quadratic complexity. The second important observation is the sequential structural learning stages in TAN and KDB\(k=1\) have not a significant impact on the performance of these classifiers. We can easily see that their results are equivalent to A1DE.

We can observe that the complexity of the algorithm is driven by the dimensionality of the problem. The execution time increases largely as the number of attributes is incremented. Next we will explore to what extent the MapReduce approach increases the efficiency of the algorithms for a given configuration of parallel resources in the cluster. For that, we have conducted an exhaustive comparison among all the problems using different amounts of parallelism, that is shown in Figure 6.5.

These experiments confirm that using a higher number of parallel tasks provides consistent scalability for the algorithms, which is in fact larger in the case of more complex models like A1DE. In addition, we can observe that scalabil-
Figure 6.5: Execution time (logarithmic scale) for all combination of number of attributes (y-axis), number of instances (y-axis) and number of parallel tasks (different plots). The color represents the execution time where blue is the lowest towards orange, red and black being the greatest.

Using MapReduce introduces a certain overhead in the execution as the number of parallel tasks in increased, Figure 6.6 compares the scalability of the different executions of the algorithm for a given problem by measuring the the speed-up and efficiency of the execution. For the speed-up, the results are compared with those using the minimum amount of parallelism (4 tasks), the efficiency is obtained as the ratio between speed-up and the actual number of parallel tasks; this measure can be interpreted as the proportional performance of a parallel task (a physical core) compared to a single sequential task executing the algorithm\(^1\).

These results reflect that the overhead is higher in the case of Naive Bayes and in the presence of a lower number of attributes in the problem. Our conclusion is that the proposed implementation is more favourable for larger datasets and high dimensional problems when the size of the cluster grows.

\(^1\)An optimal theoretical value for efficiency would be 1, where each parallel task would perform equally as compared to the single sequential task, thus obtaining perfect parallelism. In practice this is not possible, as parallelism involves an amount of overhead which is proportional to the number of active tasks, we can observe that in fact, efficiency is reduced proportionally to the number of cores used in the execution.
6.3.3 Real Data Experiments

We have conducted a range of experiments over a series of real world problems obtained from public challenges. These real datasets present a combination of both large scale and high dimensionality, making the use of scalable algorithms mandatory. A summary of their properties is shown in Table 6.3.

Execution time for every algorithm and dataset is shown in Figure 6.7, where we can observe stability for splice and EBLD14* and one anomaly in the case of epsilon when using TAN. This dataset is unique in our study given the small

<table>
<thead>
<tr>
<th>Name</th>
<th>n</th>
<th>m</th>
<th>Size</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>splice</td>
<td>141</td>
<td>50M</td>
<td>14.20GB</td>
<td>Sonnenburg &amp; Franc [2010]</td>
</tr>
<tr>
<td>ECBLD14*</td>
<td>631</td>
<td>4M*</td>
<td>5.26GB</td>
<td>Bacardit et al. [2012]</td>
</tr>
<tr>
<td>epsilon</td>
<td>2000</td>
<td>500K</td>
<td>1.90GB</td>
<td>Yuan et al. [2012]</td>
</tr>
</tbody>
</table>

*This is a sample from the original dataset.

Table 6.3: Properties of the large-scale real datasets included in the experiments.

Figure 6.6:
Speed-up and efficiency for multiple executions of MapReduce Algorithms.
Figure 6.7: Execution time for each algorithm and real dataset. The y-axis shows the execution time of the algorithms, notice that each scale is unique for each plot. The x-axis shows the different algorithms.

amount of instances that it contains, especially if we take into account that it has a large number of attribute variables. This overhead is being added at the structural stage of the TAN algorithm, in this case, the complexity of computing the contingency tables is relatively small given the imbalance between \( n \) and \( m \). In these cases where we have high dimensional problems with low sample size the performance of the algorithms that require two steps (TAN and \( k \)DB) can be compromised.

6.3.4 Sequential Implementation

In order to measure the real contribution of the MapReduce approach we will compare our proposal against an optimized sequential version. We will compare the performance of each Spark task against a single core running an optimized Java implementation\(^1\). For that we have implemented the A1DE algorithm in Java using the MOA library [Bifet et al., 2010] to read big datasets as a data stream, thus eliminating any memory restrictions. The code has been carefully optimised in order to provide a fair comparison between the two approaches when focusing on a

\(^1\)Spark runs on the JVM, so optimized Java code on a single core of our cluster is one of the most fair comparisons that can be made to test spark against a sequential counterpart.
Figure 6.8: Comparison between sequential and MapReduce implementations of the A1De algorithm. The left most value in each plot represents $h = 1$ which is the sequential execution of the algorithm, followed by several MapReduce executions using an increasing number parallel tasks ($\times 2$).

real context. Figure 6.8 shows the runtime, speed-up and efficiency progression for executions of the A1DE algorithm using different amounts of resources, including the sequential approach. We can confirm that the initial impact on scalability, when moving from sequential (1 task) to MapReduce (4 tasks), is significant, however the scalability is more evident when additional Spark nodes are included. A certain amount of overhead can be noticed when the number of parallel tasks is increased, however, the speed-up plots show that in the case of splice this overhead is almost non-existent, which means that using MapReduce and Spark favours horizontally large problems with a massive amount of examples.
6.3.5 Vertical Parallelism

Distributing the data across the cluster vertically is not a trivial task when the computation carried out in each task involves subsets of several variables that may overlap, as it is the case on A1DE, TAN or $k$DB but not on Naive Bayes. Additional data and computation distribution strategies may be explored in future work, where complex metrics may be used to ensure load balance and data locality in the cluster. In this work, we have just explored a naive approach on vertical parallelism, which is sending an exact copy of the chunk, including all attributes, to every task. This solution comes at the expense of a large overhead introduced by the additional amount of data distribution and replication that is required.

The proposed distributed implementation enables vertical parallelism naturally, an adapted version of the A1DE algorithm that combines both horizontal and vertical approaches is provided. Figure 6.9 shows the results for this new implementation. For this new approach we will express the task configuration as a proportion of vertical and horizontal tasks. More concretely, we will run the algorithm using a fixed number of 64 parallel tasks, but with a different set-up regarding the number of horizontal $(h)$ and vertical $(v)$ tasks $(h \cdot v)$. The combinations included in the experiment are: $64 \cdot 1$, $32 \cdot 2$, $16 \cdot 4$ and $8 \cdot 8$.

We can observe a notable overhead in the bigger dataset splice, however it is less significant for the other two datasets and even non-existent. In fact, we can observe that in the domains with higher dimension the vertical approach is even more efficient than the pure horizontal one, especially if $m$ is not very large like in the case of epsilon. These results support the idea that vertical parallelism is beneficial for high dimensional domains.

6.3.6 In-Memory vs Disk Frameworks

We want to provide a contrast on the real performance differences of Apache Spark and Apache Hadoop. The former is expected to obtain better performance, especially for the datasets with a larger amount of instances, while the difference should be shorter in the case of higher dimensional data, where the learning is more computationally intense than space demanding.

Figure 6.10 replicates the previous results for the A1DE classifier in both the Spark and Hadoop implementation. We can see that the differences are not much higher in the cases of ECBLD14 and epsilon, however, splice obtains huge im-
Figure 6.9: Comparison between different proportions of horizontal and vertical MapReduce tasks for the A1DE classifier.

improvements in terms of speed-up. Apache Spark is by far more scalable and will improve further than Hadoop if we keep adding more resources to the cluster.

### 6.3.7 Size of the Models

Another interesting indicator is the resulting size of the models. Although the spatial complexity of the resulting models has never been a main concern when dealing with traditional sized problems, it can become a barrier for high dimensional domains, especially when learning models over an order of complexity higher than linear. This is the case for A1DE and A2DE, where the size of the resulting model grows quadratically and cubically along with the dimensionality of the classifier.

Figure 6.11 shows a graphical comparison for the different classifiers and problems. It is clear that the difference between the size of A1DE and the other models is huge, the former having model sizes that could impact on the performance of the classification step. However, the real problem comes when A2DE and higher order versions of AkDE are considered as suitable candidates for solving high dimensional problems. Table 6.4 shows the model sizes for A1DE and A2DE, where
Figure 6.10: Comparison between Apache Hadoop and Apache Spark implementations of the A1DE classifier.
Figure 6.11: Sizes of the resulting models for each classifier and problem. The size of the circles represent approximate proportions regarding the size of the model for each dataset.

we can observe that learning an A2DE model becomes an intractable problem for most of the proposed domains. The only plausible experiment was splice, for which we even obtained a critical size of the classifier. The remaining experiments exceeded the available resources and the size was just estimated. It is clear that new approaches should be proposed in order to learn more powerful classifiers like A2DE in those domains, for which some of the constrains of the algorithm must alleviated to reduce the size of the model.

<table>
<thead>
<tr>
<th>splice</th>
<th>ECBLD*</th>
<th>epsilon</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1DE</td>
<td>11.60 MB</td>
<td>486.10 MB</td>
</tr>
<tr>
<td>A2DE</td>
<td>2.06 GB</td>
<td>≈79.16 GB</td>
</tr>
</tbody>
</table>

Table 6.4: Sizes of the resulting model for the A1DE and A2DE. Boldfaced values represent approximations, due to the impossibility of running those experiments.
6. DISTRIBUTED LEARNING OF BNCS

6.4 Conclusions

We have designed a framework to implement BNCs under one of the most popular programming paradigms nowadays that is MapReduce. We have introduced our proposal from a theoretical point of view, which do not constrains its application to any technology, so its design is agnostic in terms of languages and libraries. We do this with the purpose to serve as a reference for future research and practitioners that might implement this kind of algorithm into future relevant technologies.

We provide our own open source contribution as an implementation of such algorithms into Apache Spark library, which is the current state-of-the-art technology in big data processing and machine learning. Our solution can be scaled to high levels of performance by using any architecture such as an on premises cluster as in our experiments or even a public cloud computing environment.

We have performed an extensive empirical evaluation to show how this framework scales with the size of different problems and architectures. We hope that the performance indicators that are discussed serve as a baseline to compare future research on scalable models, specifically designed to solve large scale and high dimensional problems.

In addition, we have extended our proposal by introducing different strategies for distributing the computations and the data along a computing cluster, also introducing new future lines of research to optimize this framework for more complex domains. More concretely, we have defined a basic approach to verticalization of parallel computing, in addition to the usual horizontal strategy that is followed by the majority of the machine learning algorithms that have been adapted to MapReduce. This vertical strategy could be further extended with smart procedures to balance the data replication among the different vertical map tasks to optimize the scalability of the algorithms for high dimensional domains.

Finally, some of the main pitfalls of classical BNCs have been identified in order to propose additional research lines. In future works, we will define new classifier models specifically adapted for these complex domains and being native to the MapReduce/Spark programming paradigm.
6.5 The spark-bnc Spark Package

As mentioned, we provide a native and open source implementation of our work in the form of an Spark package. The source code and manual can be found in the following repository http://github.com/jacintoarias/spark-bnc.

This software can be used to reproduce our research but it is robust enough to be applied to real life problems. We aim at expanding the user base of these family of probabilistic classifiers. To the best of our knowledge there is no public implementation of these algorithms, surprisingly, as they have been widely used in past popular frameworks such as Weka [Hall et al., 2009]. We have followed the best practices and standards from both the Scala and Apache Spark development guides and is in our plans to provide long-term support and maintenance for this software.

We expect this contribution to help spread and divulge the contents and results of this dissertation.
6. DISTRIBUTED LEARNING OF BNCS
Chapter 7

New Ensemble Approaches for BNCs

The rationale behind augmented naive Bayesian classifiers is to control the complexity of the resulting models by relaxing the original independence assumption. This strategy leads to parametrized learners that can produce a wide spectrum of models. Expressiveness and efficiency can be controlled to adjust a trade-off specific to the problem at hand. Ultimately, this can be seen as balancing bias and variance where augmented models correspond with low-bias/high-variance classifiers. These are especially suitable for large data domains as variance can be reduced by learning from massive samples of data. Another variance reduction method is to aggregate a collection of models in an ensemble such as in AkDE, where we avoid structural learning by averaging a full family of constrained models. This largely increases the spatial and computational complexity of the method. We will study a new approach to reduce model space from the point of view of ensemble classifiers, where we study the individual contribution to error for each model and how model selection affects this via the aggregation process. We perform a thorough experimentation to analyse bias stability and variance reduction and compare the results within the context of other popular ensemble models such as random forest, leading to a discussion on the effectiveness of the previous approaches. The conclusions also support new strategies to design more consistent ensemble Bayesian network classifiers which we explore at the end of the chapter. †

†This chapter is based on the following papers: Arias et al. [2018]
7. NEW ENSEMBLE APPROACHES FOR BNCS

7.1 Introduction

During this dissertation we have introduced several approaches to deal with large-scale data and complex domains. We have seen that scalability is uniquely applied to each problem, where we look for an exclusive balance between the expressiveness of the model, in terms of predictive power, and efficiency, in terms of computational resources and time. This trade-off depends on the difficulty of the problem, which can be measured among other factors by its dimensionality and the available sample. However, in practice, scalability is more dependent on business requirements and restrictions and is addressed in a totally different way.

Many scalability improvements rely on heuristic regularization, model selection or augmentation techniques that end up obfuscating the learning algorithm by adding overly complicated configuration hyperparameters. Practitioners may find many of these algorithms too complex to apply and usually end up using blind optimization processes such as greedy or random hyperparameter tuning. While this gives good results in practice, we can lose the perspective on the statistical interpretation of the underlying technique and the hyperparameters that we are configuring.

If we search for the most popular algorithms in the reference machine learning software [Meng et al., 2016; Pedregosa et al., 2011], we can observe that most of them were proposed more than a decade ago and continue to be competitive in many domains and public competitions. This is especially true in the case of ensemble classifiers [Dietterich, 2000]. One of the most widely used techniques is Random Forest (RF) [Breiman, 2001] where we can intuitively control the performance of the model by tuning the number of trees in the ensemble, reducing variance at the expense of additional computational resources. The algorithm behaves asymptotically as the number of models grow, guaranteeing bias stability an reducing variance consistently, a behaviour that is both predictable and statistically sound.

Understanding the performance of an algorithm by studying how bias and variance are affected by changes in the hyperparameters is key to use the model in real problems. In this chapter we will apply these concepts to the family of Bayesian Network classifiers (BNCs). As out-of-core learners, they can be trained in a single pass through the data, being a good fit for large data domains. Algorithms such as kDB provide a single parameter $k$ which controls the complexity of the model, with...
larger values reducing its bias at the cost of increasing variance and complexity. Another popular approach is AkDE, which is an ensemble of averaged classifiers, providing variance reduction at the cost of largely increasing model complexity.

In Chapter 6 we saw that the size of the models can grow too large, becoming a scalability problem for many domains. For that reason, several proposals have attempted to reduce model space by performing model selection such as the modifications proposed in Chen et al. [2017a,b]; Martínez et al. [2016]. The majority of these perform a hybrid between filter and wrapper selection based on information theory metrics such as mutual information. This heuristic strategy has proved to be inconsistent in many problems and reduces the efficiency of the algorithm by introducing additional passes through the data, obfuscating the model and imposing additional restrictions.

We will explore a novel approach to evaluate these techniques in terms of bias and variance by analysing their behaviour in domains of different sizes. We will study how adding model complexity by increasing $k$ in $k$DB and AkDE modifies bias and variance. In addition, we will compare AkDE against another ensemble, random forest, to evaluate if averaging multiple fixed models can be considered as a consistent variance reduction. These results will be obtained by performing an extensive empirical evaluation of a wide machine learning benchmark.

### 7.2 Decomposition of Classification Error in terms of Bias and Variance

A large number of studies [Bauer et al., 1999; Breiman, 1998; Kohavi & Wolpert, 1996; Kong & Dietterich, 1995] have analysed the predictive performance of ensemble classifiers by decomposing their error into bias and variance terms. Different formal definitions can be found on the literature, however, the intuition behind these metrics is the same: A biased learning algorithm shows a persistent error when trained on independent samples while a high variance one has a particular fluctuating error for every sample.

Overfitting the training data or learning a model with a large number of parameters leads to low bias learners that usually show a high variance. We refer to them as strong learners [Breiman, 1998] and are suitable for difficult problems or large data samples where we can obtain a good average performance. They can
be improved by reducing the variance by performing regularization or increasing the data samples [Martínez et al., 2016].

On the contrary, simpler models will have very low variance at the expense of higher bias and will not be good performers by themselves. We refer to them as weak learners [Breiman, 1998] and are usually improved by ensemble optimization techniques [Chen & Guestrin, 2016].

While the interpretation of bias/variance decomposition seems intuitive, its application to discrete classification is not. It was originally proposed for quadratic regression in which the prediction is not categorical but belongs to a continuous domain and has a varying degree of error that can be separated into nonnegative terms. As a result, independent regression functions can be averaged to decrease variance without changing bias, while averaging classifiers models can increase the classification error [Schapire et al., 1998].

There are many adaptations of bias/variance decomposition for discrete classification in the literature [Schapire et al., 1998; Webb, 2000], among which we have selected that described in [Bauer et al., 1999; Breiman, 1998] as it is implemented in [Webb, 2000]1. This metric is drawn from the stability of a given learner \( \mathcal{L} \) when trained and tested repeatedly on a number of dataset samples \( T \). Such stability is measured by the central tendency, \( C_{\mathcal{L}T}(x) \), which is defined as the class with the maximum probability of being selected for a given example \( x \) by all classifiers learnt from \( T \).

\[
C_{\mathcal{L}T}(x) = \arg \max_y P_T(\mathcal{L}(x) = y) \tag{7.1}
\]

Bias can be measured as the error introduced by the central tendency of the algorithm, in other words, the error of the most frequent classification, and variance as the error introduced by the deviations from the central tendency. Notice that given this definition \( \text{err} = \text{bias} + \text{variance} \), so this decomposition is usually referred to as contribution of bias and variance to error.

The central tendency is usually estimated via repeated cross validation from the available sample data \( \mathcal{D} \). More concretely, we will perform a 10x3 fold cross validation, for a total of 30 different models, \( \mathcal{L}(T_k^i) \), and a their corresponding test dataset \( f_k^i \) for each of the \( i \in \{1, \ldots, 10\} \) repeats and \( k \in \{1, 2, 3\} \) training

---

1This methodology has been previously used to study AkDE classifier by its authors [Webb et al., 2005, 2011]
folds. This way we will obtain 10 independent predictions for each data point \( x \) from which we will compute the central tendency for this example, \( C_{\mathcal{L}T}^0(x) \), as the average.

Formally, the central tendency is defined by the following expression:

\[
C_{\mathcal{L}T}^0(x) = \arg \max_y P \left( \sum_{i=1}^{10} \sum_{k=1}^{3} 1 \left[ x \in f_k^i \land \mathcal{L}(T_k^i)(x) = y \right] \right)
\]  
(7.2)

The bias and variance contributions to error are then computed for each instance and aggregated over the dataset:

\[
bias = P_{(x,y),T}(\mathcal{L}(T)(x) \neq y \land \mathcal{L}(T)(x) = C_{\mathcal{L}T}^0(x))
\]  
(7.3)

\[
variance = P_{(x,y),T}(\mathcal{L}(T)(x) \neq y \land \mathcal{L}(T)(x) \neq C_{\mathcal{L}T}^0(x))
\]  
(7.4)

### 7.3 Bias and Variance in Random Forest

One of the most popular ensemble classification models is the random forest classifier [Breiman, 2001]. It builds a set of decision trees by combining a couple of powerful statistical regularization techniques. Bagging involves learning each model from a bootstrapped sample of the training data, and random subspaces, involves selecting a suboptimal set of nodes for each split of every decision tree. These techniques add diversity to the models and reduce the correlation between them.

This type of ensemble works best with strong learners such as deep decision trees. Breiman demonstrates that the suboptimal choices made in each model do not affect their overall classification strength and thus bias is stable. The conclusion is that an ensemble of multiple averaged trees has the same bias of a single model. The most interesting result is that the performance of the ensemble converges asymptotically to a margin defined by the proportion between the strength of the classifier and the correlation among the different trees. By regularizing the trees we reduce the correlation between the models and averaging decreases the variance in the overall ensemble.

A fully developed decision tree is a very unstable model that requires huge amounts of data to consistently estimate the parameters for its deepest nodes.
Small variations in the training data will modify the structure of the model, especially in such last levels and leaf nodes. If we combine Breimans’s finding of bias stability and the aforementioned definition of bias and variance, we can assume that both the ensemble and the individual models will estimate a consistent central tendency, meaning that the bias component of the error will be equivalent. This imply that classification errors will be likely caused by individual variations in the trees and thus should come from the variance component.

Before moving into BNCs we will illustrate this hypothesis by performing some empirical evaluation on random forest. For that we will compute error decomposition for a collection of datasets that constitute one of the most comprehensive benchmarks in machine learning, obtained from the UCI repository [Dheeru & Karra, 2017]. Table 7.1 lists such datasets and their properties.

Table 7.1: Properties of the datasets used in the experiments.

<table>
<thead>
<tr>
<th>Database</th>
<th>Cases</th>
<th>Atts.</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pokerhand</td>
<td>1000000</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>Adult</td>
<td>48842</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>Chess</td>
<td>28056</td>
<td>6</td>
<td>18</td>
</tr>
<tr>
<td>Letter</td>
<td>20000</td>
<td>17</td>
<td>26</td>
</tr>
<tr>
<td>Nursery</td>
<td>12960</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>PenDigits</td>
<td>10992</td>
<td>17</td>
<td>10</td>
</tr>
<tr>
<td>CensusIncome</td>
<td>10419</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>Mushrooms</td>
<td>8124</td>
<td>23</td>
<td>2</td>
</tr>
<tr>
<td>Musk</td>
<td>6598</td>
<td>168</td>
<td>2</td>
</tr>
<tr>
<td>OpticalDigits</td>
<td>5620</td>
<td>49</td>
<td>10</td>
</tr>
<tr>
<td>PageBlocks</td>
<td>5473</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>Spambase</td>
<td>4601</td>
<td>58</td>
<td>2</td>
</tr>
<tr>
<td>Hypothyroid</td>
<td>3772</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>Kr-vs.kp</td>
<td>3196</td>
<td>37</td>
<td>2</td>
</tr>
<tr>
<td>Splice</td>
<td>3190</td>
<td>62</td>
<td>3</td>
</tr>
<tr>
<td>Segment</td>
<td>2310</td>
<td>20</td>
<td>7</td>
</tr>
<tr>
<td>Mfeat</td>
<td>2000</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Car</td>
<td>1728</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>Contraceptive-mc</td>
<td>1473</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>German</td>
<td>1000</td>
<td>21</td>
<td>2</td>
</tr>
<tr>
<td>Vowel</td>
<td>990</td>
<td>14</td>
<td>11</td>
</tr>
<tr>
<td>Tic-Tac-Toe</td>
<td>958</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Anmeal</td>
<td>898</td>
<td>39</td>
<td>6</td>
</tr>
<tr>
<td>Vehicle</td>
<td>846</td>
<td>19</td>
<td>4</td>
</tr>
<tr>
<td>FimaDiabetes</td>
<td>768</td>
<td>9</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Database</th>
<th>Cases</th>
<th>Atts.</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BreastCancer-w</td>
<td>699</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>BalanceScale</td>
<td>625</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>CreditApproval</td>
<td>690</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>Cylinder-bands</td>
<td>512</td>
<td>39</td>
<td>2</td>
</tr>
<tr>
<td>Haberman</td>
<td>306</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>HouseVotes84</td>
<td>435</td>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>HorseColic</td>
<td>368</td>
<td>22</td>
<td>2</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>354</td>
<td>35</td>
<td>2</td>
</tr>
<tr>
<td>PrimaryTumor</td>
<td>339</td>
<td>18</td>
<td>22</td>
</tr>
<tr>
<td>Soybean</td>
<td>307</td>
<td>35</td>
<td>15</td>
</tr>
<tr>
<td>Haberman</td>
<td>306</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>HeartDisease-c</td>
<td>303</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>Audiology</td>
<td>226</td>
<td>70</td>
<td>24</td>
</tr>
<tr>
<td>New-Thyroid</td>
<td>215</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Glass-id</td>
<td>214</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>Sonar</td>
<td>208</td>
<td>61</td>
<td>2</td>
</tr>
<tr>
<td>Autos</td>
<td>205</td>
<td>26</td>
<td>7</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>155</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>TeachingAssistant</td>
<td>151</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Iris</td>
<td>150</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Promoters</td>
<td>106</td>
<td>58</td>
<td>2</td>
</tr>
<tr>
<td>Zoo</td>
<td>101</td>
<td>17</td>
<td>7</td>
</tr>
<tr>
<td>Post-operative</td>
<td>90</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>LaborNegotiations</td>
<td>57</td>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>LungCancer</td>
<td>32</td>
<td>57</td>
<td>3</td>
</tr>
</tbody>
</table>

The experiments in this chapter were run on a seven-node Apache Spark cluster with Intel Xeon E5-2609v3 1.90GHz hexacore processors and 64GB of RAM each.
2016]. We can observe a clear pattern, in which we can confirm how bias is stable and variance is reduced from the individual trees to the aggregated ensemble. This figure reflects the intuition behind the ensemble strategy in random forest, it is clear that adding more trees will not improve error beyond a point in which variance is minimum.

7.4 A$k$DE Under the Ensemble Perspective

A$k$DE can be seen as an ensemble of classifiers in which the predicted probabilities of many individual models are averaged. However, this interpretation is not compatible with many of the properties that make other ensemble classifiers so intuitive and powerful. If we try to translate the results discussed in the previous section from random forest to A$k$DE we will find a number of problematic differences.

First, A$k$DE can only aggregate a finite number of models given the value of $k$, as it will run out of class combinations. Given that, convergence is impossible, implying that it has limited power to reduce variance and that bias stability is difficult to measure.

Second, regularization of individual models in A$k$DE is not driven by a randomized suboptimal optimization. In random forest, we obtain consistent performance and bias stability by learning uncorrelated decision trees. The learning algorithm reduces correlation between models by regularizing the optimization using suboptimal subsets of features. In A$k$DE, the individual models are not obtained from optimization but by following an heuristic strategy that sets fixed graphs for each one of them. It is likely that such models will not be correlated but neither can we assume bias stability nor consistent predictive power, as the underlying dependency model is likely to include many spurious relationships.

In general, increasing $k$ in a $k$-dependent BNC, such as the SP$k$DE models in an A$k$DE ensemble, induces higher dimensional multivariate distributions that are likely to decrease classification bias. From a theoretical perspective, classification error can be decomposed into bias, variance and irreducible error that is inherent to the randomness of the dataset. Suppose there is an optimal BNCs $H'$ with the lowest possible bias that, by following the structural constrains of the framework, encodes the real dependencies between the different variables of the problem. It is clear that NB violates such optimal dependency model by removing every possible
Figure 7.1: Bias and variance decomposition of error for a random forest ensemble.
relationship between attributes, likely increasing inductive bias. On the contrary, algorithms such as kDB or TAN build on top of NB and try to approximate the distribution represented by $H'$ by learning a suboptimal set of dependencies, likely reducing the bias from NB or, from another perspective, adding inductive bias on top of $H'$. In contrast, fixed-structure models such as AkDE, include dependencies arbitrarily, increasing inductive bias as in NB.

This implies that we cannot expect bias stability in AkDE and that variance reduction in the ensemble will not be guided by the aggregation of non-correlated models. Experiments in the literature has shown that AkDE is a low variance classifier. However, to the best of our knowledge, there is no evidence of this being either a direct result of the averaging process or it is an inherent property of the individual models. We have reproduced the experiment shown in Figure 7.1 using A1DE\(^1\) to measure the contribution to error of each individual SP\(k\)DE model and its effects in the averaged ensemble. We have discretized continuous features by using 4 equal frequency bins.

Figure 7.2 shows that there is a clear difference between the behaviour of random forest and A1DE. The latter shows almost no variance in the individual models and thus the ensemble does not improve the error by reducing this component. Surprisingly, bias it still reduced during the aggregation for many datasets, although we can see that this is not consistent and many domains show that there are individual SP1DE classifiers that are better that the full ensemble. Our conclusion is that A1DE does not guarantee improvement over particular single models, so it is unreliable as an ensemble model as stated by Schapire et al. [1998]. Our hypothesis is that A1DE includes many spurious models that just add noise in the aggregation process.

While the previous discussion shows that A1DE has room for improvement, its results are nonetheless competitive when compared with other algorithms, including random forest. We have conducted the previous experiment including additional BNCs. Table 7.2 shows that the results along with the corresponding statistical evaluation including a Friedman rank tests followed by a pos-hoc using Holm’s correction with $\alpha = 0.05$. We can observe that A1DE is the best performing classifier followed by random forest with no significant differences between them. We also see that all three versions of kDB have obtained worse results in general.

\(^1\)The implementation of the algorithms is based in the spark-bnc package introduced in Section 6.5.
Figure 7.2: Bias and variance decomposition of error for A1DE.
## Table 7.2: Statistical comparison between BNCs and random forest with a maximum depth of 10.

Algorithms are compared with the best one (A1DE), rejected hypotheses are marked in bold.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>rank</th>
<th>p-value</th>
<th>win</th>
<th>tie</th>
<th>loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1DE</td>
<td>1.99</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RF</td>
<td>2.04</td>
<td><strong>8.7795e-01</strong></td>
<td>26</td>
<td>0</td>
<td>27</td>
</tr>
<tr>
<td>kDB1</td>
<td>2.68</td>
<td>4.9900e-02</td>
<td>37</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>kDB2</td>
<td>3.74</td>
<td>3.9885e-08</td>
<td>46</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>kDB3</td>
<td>4.56</td>
<td>2.6303e-16</td>
<td>50</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

While both ensemble methods are the top performers, their internal behaviour is totally different. Figure 7.3 condenses the bias and variance relationship between the ensemble and the individual models by measuring their differences. We can confirm that random forest reduces the error of the individual models consistently, while A1DE does not always improve over them. This is probably motivated by the absence of variance in the case of A1DE, as opposed to random forest. On the contrary, we can observe that random forest has almost no bias difference on average and that A1DE has lower bias that the individual models.
7. NEW ENSEMBLE APPROACHES FOR BNCS

7.4.1 On the Effectiveness of Model Selection

In Section 2.5 we introduced many extensions to improve AkDE [Chen et al., 2014b, 2017a,b; Zheng & Webb, 2007]. The purpose of these algorithms is twofold: to reduce model space and to improve performance. All of them are related the hypothesis of using the (conditional) mutual information among the attributes and the class, as an indicator of classification performance of the individual models. In our experiments, we have seen that even a single model can be more powerful than AkDE, and that the ensemble is not averaging the model to reduce variance but to stabilize bias.

We can conclude that AkDE should always benefit from removing certain models from the ensemble, so we may think of model selection in AkDE as part of the learning algorithm rather than a regularization step. AkDE could improve bias stability if we isolate the spurious error component that comes from highly biased models. We have seen that there is a total lack of variance in the error of the ensemble.

This reasoning leads us to a new hypothesis: The best subset of models in a given AkDE ensemble is the one that minimizes the individual bias of the models. We can test this empirically by comparing different model selection approaches, more concretely, we will consider a greedy wrapper search guided by error, bias, variance as well as mutual information and random selection. Our experiment learns an A1DE classifier incrementally by including the best model at a time according to the selected criterion. A graphical representation of this experiment can be seen on Figure 7.4, where each line represents the error progression of the ensemble (y-axis) as they include additional models (x-axis).

Table 7.3 shows a comparison of the different criteria for every dataset in Table 7.1. We measure the performance as the sum of the error on every step of the incremental search of the ensemble. We compare the superiority of each algorithm by measuring how many times it obtains a better performance than the best one, which is error. We have also conducted a Friedman test and Holm post-hoc test as usual. These results show that error and bias are equivalent as we expected, and that mutual information, variance and random selection cannot be compared. This supports our original hypothesis empirically and raises a question about the effectiveness of mutual information for model selection in the BNC ensemble framework.
Figure 7.4: Incremental selection of models in A$k$DE for different criteria.

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Rank</th>
<th>p-value</th>
<th>Win</th>
<th>Tie</th>
<th>Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>error</td>
<td>1.52</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>bias</td>
<td>1.83</td>
<td>3.6273e-01</td>
<td>24</td>
<td>3</td>
<td>17</td>
</tr>
<tr>
<td>mi</td>
<td>3.75</td>
<td>7.8353e-11</td>
<td>43</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>variance</td>
<td>3.81</td>
<td>3.7139e-11</td>
<td>41</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>random</td>
<td>4.09</td>
<td>1.0269e-13</td>
<td>43</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 7.3: Statistical comparison between different model selection criteria in A1DE.

*Algorithms are compared with the best one (error), rejected hypotheses are marked in bold.*
7. NEW ENSEMBLE APPROACHES FOR BNCS

7.5 Variance Reduction in Large Datasets

We have seen that deep decision trees present high variance, that is usually reduced by averaging a collection of them with a technique such as random forest. In general, low bias learners require complex distributions that are difficult to estimate from the data. For example, as we go deeper into the structure of a decision tree, we will have less data from which to compute the required statistics. Any change to the original training set, i.e. a bootstrapped sample, could have an amplified effect in the variance of such deepest levels of the tree. This is the main source of variance in a decision tree, so it is easy to realise that, if we provide a larger training sample, it is likely that inducing a deep tree could be more stable.

Variance reduction and bias stability can be evaluated by learning a model with incremental samples from a large dataset. For this purpose, we conducted an experiment using the *pokerhand* synthetic dataset (8 classes, 10 categorical features, 1M examples) using incremental samples of 50k instances each. We have evaluated different ensembles of random forest and decision trees with maximum number of levels of 3, 5 and 10 to illustrate how bias and variance converge for models of increasing complexity.

Figure 7.5 shows the evolution of bias, variance and error in the described experiment. We can draw several conclusions from this result:

- Larger data samples can reduce bias in more complex models. We can confirm that deeper decision trees have lower bias than shallower ones. However, if we look at trees with depth 10 and 15, we can see that the latter only becomes a more powerful model if we learn it from a larger data sample. Notice how error decreases at a larger rate for the tree with depth 15 as a result of a reduction of both bias and variance. The shallower models stabilize sooner and do not seem to improve with the addition of more data samples, so it would be just a waste of computational resources.

- Random forest is more stable than a single tree. We can easily see that random forest does not benefit as much as a single tree from a larger data sample. It converges sooner, given that variance has been reduced by the aggregation process and is not affected by additional samples.

- Non randomized trees are less biased than suboptimal ones. Learning trees from a subset of features trees, as in random forest, increases diversity at
the expense of adding inductive bias. Notice how the deeper decision tree becomes more accurate than the ensemble when we are able to reduce the variance using a larger data sample. However we should note that the ensemble is the more stable and accurate method even when the sample is very small.

We can conclude that random forest provides stability as well as predictive power even when the data is small, so it can be applied more generally and efficiently to a wider range of problems as it requires less data. However, a single decision tree that converges on variance can be even more powerful and much more efficient than an ensemble if our dataset is large enough.

In the case of BNCs, and more concretely in the kDB and A&DE frameworks, we can control the complexity of the models by tuning $k$. Increasing its value will make the algorithm compute more complex, and thus kes biases, probability distribution. As in deep decision trees, computing the joint frequency of a large
set of variables will end up using a very small subset of examples for the most infrequent cases. This can result in poorly calibrated probabilities, especially if we use smoothing techniques [Madden, 2009]. In [Martínez et al., 2016], the authors demonstrate that larger samples of data allow the algorithms induce more accurate distributions for higher values of $k$. The rationale is that larger data samples will reduce the variance of complex multivariate distributions.

As in the previous case, we want to measure the stability of various BNCs on a particular problem by increasing the amount of samples available. In Figure 7.6 we can confirm the expected behaviour for the majority of the discussed models. Highly biased models such as naive Bayes or $k$DB with $k = 1$ will not improve over an increased sample size as they show almost no variance as well as the highest bias.

On the contrary, bias is greatly improved and stabilizes for higher values of $k$. The most interesting result is the huge improvement that we can observe for $k$DB with $k = 3$ when we increase the amount of training data. Looking at the three plots we can confirm that the improvement comes from variance reduction when the data sample is larger, as we are able to calibrate the most complex parameters.

Finally, we can observe that the behaviour of A1DE is anomalous. We might expect A1DE to achieve similar performance to $k$DB with $k = 1$ or $k = 2$ given its number of parameters and its ability to reduce variance. Surprisingly, A1DE worsen its performance as bias is increased with the sample size, meaning that the central tendency drifts as the sample grows. To delve deeper into this anomaly, we ran a variant of the A1DE classifier by using majority voting among the individual models instead of averaging the probabilities for classification. This is included in Figure 7.6 as $A1DE-V$, where we can observe that its bias is now stable over the different data samples, although unfortunately at its highest level.

As a conclusion, we can state that averaged models soften the errors of biased classifiers, while majority voting ones bias the ensemble towards extreme models. In this experiment, increased sample sizes causes a calibration in the probability tables which produces more extreme probability distributions, increasing the distance to the correct class in the case of biased models.
Figure 7.6: Evolution of bias and variance for large data samples in BNCs.
7. NEW ENSEMBLE APPROACHES FOR BNCS

7.6 Alternative Approaches for Ensembles of Bayesian Network Classifiers

We have seen that while AkDE obtains good results in practice it can be inconsistent and unreliable in particular situations, especially since it has few of the desirable properties of ensemble classifiers that we have discussed. By selecting subsets of models we can improve the overall performance of the ensemble but experiments have shown that only wrapper approaches are significantly better than random choice, and these are not efficient to be computed in a real-world scenario.

We should consider if fixed-structure SPkDE models are the best choice as base classifiers for an averaged ensemble. Since AkDE requires model selection, it no longer benefits from avoiding structural learning and more complex models such as kDB could be considered to form an ensemble. A study of different types of ensembles is presented in Duan & Wang [2017], where we can highlight the kDF (k-dependence forest) algorithm. The authors define a new ensemble classifier that learns an altered version of kDB models for each predictive attribute $X$. The strategy defines a more sophisticated ordering algorithm based on conditional mutual information. Such approach outperforms A1DE empirically but still lacks some of the desired properties of an ensemble: It only averages a limited number of models and is guided by a finite non-randomized learning algorithm.

We propose a different approach, averaging kDB models as well, but taking inspiration from random forest. We define the Random $k$-Dependent Bayesian Classifiers (R$k$DB) as an ensemble of $h \in [1, \infty)$ independent models learnt by a slightly altered version of the kDB algorithm presented in Section 2.5 to introduce diversity and reduce correlation between the models:

- As in the original algorithm, a ranking $\sigma$ is established between the predictive attributes by means of their mutual information with the class variable, $MI(X, C)$ (eq. 2.15).

- For each attribute $X_i$ being $X_{\sigma(i)}$ its position on the previous ranking $\sigma$, we sample a random subset $\sigma_i$ of the attributes preceding $X_i$ in $\sigma$: $\{X_1, \ldots, X_{i-1}\}$. The size of this subset will be selected as a proportion $\alpha \in [0, 1]$ of the original size with a minimum size of $k$.

Then we proceed as in the original algorithm by computing the conditional mutual information $MI(\cdot, X_i \mid Y)$ (eq. 2.14) given the class for the subset
Table 7.4: Statistical comparison between R$k$DB and other ensembles. Algorithms are compared with the best one (A1DE), rejected hypotheses are marked in bold.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>rank</th>
<th>p-value</th>
<th>win</th>
<th>tie</th>
<th>loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1DE</td>
<td>2.48</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RF</td>
<td>2.49</td>
<td>9.7929e-01</td>
<td>26</td>
<td>0</td>
<td>27</td>
</tr>
<tr>
<td>R$k$DB1</td>
<td>2.76</td>
<td>8.7224e-01</td>
<td>30</td>
<td>2</td>
<td>21</td>
</tr>
<tr>
<td>kDB1</td>
<td>3.58</td>
<td>7.1644e-03</td>
<td>37</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>R$k$DB2</td>
<td>4.75</td>
<td>1.8647e-09</td>
<td>46</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>kDB2</td>
<td>4.93</td>
<td>7.4299e-11</td>
<td>46</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

of attributes preceding $X_i$ in $\sigma$: $\{X_1, \ldots, X_i \} - 1$ and then selecting the best $k$ attributes with the highest dependence as the parents of $X_i$.

- Finally the class variable $Y$ is added as a parent for all the predictive attributes.

The resulting models will be suboptimal according to mutual information so we can expect a convergence on predictive power while at the same time having different structures. We have considered using bagged samples of data to compute the statistics, however the results shown in Chen et al. [2017a] state that the probabilities end up being badly calibrated.

We have conducted additional experiments for this new algorithm, using the same benchmark as before and configuring it to learn 20 models and different levels for $k$. Figure 7.4 shows that in fact R$k$DB with $k = 1$ performs comparably to A1DE and random forest. Unfortunately, larger values of $k$ suffer from the same problem as the individual $k$DB models, that is, high bias due to poor probability calibration from small data samples. We could expect these classifiers to perform much better in presence of larger datasets as is the case of Figure 7.6.

The distribution of the bias and variance decomposition of the error is an additional performance indicator of this kind of classifier. Figures 7.7 and 7.8 show the decomposition for the ensemble and each individual model for R$k$DB with $k = 1$ and $k = 2$ respectively. We can observe that R$k$DB resembles RF, showing a more consistent pattern in which the individual models have similar bias and variance and the ensemble reduces the latter. However, we can see that models with $k = 1$ do not present enough variance to improve the error significantly, and that models with $k = 2$ are biased and thus the resulting error is higher. Finally,
Figure 7.9 shows the average differences between the ensembles and the individual models, in which we can confirm that both variance and bias are being reduced in the case of $R_k$DB in contrast with A1DE.

### 7.7 Conclusions

We have conducted a series of experiments that explain a number of previously untested properties of ensemble BNCs. We show that averaging models with low variance and high bias due to fixed spurious dependencies, e.g. A1DE, leads to confusing and non-relevant results in terms of bias and variance. This obfuscates the purpose of using an ensemble classifier and will be unreliable in practice. We have seen that heuristic model selection is unreliable as well, given that only very high cost wrapper approaches provide consistent results.

On the contrary, we have explored how $k$DB performs when the data sample is large enough for it to correct and calibrate its parameters if $k$ is higher than 1. We have extended existing experiments by measuring contribution of bias and variance to error to check that bias is stable but variance must be reduced.

Combining these two results we propose a new ensemble of BNCs called random $k$-dependent Bayesian classifiers in which we formulate the ensemble for an indefinite number of uncorrelated models. This approach behaves closely to existing state-of-the-art models and obtains very relevant performance, while at the same time providing a statistically interpretable framework to tune the different hyperparameters. Exploring different regularization and diversification techniques in these new kind of ensembles could lead to new powerful classifiers. Parameter estimation, as well as variable discretization and engineering are prolific paths to explore in this direction.

### 7.8 Reproducibility

Available code and data for further use of the algorithms presented in this chapter as well as instructions to reproduce the experiments can be found on [http://github.com/jacintoArias/pgm2018](http://github.com/jacintoArias/pgm2018).
Figure 7.7: Bias and variance decomposition of error for a RKDB with $k = 1$. 
Figure 7.8: Bias and variance decomposition of error for a RKDB with $k = 2$. 

140
Figure 7.9: Bias and variance differences between ensemble and individual models including RkDB.
7. NEW ENSEMBLE APPROACHES FOR BNCS
Appendix A

R package: exreport

Although it is not directly connected with the topic of this thesis, I would like to highlight an additional research results that I have developed along the different publications and research projects. This software is co-authored with Javier Cózar del Olmo.

This R package is aimed at defining and obtaining robust and transparent reproducible reports to analyse experimental data, with a focus on machine learning algorithm comparison.

The inspiration comes from the methodology proposed in Demšar [2006] and followed by García & Herrera [2008], as well as the principles of tidy data introduced by the influential Hadley Wickham in his paper Wickham et al. [2014].

Exreport proposes a methodology to define an hypothesis, collect data and generate statistical analysis and reports using a reproducible approach in the R language. It requires just a very simple specification of a document including various statistical tests, tables and graphics from a given experimental dataset. The dataset will contain methods, problems as well as parameters and outputs that will be analysed and rendered in a very friendly format with the possibility of exporting them directly to a scientific publication.

This specification can be shared along with such publication and the experimental data to allow other researchers to explore the results further or to answer their own questions, something that is not extended enough in the community.

This dissertation has used exreport intensively to produce every statistical test as well as related plots and tables and will be published with the corresponding data and exreport scripts as an example of how to reproduce experimental results.
using this tool.

Exreport is open source and is available on CRAN https://cran.r-project.org/web/packages/exreport/index.html and github https://github.com/jacintoArias/exreport. Visit http://exreport.jarias.es, where a comprehensive tutorial and examples are included as well.
Appendix B

Publications

Part of the contents discussed in this dissertation has been presented in the following papers and conference contributions.


- Jacinto Arias, José A. Gámez, Thomas D. Nielsen and José M. Puerta. A Scalable Pairwise Class Interaction Framework for Multi-dimensional Clas-


• Jacinto Arias, José A. Gámez, and José M. Puerta. Un análisis crítico del clasificador AkDE como ensemble y sus implicaciones para tratar con grandes volúmenes de datos. Conference of the Spanish Association for Artificial Intelligence. Granada, Spain. October 2018.
References


Arias, J., Gámez, J.A. & Puerta, J.M. (2013). Learning more accurate Bayesian networks in the CHC approach by adjusting the trade-off between
efficiency and accuracy. In Conference of the Spanish Association for Artificial Intelligence, 310–320, Springer. 37


REFERENCES


BREIMAN, L. (2001). Random forests. Machine Learning, 45, 5–32. 120, 123
REFERENCES


REFERENCES


REFERENCES


153
REFERENCES


JOHN WALKER, S. (2014). Big data: A revolution that will transform how we live, work, and think. 34


REFERENCES


156
REFERENCES


REFERENCES


