

Incremental Bayesian network structure learning from data streams

Amanullah Yasin

▶ **To cite this version:**

Amanullah Yasin. Incremental Bayesian network structure learning from data streams. Machine Learning [cs.LG]. Université de Nantes, 2013. English. NNT : . tel-01284332

HAL Id: tel-01284332
https://theses.hal.science/tel-01284332

Submitted on 7 Mar 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

l'université nantes angers le mans
PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Amanullah YASIN

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université de Nantes
sous le label de l'Université de Nantes Angers Le Mans

Discipline : Informatique
Spécialité : Informatique
Laboratoire : Laboratoire d'informatique de Nantes-Atlantique (LINA)

Soutenue le 08 Octobre 2013

École doctorale : 503 (STIM)
Thèse n° : ED 503-198

Incremental Bayesian network structure learning from data streams

JURY

Rapporteurs :	**M. Florent MASSEGLIA**, Chargé de Recherche, INRIA **M. Ioannis TSAMARDINOS**, Associate Professor, University of Crete, Greece
Examinateurs :	**M. Marc GELGON**, Professeur des universités, Université de Nantes **M. Pierre-François MARTEAU**, Professeur des universités, Université de Bretagne Sud **M. Karim TABIA**, Maître de conférences, Université d'Artois
Directeur de thèse :	**M. Philippe LERAY**, Professeur des universités, Université de Nantes

Acknowledgments

"*I never guess. It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts.*" - Sir Arthur Conan Doyle

Firstly of all I am extremely thankful to Allah Almighty for giving me the strength and fortitude to accomplish this milestone.

It is with immense gratitude that I will forever be thankful to my PhD advisor, Professor Philippe Leray. His technical guidance, encouragement and moral support from the preliminary to the concluding level enabled me to enhance my subject knowledge and polish my research skills. Their valuable supervision will enable me to undertake challenging research problems in the future.

I would like to thank Higher Education Commission (HEC), Pakistan and the University of Nantes for giving me opportunity to do PhD at this premier institution and to HEC in particular for funding my research. I am also thankful to my research lab LINA and Computer Science department. I am very thankful to Prof. Pascal Kunz, Prof. Fabrice Guillet, Mr. Julien Blanchard, Mr. Fabien Picaro and office staff for their administrative support during my study in the department.

I also acknowledge Mr. João Gama from Laboratory of Artificial Intelligence and Decision Support, and Faculty of Economics, University of Porto, Portugal, for insightful discussions we had together on data stream mining.

This acknowledgment would not be complete without mentioning my colleagues from the Polytech Nantes and the HEC scholar community. Their companionship has been a source of great relief and entertainment in this intellectually challenging journey.

Last but not least, I would not have been standing at the finish line had it not been for the selfless love and prayers of my parents and my wife. Their affection and encouragement helped me pass through the thick and thin.

I fully understand that my memory serves me well only to a certain extent rendering a limitation to my recollection of the good deeds that many other people have done to and for me. For those who I have failed to mention, I sincerely apologize, and I thank you from the bottom of my heart.

To my kids, ***Hassan*** *and* ***Ammar****,*
and my little doll, ***Umaimah***

Contents

1 Introduction **3**

1.1 Motivation . . . 3
1.2 Problem statement . . . 5
1.3 Scope of the thesis . . . 6
1.4 Structure of the thesis . . . 8
1.5 Publications . . . 9

I State of the art **11**

2 Data stream mining **13**

2.1 Introduction . . . 14
2.2 Data stream . . . 14
2.2.1 Properties of a data stream . . . 14
2.2.1.1 Data access . . . 15
2.2.1.2 Unboundness . . . 15
2.2.1.3 Evolving nature . . . 15
2.3 Data stream handling techniques . . . 17
2.3.1 Data based stream handling . . . 18
2.3.1.1 Vertical Summary . . . 19
2.3.1.2 Horizontal Summary . . . 19
2.3.2 Task based stream handling . . . 20
2.3.2.1 Window model . . . 20
2.3.2.2 Approximation algorithms . . . 23
2.3.2.3 Algorithm output granularity . . . 23
2.4 Data stream mining . . . 23
2.4.1 Challenges for data stream mining . . . 24
2.4.2 Usual data stream mining techniques . . . 24
2.4.2.1 Clustering . . . 25
2.4.2.2 Classification . . . 25
2.4.2.3 Frequent pattern mining . . . 26
2.5 Applications . . . 28

2.6 Conclusion 29

3 Probabilistic graphical models **31**

3.1 Introduction 33
3.2 Preliminaries 33
3.2.1 Probability and information theory 33
3.2.2 Graph theory 35
3.3 Probabilistic graphical models 36
3.3.1 Markov network 36
3.3.2 Bayesian network 37
3.3.3 Some principles of Bayesian network 38
3.3.3.1 D-separation 38
3.3.3.2 Markov equivalent class 39
3.3.4 Querying a distribution 40
3.3.4.1 Exact inference 41
3.3.4.2 Approximate inference 42
3.4 Learning 42
3.4.1 Parameter learning 43
3.4.1.1 Maximum likelihood estimation (MLE) 43
3.4.1.2 Bayesian estimation 43
3.4.2 Structure learning 44
3.4.2.1 Score-and-search based methods 44
3.4.2.2 Constraint based methods 48
3.4.2.3 Hybrid learning 50
3.4.3 Evaluating structural accuracy 52
3.5 Problem of high dimensional domain 53
3.6 Conclusion 54

4 Bayesian networks and data streams **55**

4.1 Introduction 56
4.2 Learning from stationary domains 57
4.2.1 Score-and-search based methods 57
4.2.1.1 Buntine's approach 57
4.2.1.2 Lam and Bacchus's approach 60
4.2.1.3 Friedman and Goldszmidt's approach 64
4.2.1.4 Roure's approach 69
4.2.2 Hybrid method 75
4.3 Learning from non-stationary domains 78
4.4 Conclusions 83

II Propositions **85**

5 Incremental Max-Min hill climbing **87**

5.1 Introduction . . . 89
5.2 Constraint based incremental local search . . . 90
5.2.1 Principle . . . 90
5.2.2 Objective . . . 90
5.2.3 Incremental MMPC . . . 90
5.2.4 Incremental $\overline{\mathbf{MMPC}}(\mathbf{T})$ . . . 92
5.2.4.1 Forward phase of $\overline{MMPC}(T)$ as a greedy search 92
5.2.4.2 Algorithm . . . 94
5.2.4.3 An illustrative example . . . 97
5.2.5 Complexity . . . 99
5.3 Incremental Max-Min Hill-Climbing . . . 100
5.3.1 Principle . . . 100
5.3.2 Objective . . . 101
5.3.3 Incremental local search . . . 101
5.3.4 Incremental global optimization . . . 101
5.3.5 Comparison with existing methods . . . 103
5.3.6 Optimizing frequency counting . . . 103
5.3.7 Complexity analysis . . . 105
5.4 Adaptation of iMMHC for sliding window . . . 106
5.4.1 Principle . . . 106
5.4.2 Objective . . . 107
5.4.3 Update of sufficient statistics . . . 107
5.4.4 Complexity . . . 108
5.5 Adaptation of iMMHC for damped window . . . 109
5.5.1 Principle . . . 109
5.5.2 Objective . . . 109
5.5.3 Forgetting mechanism . . . 109
5.6 Conclusions . . . 110

6 Experimentation **113**

6.1 Introduction . . . 114
6.2 Evaluation measures . . . 114
6.2.1 Computational efficiency . . . 114
6.2.2 Model accuracy . . . 115
6.3 Experimental protocol . . . 115
6.3.1 Benchmarks . . . 115
6.3.2 Algorithms . . . 116

6.4 Results and interpretations with landmark windows 117
6.4.1 Parameters analysis 117
6.4.2 iMMHC for incremental learning 120
6.4.3 Incremental versus batch MMHC 123
6.4.4 iMMHC for high dimensional domains 123
6.5 Results and interpretations with sliding window 124
6.6 Results and interpretations with damped window 126
6.6.1 Non-stationary domains: 126
6.6.2 Stationary domains: 128
6.7 Conclusions 128

III Conclusion and perspectives 131

7 Conclusion and perspectives 133
7.1 Summary 133
7.2 Perspectives and open problems 135

Appendices

A Comparison table for different data stream mining techniques 139

B Résumé en langue française 145
B.1 Introduction 146
B.2 Approches existantes 147
B.3 Contexte 149
B.3.1 MMPC(T): recherche locale pour l'apprentissage de la structure d'un réseau bayésien 149
B.3.2 Adaptation incrémentale des méthodes d'apprentissage à base de score 150
B.4 Une approche locale pour l'apprentissage incrémental 152
B.4.1 Interprétation "gloutonne" de la phase *forward* de $\overline{MMPC(T)}$ 152
B.4.2 Notre proposition : *MMPC(T)* incrémental 154
B.4.3 Exemple jouet 155
B.5 Notre proposition : *MMHC* incrémental 157
B.5.1 Une approche locale pour l'apprentissage incrémental: . . . 157
B.5.2 La construction incrémentale d'un réseau bayésien global: . 158
B.5.3 Optimisation de comptage de fréquence 158
B.6 L'adaptation *MMHC* incrémental sur "Sliding Window" 160
B.6.1 Mise à jour des statistiques suffisantes 160
B.7 L'adaptation *MMHC* incrémental sur "Damped Window" 161
B.7.1 Mécanisme de l'oubli 162

B.8 Etude expérimentale 162
B.8.1 Protocole expérimental 162
B.8.2 Résultats et interprétations: 165
B.8.2.1 Analyse des paramètres 165
B.8.2.2 iMMHC pour l'apprentissage incrémentale 167
B.8.2.3 Incrémentale contre MMHC batch 167
B.8.2.4 iMMHC pour les domaines de grande dimension . 168
B.8.2.5 Résultats et interprétations avec "sliding window" 168
B.8.2.6 Résultats et interprétations avec"damped window" 169
B.9 Conclusion 170

Notations

D	Data set (without missing values)
w_i	i^{th} time window in data stream
D_i	Data set in i^{th} time window
D'	New data
N	Number of examples in data
n	Number of attributes or variables in data
X, Y, Z	A Generic random variable or an event or a node of a graph
x, y, z	Particular value may taken by the corresponding variables
$\mathbf{X}, \mathbf{Y}, \mathbf{Z}$	Set of variables or nodes of a graph
$\mathcal{X}$	Domain of a variable X
$\mathcal{G}$	Graph
V	Set of vertices or nodes of $\mathcal{G}$
E	Set of edges or links of $\mathcal{G}$
$Pa(X_i)$	Parents of node i in graph $\mathcal{G}$
N_{ijk}	Number of examples where set of $Pa(X_i)$ takes its j^{th} configuration while the variable X_i takes its k^{th} configuration in the database D
P	A probability distribution
$P(X)$	Probability of X
θ	Parameters of a Bayesian network
M	A Bayesian network model $\langle \mathcal{G}, \theta \rangle$
$\mathcal{I}$	Independence test
r	Cardinality, number of configurations for a variable

1

Introduction

1.1 Motivation

There has been a rapid evolution of the hardware and software technologies over recent years. Nowadays growing number of applications generating massive streams of data, which need intelligent data processing and online analysis. Telecommunication, real-time surveillance systems, sensor networks, set of retail chain transactions, social networks and bioinformatics are some examples of such applications. In these applications, data is generated continuously and supplied to decision support systems. These systems have to update their existing knowledge in the light of novel data. The challenges for data stream mining systems involve storage, querying and mining. Therefore, incremental data mining has been an active area of research. The problem aggravates for incoming data with high dimensional domains (several hundreds or thousands of variables).

An analysis of the existing literature reveals that traditional data mining techniques do not directly apply to the data stream. These techniques consider a batch learning scenario where the whole dataset is available for the algorithms. On the contrary, the scenario where dataset is growing continuously and we cannot wait for its completion, the obtained model at a certain time will be outdated. So, revising

the existing model by re-executing the batch learning algorithm will not only take a lot of resources as well as it is also a costly procedure. For this reason, incremental learning algorithms are needed in order to efficiently integrate the novel data with the existing knowledge. These algorithms must be capable to incorporate new data, forgetting outdated data and adopt most recent states of the data. Now we extend our considerations in the perspective of probabilistic graphical models and especially in the context of Bayesian networks.

The graphical models are introduced by *Jordan* as:
Graphical models, a marriage between probability theory and graph theory, provide a natural tool for dealing with two problems that occur throughout applied mathematics and engineering–uncertainty and complexity. In particular, they play an increasingly important role in the design and analysis of machine learning algorithms. Fundamental to the idea of a graphical model is the notion of modularity: a complex system is built by combining simpler parts. Probability theory serves as the glue whereby the parts are combined, ensuring that the system as a whole is consistent and providing ways to interface models to data. Graph theory provides both an intuitively appealing interface by which humans can model highly interacting sets of variables and a data structure that lends itself naturally to the design of efficient general-purpose algorithms [79].

Bayesian networks (BNs) are a family of probabilistic graphical models (PGMs) and a powerful tool for graphical representation of the underlying knowledge in the data. Bayesian networks, which were named after Thomas Bayes (1702-1761), one of the founders of the probability theory. The importance of Bayesian networks highlighted by *Darwiche* in his article Bayesian networks [36] as:

What are Bayesian networks and why are they widely used, either directly or indirectly, across so many fields and application areas? Intuitively, Bayesian networks provide a systematic and localized method for structuring probabilistic information about a situation into a coherent whole. They also provide a suite of algorithms that allow one to automatically derive many implications of this information, which can form the basis for important conclusions and decisions about

the corresponding situation.

Bayesian networks are used in several applications like assistance for blind people [89], weather prediction [30], junk e-mail filtering [125], image analysis for tactical computer aided decision [45], user assistance in software use [72], fraud detection [44] etc. A Bayesian network is graphically represented by a directed acyclic graph (DAG), where nodes of the DAG represent the random variables and every edge of the DAG represents the direct influence of one variable to the other. Structure learning is the task corresponding to determining the best DAG for a given dataset.

1.2 Problem statement

Data stream mining means extracting useful information or knowledge structures from a flow of data. It becomes the key technique to analyze and understand the nature of the incoming data. Typical data mining tasks including association mining, classification, and clustering, help to find interesting patterns, regularities, and anomalies in the data. However, traditional data mining techniques cannot directly apply to data streams. This is because most of them require multiple scans of data to extract the information, which is unrealistic for data stream. More importantly the characteristics of the data stream can change over time and the evolving pattern needs to be captured [55]. Furthermore, we also need to consider the problem of resource allocation in mining data streams. Due to the large volume and the high speed of streaming data, mining algorithms must cope with the effects of system overload. Thus, how to achieve optimum results under various resource constraints becomes a challenging task. The main objective of this task is to minimize the resource allocation as compared to batch approach and optimize the accuracy of the resulting model.

In the light of bibliographic study we identify the following main constraints in data stream mining:

Drift Detection: Detecting the change in underlying distribution of data is called

drift detection. It is helpful to decide whether we need to update our existing model or not and when and which part of the model needs to be updated.

Time Complexity: As we are dealing with the continuous arrival of large amount of data in the form of data streams, it needs an efficient algorithm which can update in limited time.

Memory Complexity: Due to the online nature of the data stream, it is not feasible to store it on secondary storage and then process. Because multiple scanning of such data is unfeasible, sufficient statistics are necessary to maintain in the memory for further use.

This continuous nature of the data have inspired a great deal of research into performing the classical data mining techniques on data streams. In incremental Bayesian network structure learning field, there are few approaches that have been proposed in the literature. These approaches are taking into account certain scenarios depending upon the nature of data and structure learning methodology. The most of the algorithms use *scoring-and-search* based methods and treat the incremental process in different ways for *stationary domains*. *Score-and-search* based methods are not scalable for data stream mining, the judicious choice for handling a great number of variables is a local search hybrid approach. There is an encouraging trend to find an approach that might be applicable to high dimensional domains, which deal with the accuracy of final model and the learning cost. The existing state of the art in incremental structure learning involves only several tens of variables and they do not scale well beyond a few tens to hundreds of variables.

1.3 Scope of the thesis

This research study is sponsored by Higher Education Commission of Pakistan (HEC), under scientific and technical cooperation program between Government of France and Pakistan. The objective of the study is to explore existing state of the art in data stream mining and to propose a solution for Bayesian network structure learning from data stream. In the light of existing state of the art, figure 1.1 shows

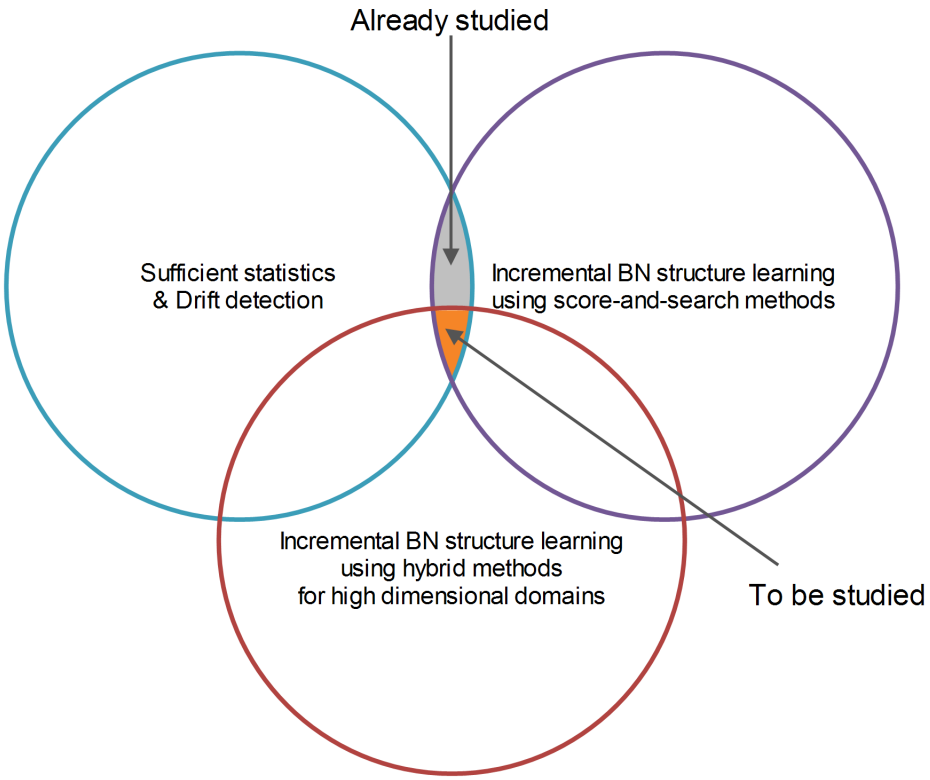


Figure 1.1: *Scope of the thesis*

the objective of this study.

Based on the analysis of already existing solutions the main contributions of this study in incremental Bayesian network structure learning include:

- A constraint based incremental local search algorithm ($iMMPC$) to identify the set of candidate parent children (CPC) of a target variable in any Bayesian network, which faithfully represents the distribution of data. We applied incremental hill-climbing principle to find a set of CPCs for a target variable and observed that it saves considerable amount of computational complexity.
- Incremental Max-Min Hill-Climbing ($iMMHC$), a novel method for incremental Bayesian network structure learning from high dimensional domains over a *landmark window*. We adopted the principle of *Max-Min hill-climbing* algorithm [138], one of the most robust structure learning algorithms for high dimensional domains.
- Frequency Counter, a middle layer between the learning algorithms and the data. It saves lot of computing time and speeds up the learning process.

- A solution to adapt $iMMHC$ over sliding window. It stores only minimum possible sufficient statistics from the previous data, rather than storing the whole data stream.
- A solution to adapt $iMMHC$ over damped window by introducing a forgetting factor to keep model up to date with recent trends of data and forgetting outdated informations.

1.4 Structure of the thesis

This manuscript is composed of five major chapters excluding, chapter 1, which gives a general introduction of the work and chapter 7, which summarizes the whole work and presents some perspectives for future work.

Chapter 2

An overview of data stream mining and usual techniques handling data stream are discussed in chapter 2. We also try to identify the main challenges for learning algorithms to deal with data streams. Finally, we show some typical data mining techniques such as *clustering*, *classification* and *frequent pattern mining* dealing with incremental environment.

Chapter 3

Chapter 3 is dedicated to introduce some basics of probabilistic graphical models. It provides background on Bayesian networks and motivate the local search based hybrid structure learning paradigm as it can scale up for high dimensional domains.

Chapter 4

In preparation for our approach to incremental Bayesian network structure learning, chapter 4 presents existing state of the art algorithms dealing with data streams. There are a few approaches in this category and most of them use *score-and-search* based structure learning technique.

Chapter 5

This chapter provides first comprehensive hybrid solution for Bayesian network

structure learning from data stream over landmark and sliding window. We introduce a frequency counter technique to speed up the learning process. Finally, we adapt our method for non stationary domains and introduce a forgetting factor to keep model up to date with recent data.

Chapter 6

This chapter provides an empirical evaluations for our propositions of $iMMHC$, frequency counter and using sufficient statistics. The results show that our method is significantly better than existing state of the art methods and justifies its effectiveness for incremental use.

Chapter 7

Finally, the work concludes in chapter 7 presenting a number of conclusions drawn from the study. Recommendations for future work in continuity of this work will also be presented.

1.5 Publications

First overview of incremental constraint based local search has appeared in the following publications :

Yasin, A. and Leray, P. (2011b). Local skeleton discovery for incremental Bayesian network structure learning. In *Proceedings of the 1st IEEE International Conference on Computer Networks and Information Technology (ICCNIT)*, pages 309-314, Peshawar, Pakistan.

Incremental Max-Min Parent Children algorithm has first been presented in :

Yasin, A. and Leray, P. (2011a). iMMPC: A local search approach for incremental Bayesian network structure learning. In Gama, J., Bradley, E., and Hollaé, J., editors, Advances in Intelligent Data Analysis X, *Proceedings of the Tenth International Symposium on Intelligent Data Analysis (IDA 2011), volume 7014 of Lecture Notes in Computer Science,* pages 401-412. Springer Berlin / Heidelberg.

A short summary in French language of Incremental Max-Min Parent Children algorithm has appeared in the following publication :

> Yasin, A. and Leray, P. (2012). iMMPC: apprentissage incrémental local de réseaux Bayésiens. In 12^e *Conférence Internationale Francophone sur l'Extraction et la Gestion des Connaissances (EGC2012),* pages 587-588, Bordeaux, France.

Incremental Max-Min Hill-Climbing approach has been developed in :

> Yasin, A. and Leray, P. (2013). Incremental Bayesian network structure learning in high dimensional domains. *In Proceedings of the 5th International Conference on Modeling, Simulation and Applied Optimization (ICMSAO 2013),* pages 1-6, Hammamet, Tunisia.

I

State of the art

2

Data stream mining

Contents

2.1 Introduction 14
2.2 Data stream 14
2.2.1 Properties of a data stream 14
2.2.1.1 Data access 15
2.2.1.2 Unboundness 15
2.2.1.3 Evolving nature 15
2.3 Data stream handling techniques 17
2.3.1 Data based stream handling 18
2.3.1.1 Vertical Summary 19
2.3.1.2 Horizontal Summary 19
2.3.2 Task based stream handling 20
2.3.2.1 Window model 20
2.3.2.2 Approximation algorithms 23
2.3.2.3 Algorithm output granularity 23
2.4 Data stream mining 23
2.4.1 Challenges for data stream mining 24
2.4.2 Usual data stream mining techniques 24
2.4.2.1 Clustering 25
2.4.2.2 Classification 25
2.4.2.3 Frequent pattern mining 26
2.5 Applications 28
2.6 Conclusion 29

2.1 Introduction

In this chapter we provide an overview of *data stream mining*. We divide it in two parts. In the first part, we discuss about the structure and characteristics of a data stream, also we see the different analysis methods and handling techniques of unbounded data streams. Later, we give the main issues to the mining of data stream.

In the second part, we discuss how different data mining techniques have been applied to data stream, to extend our considerations to build incremental learning model for Bayesian network structure.

2.2 Data stream

Many sources produce data by gathering continuous information over long period of time i.e. customer click streams [34], sensor data [56], network traffic monitoring [7], and sets of retail chain transactions [146] etc. Thus this new kind of data is called data stream.

A data stream is a continuous sequence of items generated in real time. It can be considered as a table of infinite size. The order of the incoming items is defined with respect to time and it can not be controlled. Similarly, it is not possible to store the entire sequence (stream) in the memory. These characteristics made a new area of research in the learning field that is incremental or online learning.

2.2.1 Properties of a data stream

The data stream environment differs from conventional database systems to some extents. A summary of the differences between traditional and stream data processing is presented in Table 2.1. There are several data stream properties discussed in the literature, review articles and surveys [1, 9, 76, 56]. Here we can summarize these properties in three categories: data access, unboundness and evolving nature.

	Traditional	Stream
Number of passes	Multiple	Single
Processing Time	Unlimited	Restricted
Memory Usage	Unlimited	Restricted
Type of Result	Accurate	Approximate
Distributed	No	Yes

Table 2.1: *Differences between traditional and stream data processing [56]*

2.2.1.1 Data access

In conventional database systems whole datasets are available for analysis. Algorithm can access it multiple time as well as on random basis. On the other hand in data stream environment, the applications or the systems are continuously generating huge amount of data. Therefore, in some cases it is not feasible to store all data on physical memory. Hence, the learning algorithms could not do multiple scan and also it is expensive to do a random access.

Therefore, the data streams are real time in nature, which gives a restricted time for the analysis algorithms.

2.2.1.2 Unboundness

The data stream is considered as an unbounded set of examples that is continuously updated. Neither, we can wait to finish the data stream, nor we store the whole stream in the memory. Therefore, the amount of computation time for learning algorithms should be low.

2.2.1.3 Evolving nature

As data streams are real time and unboundness in their nature, the order of the incoming events could not be controlled. Therefore, the behavior of the data could also be changed over time as well as the underlying probability distribution could evolve over time. Figure 2.1 shows an example of drift in classification. We can see how data points in each time window affected by decision boundaries. Therefore, algorithms trained with past test data will no more be valid, as well as the error rate obtained by training dataset can be changed on future data. So, the stream analysis

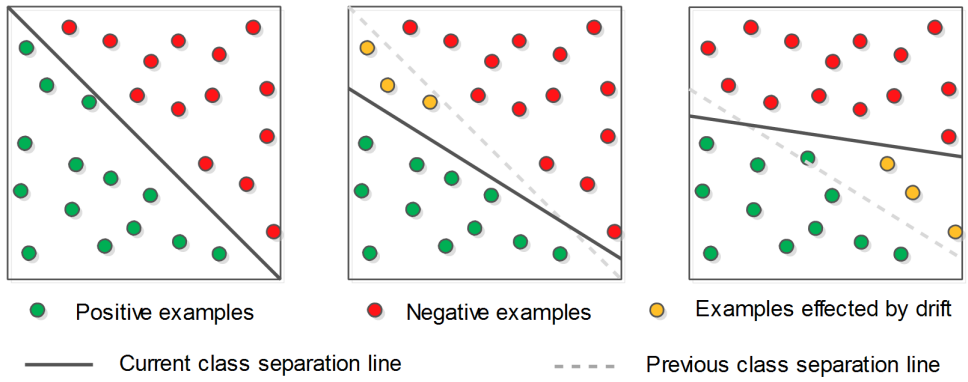


Figure 2.1: *An Example of drift in classification with recently observed data examples in different time windows and the associated decision boundaries.*

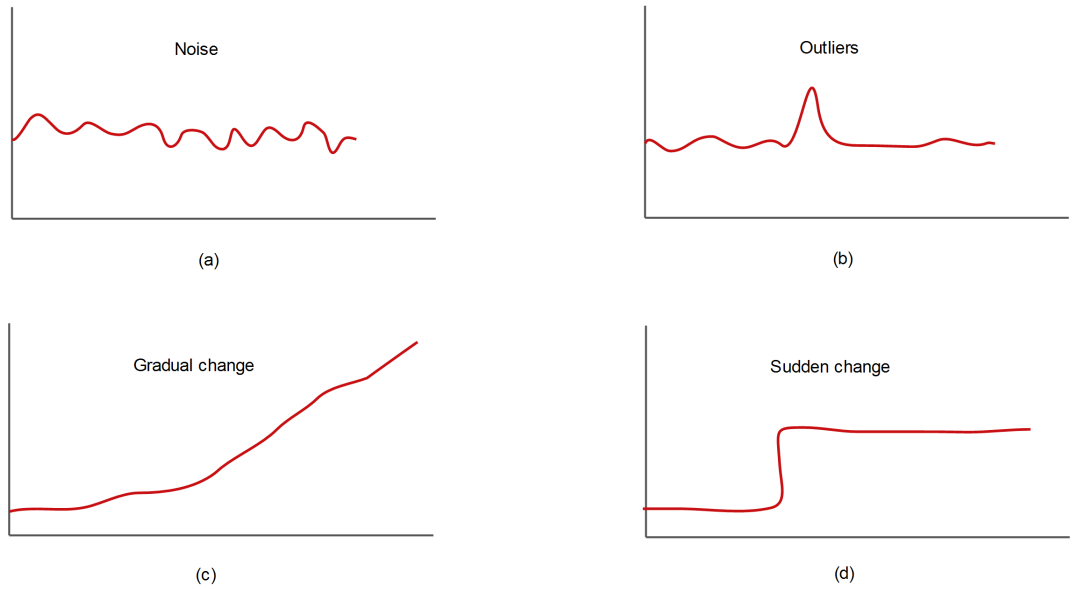


Figure 2.2: *Some change types in data stream*

algorithms should be capable to adopt these changes. The type of the change could be a noise, outliers, rare event (intrusion), gradual change or sudden change (cf fig. 2.2).

In the literature, there is a complete study address this issue, both in learning under evolving conditions [73, 42, 54, 147], as well as identifying the evolution [57, 84, 96, 103] in machine learning and statistical community.

Because of these characteristics of data stream, it requires specialized algorithms which are capable of incremental and adaptive processing of unbounded amount of

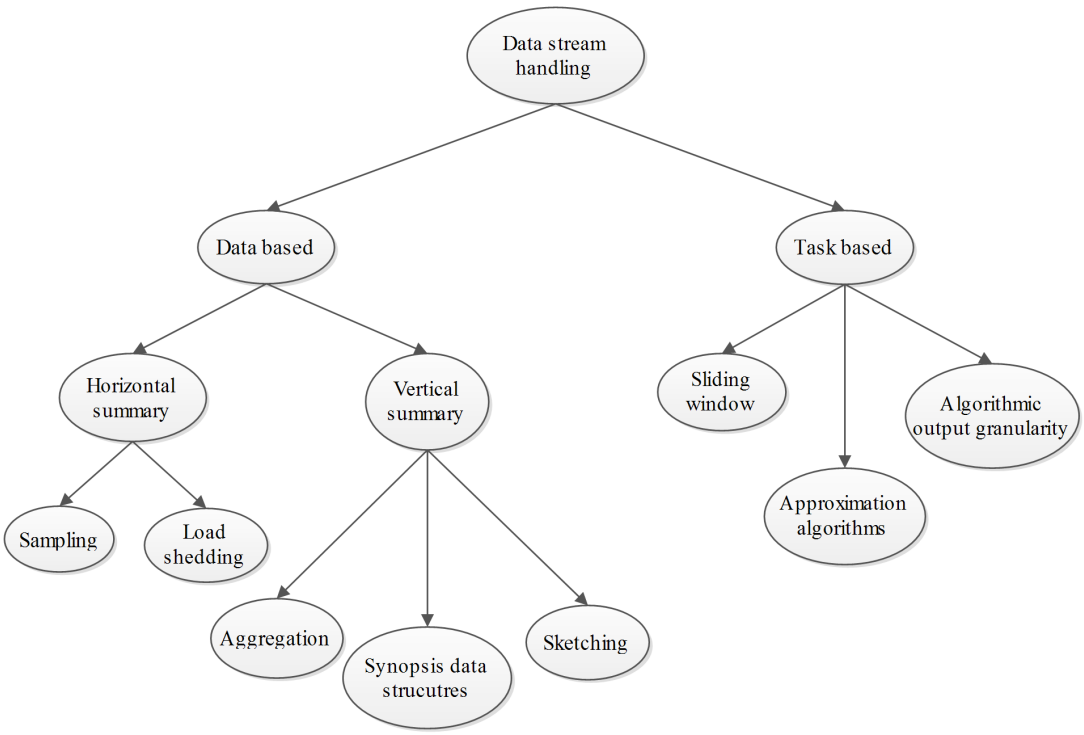


Figure 2.3: *Classification of data stream processing methods (adapted from [80]).*

data using a controlled environment like limited amount of memory and available time.

2.3 Data stream handling techniques

As we discussed in the previous section 2.2.1 that data streams are naturally unbounded. This property creates time and memory limitations which impose several questions for learning algorithms. To handle these resource constraints, many solutions have been proposed in the literature. Gaber [54] divided these solutions in two categories: data based and task based. Data based techniques involve preprocessing of data stream by summarizing or choosing a subset of the incoming stream. Later, this summary could be used for analysis. On the other hand, task based solutions modify existing methods or invent new one in order to address the limitations of data stream processing. Both the above mentioned techniques have been illustrated in the following sections 2.3.1 and 2.3.2.

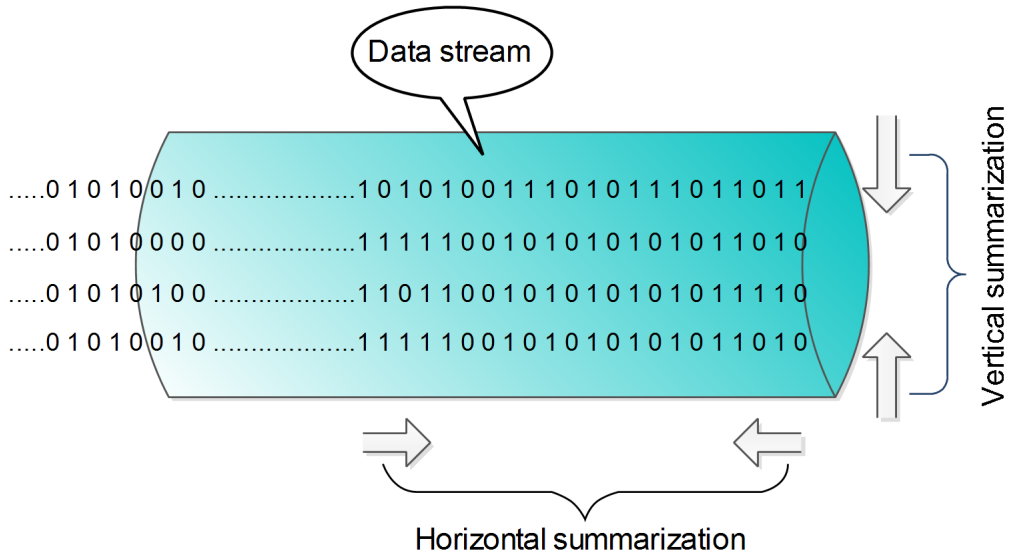


Figure 2.4: *Horizontal and vertical summarizing techniques for a data stream.*

2.3.1 Data based stream handling

In this section, we will discuss data based techniques for handling data stream like sketching, sampling, aggregation, load shedding and synopsis data structures. These methods provide means to examine only a subset of the whole dataset or to transform the data *vertically* or *horizontally* to an approximate smaller size data representation. Therefore, known data mining techniques can be used. In this section we review these theoretical foundations by dividing them in *horizontally* and *vertically* summarizing techniques(cf. 2.4). To make summary query-able for any past portion, it requires keeping historical summary of the contents of the stream with the following requirements [66]:

- It should be non-specialized, i.e. can be used for both purposes, either for answering queries or applying mining algorithms.
- It must have a fixed or low increasing size.
- The construction of the summary must be incremental (done on the fly).
- The amount of CPU used to process each item of the streams must be compatible with the arrival rate of items.
- It must cover the whole stream and enable to build summaries of any past part of the history of the stream.

2.3.1.1 Vertical Summary

Vertically summarization techniques summarize the whole incoming stream of instances or variables e.g. sketching and principal component analysis.

Sketching is a vertical summarization process of data stream for better memory management. It is the process of vertically sampling the incoming stream by randomly project a subset of features [9, 105].

Dobra et al. in [41] demonstrated that sketching can be generalized to provide approximate answers to complex, multi-join, aggregate SQL queries over streams. Whereas, it is not feasible to use sketching for data mining context. The accuracy is also a major drawback.

Principal component analysis (PCA) could be a better solution for data mining context. It involves a mathematical procedure that transforms a number of possibly correlated variables into a smaller number of uncorrelated variables called principal components. It is a singular value decomposition (SVD) of the covariance matrix between variables. Covariance matrix can be computed incrementally. This solution is applicable to any additive or pseudo-additive data mining method [67].

2.3.1.2 Horizontal Summary

These techniques summarize the observations of the individuals [76].

Sampling It is the process of statistically selecting the elements of the incoming stream that would be analyzed. It represents a large dataset by a small random sample. In [43], Domingo et al. studied sampling approach to tackle decision tree classification and k-means clustering. Sampling plays an important role in developing techniques for clustering data streams [64]. Sampling does not address the problem of fluctuating data rates. When using sampling, it would be worth investigating the relationship among the three parameters: data rate, sampling rate and error bounds [54].

Load-shedding Load shedding refers to the process of eliminating a batch of subsequent elements (randomly or semantically) from being analyzed [21]. It has the same problems of sampling. Load shedding is not a preferred approach with mining algorithms, especially in time series analysis because it drops chunks of data streams that might represent a pattern of interest. Still, it has been successfully used in sliding window aggregate queries [10].

Aggregation Aggregation is the representation of number of elements in one aggregated element using some statistical measure such as means, variance or the average. It is often considered as a data rate adaptation technique in a resource-aware mining [52]. The problem with aggregation is that it does not perform well with highly fluctuating data distributions. Algorithms over data streams that utilize aggregation include approximate quintiles [101].

Synopsis Data Structures Synopsis data structures represent the idea of small space and an approximate solution to massive dataset problems. Creating synopsis of data refers to the process of applying summarization techniques that are capable of summarizing the incoming stream for further analysis. Wavelet analysis, histograms, and frequency moments have been proposed as synopsis data structures [62, 9].

2.3.2 Task based stream handling

In this section, we will discuss the task based techniques to overcome the limitations of data stream processing. In this category there are window model, approximation algorithm and algorithm output granularity. In the following sections we examine each of these techniques.

2.3.2.1 Window model

Window model is an approximation technique. Mostly, data mining tasks applied to data streams use one of the window models. We are interested in recent data points rather than old ones because they are more relevant than the older data. Therefore,

we compute statistics over window model rather than computing over whole stream. It is extremely used in data stream mining field. Several window model have been presented in the literature [150]. The window model on data stream mining can be divided into four categories:

Sliding window

In sliding window model, data mining task is conducted over a fixed number of recently arrived tuples and they are the target of mining. Window serves as first-in and first-out data structure (cf fig. 2.5 (a)). The objective is to maintain the size of main memory for data stream processing.

Landmark window

In landmark window model, knowledge discovery is performed based on the values between a specific time-points called landmark and the present as shown in the figure 2.5 (b).

Tumbling window

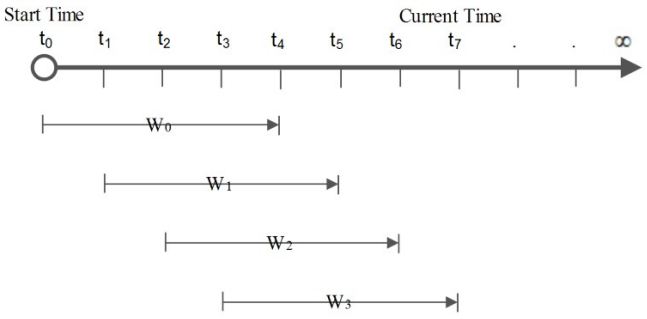
In tumbling window all tuples within the window expires at the same time as shown in the figure 2.5 (c).

Damped window

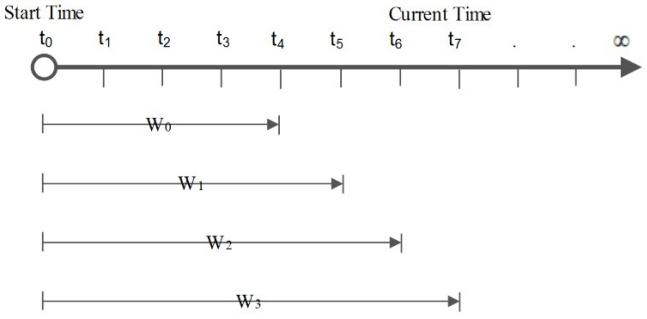
In damped window model, recent data is more important than previous ones, and the weights of data decreases exponentially into the past [143, 85]. It is based on the fact that users are more interested in the recent trends. As shown in figure 2.5 (d) if w_i is the ith window and α is the forgetting factor (weight) then data in each window could be found as:

$$\sum_{i=0}^{CurrentTime} (w_i \times \alpha^{CurrentTime-i}) \tag{2.1}$$

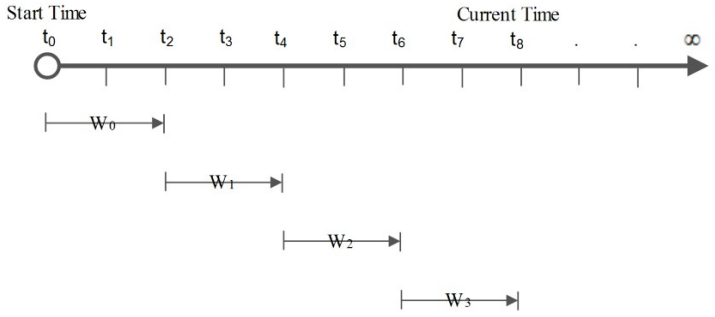
A second way is to multiply each transaction with the forgetting factor so, it will add an exponentially decay factor to the data stream [143].



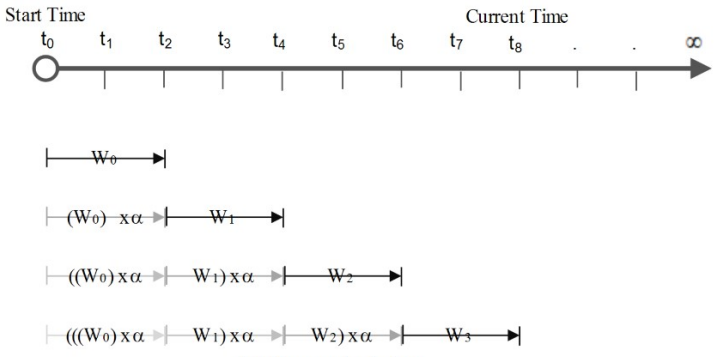
(a) Sliding window



(b) Landmark window



(c) Tumbling window



(d) Damped window

Figure 2.5: *Different window models*

2.3.2.2 Approximation algorithms

Approximation algorithms directly addressed the data mining challenges within their algorithmic design [105]. These algorithms can produce approximate solutions with error bounds. However, the problem of data rates with the available resources could not be solved using approximation algorithms. So, it requires to use supplementary techniques to adopt the available resources. Approximation algorithms have been used in [33].

2.3.2.3 Algorithm output granularity

The algorithm output granularity (AOG) techniques are resource aware data analysis approaches [52, 51, 53]. It can handle very high fluctuating data streams, according to the available memory and time constraints. These methods performs local data analysis on the stream generating or receiving devices. The main processing stages are: mining, adaptation to the resources and data stream rate and merging the obtained knowledge structures, if processing going to exceed resource limitation. AOG methods has been successfully used in clustering, classification and frequency counting [51].

2.4 Data stream mining

This section provides an overview of goals and challenges in data stream mining as well as existing techniques. First, we see what is data mining?

Data mining, meaning extracting useful information or knowledge from large amounts of data, has become the key technique to analyze and understand data. Typical data mining tasks include *association mining*, *classification*, and *clustering*.

In general, the objective of all techniques is to find the interesting patterns, correlations, regularities, and anomalies in the data. The particular goal of individual technique will be briefly discussed in section 2.4.2.

2.4.1 Challenges for data stream mining

Early research in data mining and machine learning usually focused on batch learning using small datasets. In batch learning environment whole dataset is available for learning algorithms and most of them extract information or generate decision models by scanning the dataset multiple times. Which is unrealistic for data stream due to their transient nature. The amount of previously happened events is usually overwhelming, so they can be either dropped after processing or archived separately in secondary storage.

More importantly, for large time periods in complex systems, we expect the characteristics of the data stream can change over time and the evolving pattern needs to be captured. Furthermore, we also need to consider the problem of resource allocation in mining data streams. Due to the large volume and the high speed of streaming data, mining algorithms must cope with the effects of system overload. A natural approach for these incremental tasks is adaptive learning algorithms.

As a conclusion, the challenging problem in data stream mining is to permanently maintain an accurate decision model. This issue requires learning algorithms that can adapt new data and modify current model according to the new data in available time and memory.

2.4.2 Usual data stream mining techniques

Much work has been done to extend some standard data mining algorithms to analyze rapidly arrived data in real time. The aim of all these algorithms is to modify or evolve an already known structure when new data are available. In this section, we discussed common data stream mining techniques: clustering, classification and association rule mining techniques. A summarized comparison of these techniques can be find in Annex A.

2.4.2.1 Clustering

Clustering is the task of grouping the object having same properties, so that the elements in each group has similar characteristics. Therefore, the similarity within a group is high and between different groups is low. Each individual group is called cluster. Clustering over data stream is to continuously maintain a clustering sequences observed so far, using available limited resources.

Clustering is one of the most studied problem in the data mining literature. While it is difficult to adopt traditional techniques for data stream mining due to one-pass constraints. The data stream mining constraints require incremental clustering techniques, which can observe the underlying evolution of the data and introduce new clusters as they appear as well as remove the clusters as they disappear in the data.

The center of the attention for many researchers has been the k-median and k-mean problems. The objective is to minimize the average distance from data points to their closest cluster centers. The K-median problem is discussed by Guha et al [64, 63], they proposed an algorithm that makes a single pass over the data stream and uses small space. Babcock et al [8] used exponential histogram data structure to improve Guha et al's algorithm [64]. Charikar et al [17] proposed k-median algorithm. In [42, 43] Domingos et al proposed a general method for scaling up machine learning algorithms named Very Fast Machine Learning (VFML). They have applied this method to K-means clustering (VFKM). Ordonez [112] proposed an improved incremental k-means algorithm for clustering binary data streams. Georgieva et al [59] proposed Gustafson-Kessel algorithm for real time clustering data streams. Aggarwal et al proposed *Clustream* algorithm [2] which perform clustering on any past portion of a stream where all fields are numerical. A bounce rate problem in clustering of Web usage data stream is studied by Masseglia [148].

2.4.2.2 Classification

Another most widely studied data mining tasks in data stream perspectives is *classification*. In classification, we label incoming data with one of predefined classes.

The problem of classification becomes more difficult when underlying distribution of incoming data change over time. Therefore, an effective data stream classification algorithm must take into account the data stream constraints e.g. concept drift and single pass.

Domingos et al [42] build decision trees incrementally from data streams, idea is that it is not necessary to wait for all examples to decide of a split in the tree. A statistical test is performed to decide when a split can be done. The statistical test can be done by keeping track of appropriate statistics which can be computed incrementally in bounded space.

The evolving nature of data stream classification has been addressed by several authors. Wang et al [144] have proposed a general framework for mining concept drift in data streams. The proposed technique uses weighted classifier ensembles to mine data streams. Last [93] proposed an online classification system which dynamically adjusts the size of the training window and the number of new examples between model reconstructions to the current rate of concept drift.

For the *classification* of data stream, incremental decision tree proposed in [42], a framework for mining concept drift data streams has been proposed in [144] and [93] and they discussed the online classification by adjusting the dynamic window size. The CVFDT [73] is also a well-known tree induction method on continuously changing data, based on sliding window. It constructs alternative sub-tree on each node then replaces the old tree with new data. This work has been further extended in [99, 71, 14]. The Ultra Fast Forest of Trees (UFFT)[87] is an incremental algorithm, it uses naive-Bayes classifier on decision nodes to classify test examples and to detect drift. Further in *Streaming Parallel Decision Tree* (SPDT) algorithm [13] a distributed method has been proposed to construct a decision tree.

2.4.2.3 Frequent pattern mining

Frequent pattern mining is also one of the mostly studied tasks in data mining. It is the main concept in some other data mining tasks and theories e.g. finding the frequent items or item sets is the main task in *association rule* mining and *sequential*

pattern mining.

In frequent pattern mining, the objective is to find the sets of items, which are occur frequently together in the transactions of data. It become challenging in data stream environment. Along with, single pass and limited memory constraints, the challenging task in frequent pattern mining from data stream is that infrequent patterns found in the stream may become frequent in the future. And the same, frequently occurred patterns in the past may become infrequent.

There are lot of research works in this field, some of them presents the new algorithms and others improve the existing algorithms to deal with data stream constraints more efficiently. We can find frequent patterns over whole data stream or within a specific time period using window model.

Cormode and Muthukrishnan [33] have developed an algorithm for frequent pattern mining. This algorithm uses group testing to find the hottest patterns. It maintains a small space data structure that monitors the transactions on the relation, and when required, quickly outputs all hot items, without rescanning the relation in the database.

Giannella et al [61] have developed an algorithm for computing and maintaining frequent pattern over data stream. They used a frequent pattern tree data structure and proposed an incremental algorithm to maintain frequent pattern over data stream "FP-stream".

Manku and Motwani [100] have proposed an incremental algorithm for approximate counting of frequent item sets in data streams. It uses all the previous historical data to calculate the frequent patterns. It maintains a counter of observed frequency for all frequent or potentially frequent items in a set of logical buckets. It incrementally maintains these buckets over the time while the less frequent ones are deleted periodically.

Sequential pattern mining deals with data represented as sequences (a sequence contains sorted sets of items), Masseglia et al in [134] present a survey on issues and approaches dealing with sequential pattern mining. A clustering based approach for mining approximate sequential patterns in Web usage data streams proposed in

[102]. There are some recent works on this topic [11, 12, 81, 98].

2.5 Applications

In last decades data stream mining become an active area of research, due to the importance of its applications and increasing in the generation of streaming data. Motivating examples can be found in many application domains including finance, web applications, security, networking, and sensor monitoring [9].

- *Traderbot* [135] is a web-based financial search engine that evaluates queries over real-time streaming financial data such as stock tickers and news feeds. It gives some examples of one-time and continuous queries that are commonly posed by its customers.
- Modern security applications often apply sophisticated rules over network packet streams. For example, *iPolicy Networks* [77] provides an integrated security platform providing services such as firewall support and intrusion detection over multi-gigabit network packet streams. Such a platform needs to perform complex stream processing including URL filtering based on table lookups, and correlation across multiple network traffic flows.
- Large web sites monitor web logs (clickstreams) online to enable applications such as personalization, performance monitoring, and load-balancing. Some web sites served by widely distributed web servers may need to coordinate many distributed click stream analyses, e.g., to track heavily accessed web pages as part of their real-time performance monitoring.
- There are several emerging applications in the area of sensor monitoring where a large number of sensors are distributed in the physical world and generate streams of data that need to be combined, monitored, and analyzed.

2.6 Conclusion

This chapter gave an overview of the relevant background information on data stream mining. We discussed the main properties of data stream, limited data access, unbounded and evolving nature. Further, the processing of data stream can be done over different window models or using some data stream summarizing techniques. We also discussed about the main issues in data stream mining; limited storage memory of unbounded data stream, processing time due to real time processing and concept drift. This means that usual data mining techniques can not be applied directly on data stream. Therefore, data stream learning algorithms should be capable to handle its characteristics to overcome the main limitations like continuously learning, forgetting, self-adaptation, and self-reaction etc.

In recent years, learning from data streams is a growing area of research due to its importance and limitations in its processing. It motivated us to study the problem of incremental Bayesian network structure learning from data stream. In the next chapters we will discuss what are the Bayesian networks, how we can learn it from data and later we will review the existing Bayesian network techniques applied on data stream environment.

3

Probabilistic graphical models

Contents

3.1 Introduction 33
3.2 Preliminaries 33
3.2.1 Probability and information theory 33
3.2.2 Graph theory 35
3.3 Probabilistic graphical models 36
3.3.1 Markov network 36
3.3.2 Bayesian network 37
3.3.3 Some principles of Bayesian network 38
3.3.3.1 D-separation 38
3.3.3.2 Markov equivalent class 39
3.3.4 Querying a distribution 40
3.3.4.1 Exact inference 41
3.3.4.2 Approximate inference 42
3.4 Learning 42
3.4.1 Parameter learning 43
3.4.1.1 Maximum likelihood estimation (MLE) . . . 43
3.4.1.2 Bayesian estimation 43
3.4.2 Structure learning 44
3.4.2.1 Score-and-search based methods 44
3.4.2.2 Constraint based methods 48
3.4.2.3 Hybrid learning 50
3.4.3 Evaluating structural accuracy 52

3.5 Problem of high dimensional domain 53
3.6 Conclusion 54

3.1 Introduction

In this chapter we review probabilistic graphical models (PGM) in general and Bayesian networks in detail. First, we give some standards and definitions to build basic knowledge about probability and graph theory. The purpose of this chapter is to revise some basic concepts and introduce some notations that will be used later in this manuscript.

3.2 Preliminaries

3.2.1 Probability and information theory

Definition 1. ***(Random variables):*** *A random variable is considered an outcome of a measurement process.*

Definition 2. ***(Cardinality):*** *The number of states associated with random variable known as the cardinality.*

The uppercase letters as X, Y, Z represent the random variables and lowercase letters x, y, z denote the state/value taken by the corresponding variables. Set of variables are represented by boldface uppercase letters like $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$.

Definition 3. ***(Probability):*** *The probability distribution $P(X)$ of a discrete random variable X is a list of probabilities $P(X = x)$ associated with each of its possible values $x \in \mathcal{X}$.*

Definition 4. ***(Conditional probability):*** *Given the event $Y = y$ has occurred or will occur then the probability of X is:*

$$P(X|Y = y) = \frac{P(X, Y = y)}{P(Y = y)} \tag{3.1}$$

Definition 5. ***(Independence):*** *The two random variables X and Y are independent*

if and only if:

$$P(X|Y) = P(X)$$
$$or P(Y|X) = P(Y)$$
$$or P(X,Y) = P(X).P(Y)$$

and it is denoted by $X \perp Y$. If the above equality does not exist, it means that the two random variables are dependent.

Definition 6. ***(Conditional independence):*** *The two random variables X and Y are conditionally independent given a set of random variables* **Z** *if and only if:*

$$P(X,Y|\mathbf{Z}) = P(X|\mathbf{Z}).P(Y|\mathbf{Z}) \tag{3.2}$$

and it is denoted by $X \perp Y|Z$. If the above equality does not exist then it means that the two random variables X and Y are dependent given **Z***.*

Definition 7. ***(Mutual information):*** *Between two random variables X and Y, mutual information can be defined as:*

$$MI(X,Y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x,y) log \left(\frac{P(x,y)}{P(x)P(y)} \right) \tag{3.3}$$

If $MI(X,Y) = 0$, it means that X and Y are independent.

Definition 8. ***(Conditional mutual information):*** *Between two random variables X and Y given a third variable Z or set of random variables* **Z***, conditional mutual information can be defined as:*

$$MI(X,Y|\mathbf{Z}) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} \sum_{z \in \mathbf{Z}} P(x,y,z) log \left(\frac{P(x,y)}{P(x|z)P(y|z)} \right) \tag{3.4}$$

If $MI(X,Y|\mathbf{Z}) = 0$, the random variables X, Y are independent given **Z***.*

Definition 9. ***(Bayes theorem):*** *A posteriori probability of a random variable X*

given a random variable Y can be obtained by Bayes theorem as:

$$P(X \mid Y) = \frac{P(Y|X)P(X)}{P(Y)} \tag{3.5}$$

3.2.2 Graph theory

Definition 10. ***(Graph):*** *If $V = v_1, ..., v_n$ is a non empty set, then a pair $\mathcal{G} = V, E$ is called graph (on V), where V is the set of vertices or nodes of $\mathcal{G}$ and E are the edges or links of $\mathcal{G}$ such that $E \subset \{(u, v) \parallel u, v \in V, u \neq v\}$*

Definition 11. ***(Directed and undirected graph)*** *A graph $\mathcal{G} = V, E$ is called directed graph if and only if one direction is assigned to each edge of E. In an undirected graph, no direction is assigned to each edge.*

Definition 12. ***(Path):*** *A path in a graph represents a way to get from an **origin** to a **destination** by traversing edges in the graph.*

Definition 13. ***(Directed path):*** *A directed path is a path in a directed graph, where the directions of all edges or arcs in the path are going in the same direction,*

Definition 14. ***(Directed acyclic graph):*** *A directed graph $\mathcal{G}$ is called "directed acyclic graph" (DAG) if and only if there is no path that starts and ends at the same vertex (directed cycle).*

Definition 15. ***(Partially directed acyclic graph):*** *A graph $\mathcal{G}$ is called "partially directed acyclic graph" (PDAG) if it contains both directed and undirected edges but no directed cycle.*

Definition 16. ***(Parents and children of a node)*** *If there is a directed edge from u to v then we can say that u is a parent of v and v will be the child of u.*

Definition 17. ***(Ancestor and Descendants):*** *If there is a directed path from u to v, we say u is an ancestor of v and v is a descendant of u.*

Definition 18. ***(Skeleton):*** *The skeleton of a directed graph is the undirected graph obtained by removing directions from all arcs in $\mathcal{G}$.*

3.3 Probabilistic graphical models

Probabilistic graphical models (PGM) are the conjunction of probability theory and graph theory. They provide a tool to deal with two major problems *uncertainty* and *complexity*. Graphical Models are an unified framework that allows to express complex probability distributions in a compact way. They are playing increasingly an important role in designing and analysis of machine learning algorithms.

Probabilistic graphical models are graphs, where nodes represent random variables, and edges represent dependences between variables. These models provide a compact representation of joint probability distributions.

There are two major types of PGMs first, undirected graphical models, also known as Markov Random Fields (MRFs) or Markov Networks and second, directed graphical models also called Bayesian Networks. An example of these two models with their graphical representation, independences induced from the graph and factorization obtained from these graphs are presented in the figure 3.1.

3.3.1 Markov network

Markov network or undirected graphical model is a set of random variables having a Markov property described by an undirected graph. The joint probability distribution of the model can be factorized according to the cliques of the graph $\mathcal{G}$ as:

$$P(X = x) = \frac{1}{\mathcal{Z}} \prod_{C \in cl(\mathcal{G})} \phi(C) \tag{3.6}$$

where, $\mathcal{Z}$ is a normalization factor, $cl(\mathcal{G})$ is the set of cliques of $\mathcal{G}$ and the function $\phi(C)$ is known as factor or clique maximal potential.

Undirected models are useful in the domains where interaction between variables is symmetrical and the directionality is not important.

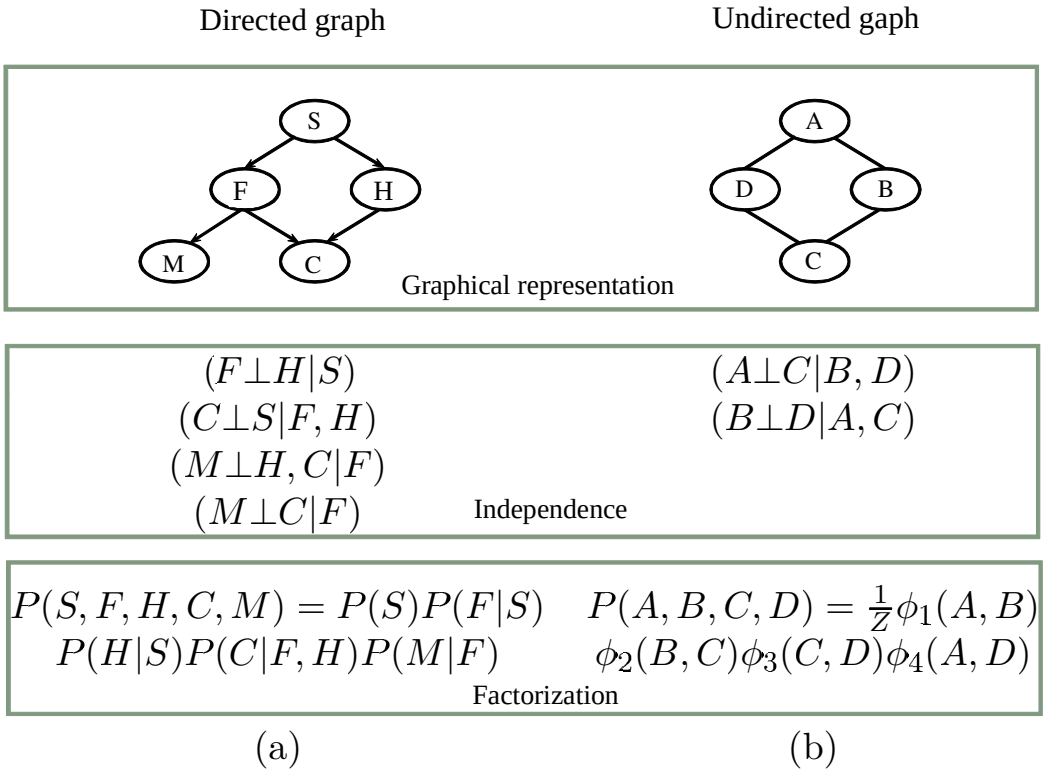


Figure 3.1: *Conditional independences and factorization of the joint distribution by probabilistic graphical models: first row - graphical representation; second row - independences induced by the structure of graph; third row - factorization driven from the structure of graph. (a) an example of DAG. (b) an example of Markov random field (MRF) [5].*

3.3.2 Bayesian network

Definition 19. ***(Bayesian network (BN)):*** *A Bayesian network* $M = \langle \mathcal{G}, \theta \rangle >$ *is composed of a directed acyclic graph (DAG)* $\mathcal{G} = (\mathbf{X}, E)$ *and a set of parameters* θ*.*

$\mathbf{X}$ *is a collection of nodes or vertices corresponding to the set of random variables* $\{X_1, X_2, ...X_n\}$ *and dependencies among these variables are expressed by the set of edges* E*. The parameters* θ *represent the probability distributions of each random variable given its set of parents:* $\theta_i = P(X_i|Pa(X_i))$*.*

A Bayesian network is a compact representation of joint probability distribution.

$$P(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i|Pa(X_i)) \tag{3.7}$$

In addition it must satisfy the Markov condition (c.f. definition 20). An example of Bayesian network model is given in Fig. 3.2.

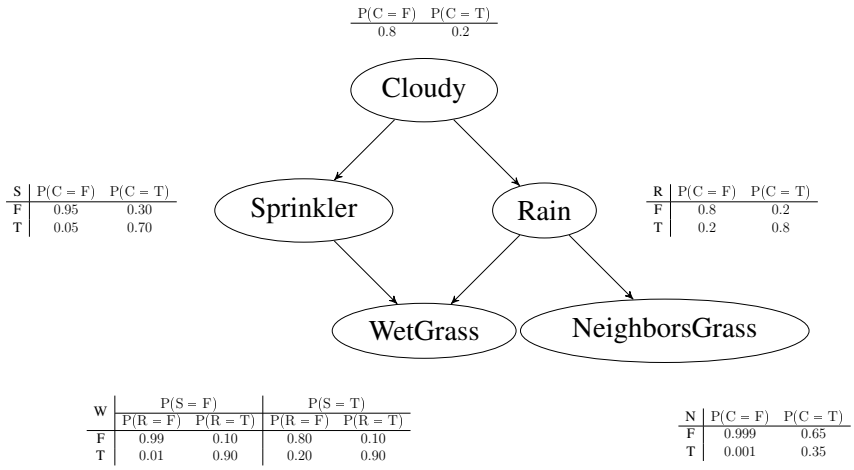


Figure 3.2: *An example of Bayesian network model; A DAG with five nodes and parameters representing the probability distribution.*

Definition 20. ***(Markov Condition):*** *M is a Bayesian network with respect to $\mathcal{G} = (\mathbf{X}, E)$ if all variables $X \in \mathbf{X}$ are independent of any subset of non-descendant variables given its parents.*

$$(X \perp NonDesendent(X) \mid Pa(X)) \tag{3.8}$$

Definition 21. ***(Faithfulness):***

A Bayesian network model M, and a probability distribution P are faithful to one another if every one and all independence relations valid in P are those entailed by the Markov assumption on M.

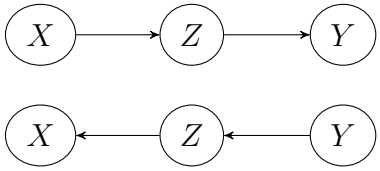
3.3.3 Some principles of Bayesian network

3.3.3.1 D-separation

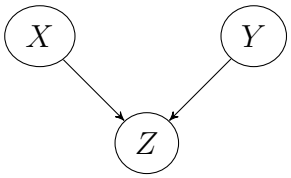
To see how probabilistic influence flow in the graph, we observe how information passes from X through Z to influence our belief about Y. Consider three nodes X, Y and Z, and there exist a path $X - Z - Y$. If influence flow from X to Y via Z then we can say that path $X - Z - Y$ is active otherwise blocked. There are three possible ways:

Serial or head-to-tail connection: If Z is not observed then path from X to Y will

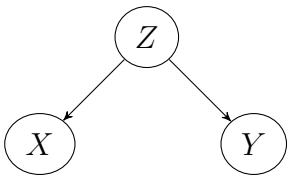
be active otherwise it will be blocked so, $X \perp Y | Z$ and $X \not\perp Y$.



Converging or head-to-head connection: If Z is not observed or any descendant of Z then path from X to Y will be active otherwise it will be blocked so, $X \not\perp Y | Z$ and $X \perp Y$. It is also called a $v - structure$.



Diverging or tail-to-tail connection: If Z is not observed then path from X to Y will be active otherwise it will be blocked so, $X \perp Y | Z$ and $X \not\perp Y$.



Definition 22. ***(D-separation)*** *or directional separation. If X, Y is a set of two random variables and* **Z** *is a set of random variables, X and Y are d-separated by* **Z** *if and only if* **Z** *block all paths from X to Y.*

Theorem 1. ***(Pearl, 1988** [115]) In a faithful Bayesian network to a distribution P, any d-separation condition in the network corresponds to actual independence condition in P.*

3.3.3.2 Markov equivalent class

Two DAGs $\mathcal{G}_1$ & $\mathcal{G}_2$ are called Markov equivalent if they share the same conditional independencies. In other words, all the DAGs which share the same d-separation are known as Markov equivalent. According to Verma and Pearl's theorem:

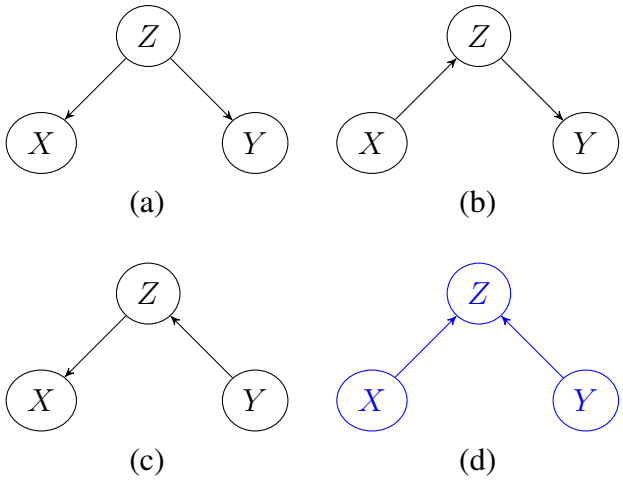


Figure 3.3: *An example of Markov equivalent class for three variables, X, Y, Z and four DAGs . First three DAGs (a), (b) and (c) have the same independence structure and (d) corresponds to another set of independencies.*

Theorem 2. *(**Verma and Pearl:** [142]) Two DAGs are equivalent if and only if they have the same skeleton and v-structures (head-to-head connection).*

For example in figure 3.3, there are four different DAGs with same number of variables, and having the same skeletons. According to the above theorem DAGs (a), (b) and (c) belong to the same Markov equivalent class. But DAG (d) has a *v-structure* $X \rightarrow Z \leftarrow Y$ and it's the only graph in its own equivalence class.

Definition 23. ***(Non-reversible and reversible edges):** A directed edge $X \rightarrow Y$ is non-reversible in graph $\mathcal{G}$ if for every DAG $\mathcal{G}'$ equivalent to $\mathcal{G}$, $X \rightarrow Y$ exists in $\mathcal{G}\prime$ otherwise it will be reversible.*

The completed PDAG (CPDAG) corresponding to an equivalence class is the PDAG consisting of a directed edge for every non-reversible edge in the equivalence class, and an undirected edge for every reversible edge in the equivalence class [24].

We can convert a DAG into its Markov equivalent class CPDAG by using a method proposed by Chickering in [24].

3.3.4 Querying a distribution

The Bayesian network models represent full joint probability distribution as $P(\mathbf{X})$. It can be used to answer probabilistic queries about a subset of unobserved

variables when other variables are observed. A common query type is a *conditional probability query*. In this query type we ask a query to the joint distribution and the objective is to compute [82]:

$$P(\mathbf{X} \mid \mathbf{E} = e) = \frac{P(\mathbf{X}, e)}{P(e)} \tag{3.9}$$

The equation 3.9 has two parts,

– query variables, $\mathbf{X}$ is a subset of random variables in the network.

– evidence, $\mathbf{E}$, a subset of random variables in the model and an instantiation e

Another type of query is *maximum a posteriori probability* (MAP) query. It compute the most likely assignment to all of the (non-evidence) variables [82]. If $\mathbf{X}$ is a set of query variables and evidence $\mathbf{E} = e$ then

$$MAP(\mathbf{X} \mid e) = \arg\max_{\mathbf{X}} P(\mathbf{X}, e) \tag{3.10}$$

This process of computing the posterior distribution of variables given evidence is called probabilistic inference [40] and it is proved as NP-hard [31]. There are different methods proposed in the literature for probabilistic inference. We can divide these algorithms in exact and approximate inference techniques. A detailed survey about inference algorithms by Guo and Hsu [65] could be interesting for more information.

3.3.4.1 Exact inference

Pearl introduced a message propagation inference algorithm for tree-structured Bayesian networks in [114]. It is an exact inference algorithm and has polynomial complexity in the number of nodes. Another popular exact inference algorithm is *junction tree* or *clique tree* [94]. It is also known as *clustering* algorithm. It is a generalization of message propagation algorithm. Its complexity is exponential in the size of largest clique of junction tree. *Variable elimination* [149] is also an exact Bayesian network inference algorithm. It eliminates other variables one by one by summing out them. Its complexity can be measured by the number of numerical

multiplications and numerical summations it performs.

As inference in Bayesian networks was found to be NP-hard problem in general [31]. Therefore, exact inference methods could be an effective solution only for networks having small cliques. The complexity of these algorithms depends upon the structure of the graph, if the structure is complex i.e. large cliques, then it will take long time to compute the inference.

3.3.4.2 Approximate inference

For complex structures, approximate inference algorithms are widely used than exact techniques. Most of the approximate inference techniques are based upon the Monte Carlo methods. They generate a set of randomly selected samples according to the conditional probability tables int the model, and approximate probabilities of query variables by the frequencies of appearance in the sample. The accuracy depends on the size of samples regardless of the structure of the network [65]. The complexity of generating a sample is linear in the size of the network [141]. While, the issue with these methods is related to the quality of answers they compute. First technique that uses Monte Carlo methods is *logic sampling* developed by Henrion [69]. Some of other methods are including *likelihood weighting* [50, 127], *self-importance sampling* [127], *heuristic importance* [127], *adaptive importance sampling* [20] etc.

3.4 Learning

Practically, it is a difficult (or impossible) task to manually specify the structure and the parameters of a BN by domain expert. Therefore, learning BN from data is an important problem. There are two major learning tasks: estimating the parameters of the model and learning the structure of the network. First we introduce the parameter learning problem and then structure learning [1]. For more informations, we

1. Generally, most of the algorithms first learn the structure of the model and then estimate its parameter. However, normally books discuss parameter learning problem first, because scoring based structure learning algorithms estimate parameters as part of their process.

will recommend our readers to consult a detailed literature review about Bayesian network learning by Daly et al [35]. We will consider here only fully observed (there are no missing values) data. Reader can find more information about learning with incomplete data in [68, 97].

3.4.1 Parameter learning

Parameter learning is a major part of Bayesian network learning process. In parameter estimation, we have to determine the parameters θ from data, given the structure of the network. The parameters define the conditional probability distributions for the given structure. There are two main approaches: *maximum likelihood estimation* and *Bayesian estimation.*

3.4.1.1 Maximum likelihood estimation (MLE)

The maximum likelihood is the principle of estimating the probability distribution of the parameter thats best fit the data. It estimates the probability of the parameter by using its frequency in the observational data:

$$\theta^{\hat{MLE}} = \hat{P}(X_i = x_k | Pa(X_i) = x_j) = \frac{N_{ijk}}{\sum_k N_{ijk}} \tag{3.11}$$

3.4.1.2 Bayesian estimation

An alternative to the maximum likelihood is *Bayesian estimation*. For instance, the *Maximum a posteriori* (MAP) estimation finds the most probable parameters given the data.

$$\theta^{MAP}_{i,j,k} = P(X_i = x_k | Pa(X_i) = x_j) = \frac{N_{i,j,k} + \alpha_{i,j,k} - 1}{\sum_k N_{i,j,k} + \alpha_{i,j,k} - 1} \tag{3.12}$$

Where $\alpha_{i,j,k}$ is the Dirichlet distribution parameter associated with the prior probability $P(X_i = x_k | Pa(X_i) = x_j)$.

n	Number of possible DAGs
1	1
2	3
3	25
4	543
5	29 281
6	3 781 503
7	1 138 779 265
8	8 783 702 329 343
9	1 213 442 454 842 881
10	4 175 098 976 430 598 143

Table 3.1: *Number of possible DAG patterns generated by n variables [122].*

3.4.2 Structure learning

BN structure learning aims to select a probabilistic dependency structure that explains a given set of data. In many applications it is not provided by the experts and needs to be learned from data. Bayesian network structure learning has been proven as NP-hard problem by Chickering [22] and is known as a more complex phase in Bayesian network learning. Consequently, number of heuristic methods proposed in literature to reduce the search space of the BN structure. The number of possible structures g is super-exponential in the terms of number of variables n [119]:

$$g(n) = \begin{cases} 1 & \text{if n = 0 or 1} \\ \sum_{i=}^{n} (-1)^{i+1} \binom{n}{i} 2^{i(n-1)} g(n-i) & \text{if n > 1} \end{cases} \tag{3.13}$$

The number of possible DAGs are increasing exponentially (cf. figure 3.1), for this reason heuristic based methods have been proposed. We can divide these methods in three categories: score based, constraint based and hybrid methods.

3.4.2.1 Score-and-search based methods

One of the mostly studied ways of Bayesian network structure learning are *score-and-search* techniques. It addresses the problem as a model selection, therefore it

searches through possible DAG space for the DAG which maximizes a certain score function. To reduce the search space, different heuristics have been proposed in the literature like greedy-search or searching by MCMC. Most famous algorithms are : K2 [32], Maximum Weight Spanning Tree (MWST) [29], Greedy Equivalence Search (GES) [25] and Greedy search (GS) [25].

These methods can be compared with respect to:

– the way they define a search space.
– a criteria to explore this search space.
– a scoring function to assign a score value to the DAG structure to see how good a match is made with data.

Search space

The search space can be a space over DAGs or space over CPDAGs, where equivalent DAGs (cf. section 3.3.3.2) are summarized by their representatives.

Heuristics

In general, the search space is very large, if the search space is reduced then it is easy to explore the complete space and find a structure with high score. Otherwise there are different heuristics to optimize this problem like optimal tree in the first family (reduced search space and optimal solution), greedy search, TABU list, random restart, simulated annealing, genetic and evolutionary algorithms, searching with different ordering of the variables etc.

Scoring function

A score function is the evaluation criteria for the DAGs and it assigns a value to a given DAG based on the observed data. Hence, it tells us how well the structure fits the data. Assuming a DAG structure $\mathcal{G}$ and data $P(\mathcal{G}|D)$ is the marginal likelihood, i.e. the theoretical score [27]:

$$Score(\mathcal{G}, D) = P(\mathcal{G}|D) \tag{3.14}$$

and the following scores are some different approximations: *Bayesian Information Criteria (BIC)* [126], *Akaike's Information Criteria (AIC)* [4] , *Bayesian Dirichlet equivalent (BDe)* [32] and *Minimum Description Length (MDL)* [118].

$$Score_{BIC}(\mathcal{G}, D) = \log L(D|\theta^{ML}, \mathcal{G}) - \frac{1}{2} Dim(\mathcal{G}) \log N \tag{3.15}$$

$$Score_{MDL}(\mathcal{G}, D) = \log L(D|\theta^{ML}, \mathcal{G}) - \mid A_B \mid \log N - c.Dim(\mathcal{G}) \tag{3.16}$$

$\log L$ represents log likelihood and $\theta^{ML} = argmax\, P(D \mid \theta)$ where ML denotes the maximum likelihood. $Dim(\mathcal{G})$ is a dimension of graph that is number of parameters to write $\mathcal{G}$. N is the number of observations. $\mid A_B \mid$ is the number of edges in graph $\mathcal{G}$ and c is the number of bits used to store each parameter.

The scores should be locally decomposable and assign the same score value to the Markov equivalent classes (cf. section 3.3.3.2).

Definition 24. ***(Decomposable score)*** *Given a DAG $\mathcal{G}$ and a scoring function $Score(\mathcal{G},\ D)$, $Score$ is said to be decomposable if we can find a function f such that*

$$Score(\mathcal{G}, D) = constant + \sum_{i=0}^{n} f(X_i,\ Pa(X_i)) \tag{3.17}$$

Definition 25. ***(Score equivalence)*** *Given two Markov equivalent DAGs $\mathcal{G}$ and $\mathcal{G}'$ a scoring function f, f is said to be score equivalent if and only if*

$$f(\mathcal{G}, D) = f(\mathcal{G}', D) \tag{3.18}$$

Algorithms

Some of well known algorithms using above criteria are:

- The MWST algorithm [29] is an example of this category. It finds the maximal spanning tree, where each edge is weighted according to the mutual information between the two variables.
- Algorithm K2 [32] uses greedy search with defining an order on the variables, such that if the order of X is before Y then Y will not be considered as a

parent of X.

– Algorithm greedy search (GS) [23] uses certain operators to explore the search space e.g. add_edge, $remove_edge$ and $reverse_edge$. Before applying add_edge or $reverse_edge$ operators it will ensure that these operators do not create a directed cycle in the new graph (acyclic). Next section gives more detail about greedy search.

Greedy search:

The simplest algorithm based on *score-and-search* technique is a *Greedy search* (GS) [25]. It is also known as *Hill-climbing search* (HC). As shown in Algorithm 1, it traverses the search space by examining only possible local changes in the neighborhood of the current solution and applying the one that is maximizing the score function.

Definition 26. ***(Neighborhood)****[122]*

The neighborhood of a model M is the set of all alternative models that can be generated by applying a single operator; add_edge or delete_edge or reverse_edge, and an argument pair A such that these operators do not create a directed cycle in the new graph.

Hill-climbing search starts from an initial model M_0. It iteratively constructs a sequence of models $\{M_0, M_1, M_2, ..., M_n\}$ by examining their neighborhoods and choosing the model having best quality function score (cf. figure 3.4). It stops when there is no more improvement in existing model score and the current model has a highest score in its neighborhood. The path followed by this search procedure is known as its *search path*.

Definition 27. ***(Search Path)****[122]*

Let M_0 be an initial model and M_f is a final model obtained by a hill-climbing search algorithm as:

$$M_f = op_n(...(op_2, (op_1,\ A_1),\ A_2), ...,\ A_n) \tag{3.19}$$

Algorithm 1 *Greedy Search (GS)*

Require: *data* D*; scoring function* $Score(M,\ D)$*; a set of operators* $OP = \{op^1, \ldots, op^k\}$

Output: *DAG*: a model M of high quality

i = 0

M_i = $\varnothing$

Repeat

$oldScore = Score(M_i,\ D)$

i++

$M_i = op(M_{i-1},\ A_i)$

** where $op(M_{i-1},\ A_i)$

$= arg\,max_{(op^k,\ A) \in \mathcal{G}_n} Score\left(op^k(M_{i-1},\ A_i),\ D\right)$**\

Until $oldScore \geqslant Score(M_i,\ D)$

Return M_i

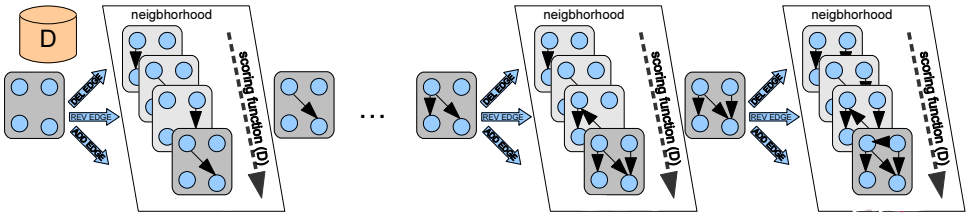


Figure 3.4: *Outline of greedy/hill-climbing search*

where each operator "op" and argument pair "A" yields a model with highest score to the neighborhood. So search path is the sequence of operators and argument pairs $\mathcal{O}_{op} = \{(op_1,\ A_1),\ (op_2,\ A_2),\ \ldots\ (op_n,\ A_n)\}$ *used to build* M_f*.*

The models in the search path are in an increasing order of their score values and the final model is a local maxima of the domain models.

$$Score(M_0,\ D) < Score(M_1,\ D) < Score(M_2,\ D) \cdots < Score(M_f,\ D) \quad (3.20)$$

3.4.2.2 Constraint based methods

Constraint based methods are considering a Bayesian network as a representation of independencies. They use statistical tests to obtain the dependency and conditional independency in the data. Some well known algorithms in this category are SGS

algorithm [94], CI algorithm [80], and PC algorithm [132].

These algorithms are based on the same principle [106]:

- Construct a skeleton of BN, having the relations between variables using conditional independence tests.
- For edge orientation, first find the v-structure (non-reversible edges) then infer some arrow orientations shared by all DAGs Markov equivalents. Further it propagates orientations of remaining arcs.

Statistical tests: The constraint based algorithms used statistical tests e.g. χ^2, G^2 and Mutual Information (MI), for conditional independence and measure of the strength of association between two variables. G^2 statistic is asymptotically distributed as χ^2 with appropriate degrees of freedom [138] as well as Mutual Information (MI) too, according to the Kullback theorem [86].

Theorem 3. ***(Kullback)*** *Given a data set D with N elements, if the hypothesis that X and Y are conditionally independent given $\mathbf{Z}$ is true, then the statistics $2N\ MI_D(X, Y|\mathbf{Z})$ approximates to a distribution $\chi^2(l)$ (Chi-square) with $l = (r_X - 1)(r_Y - 1)r_{\mathbf{Z}}$ degrees of freedom, where r_X, r_Y and $r_{\mathbf{Z}}$ represent the number of configurations for the sets of variables X,Y and $\mathbf{Z}$, representatively. If $\mathbf{Z} = \varnothing$, the statistics $2N\ MI_D(X, Y)$ approximates to a distribution $\chi^2(l)$ with $l = (r_X - 1)(r_Y - 1)$ [86]*

Constraint based methods are more efficient than score based methods as they have a deterministic search procedure and well defined stopping criteria, on the other hand, these methods are criticized due to problem of insufficient data and number of conditioning variables in statistical test.

The acceptance or rejection of null hypothesis is implicitly depends upon the degree of freedom. The degree of freedom exponentially increases as the number of variables increase in the conditioning set. Therefore, independence is usually assumed when there is not enough data to perform the test reliably [38].

Algorithm 2 $MMHC(D)$

Require: *Data* D

Output: *a DAG representing Bayesian network structure* ($\mathcal{G}$)

** ***Local identification*** **\

For $j = 1$ *To* n **do**

$CPC(X_j)$ = MMPC(X_j, D)

Endfor

** ***Greedy search*** **\

starting model M = empty graph.
Only try operators *add_edge* $(Y \rightarrow X)$ if $Y \in CPC(X)$
(no constraint for *remove_edge* or *reverse_edge operators*)

Return $\mathcal{G}$ the highest scoring DAG found

3.4.2.3 Hybrid learning

Both score and constraint based methods have their own advantages e.g. *score-and-search* based algorithms can deal efficiently with small datasets, whereas the ability to scale up to hundreds of thousands of variables is a key advantage of *constraint* based algorithms [38]. These two kinds of approaches can be combined together for Bayesian network structure learning as: use the learned network from constraint-based methods as the starting point for the search-and-score-based methods. This approach is known as hybrid. For example let us cite, Singh and Valtorta [130], where they proposed to generate the ordering of the variables with *constraints-based* approach and learn the Bayesian network structure with the *score-and-search* based approach using the ordering of variables. Sparse Candidate algorithm (SC) [49] and Max-Min Hill Climbing algorithm (MMHC) [138], some other examples are [133] and [130]. The MMHC algorithm has been proposed to solve high dimensionality problem and it outperforms on wider range of network structure.

Max-Min hill climbing algorithm: MMHC algorithm (cf. Algorithm 2) combines both *constraint based* and *score-and-search based* approaches. In first step, it learns the possible skeleton of the network using a constraint based local algorithm Max-Min Parent Children (MMPC) [136]. The efficiency of MMPC is obtained by

Algorithm 3 $MMPC(T,\ D)$

Require: *Data* D*, target variable* T
Output: *a set of Candidate Parent and Children (CPC) of T*

$CPC = \overline{MMPC}(T, D)$
For each $X \in CPC$ **do**
  if $T \notin \overline{MMPC}(X, D)$ **then**
    $CPC = CPC \backslash X$
  Endif
Endfor
Return CPC

restricting the parent set. If two variables are found independent then it assumes that they will not be connected in final network. MMPC returns a very concise search space. Later on, in its second step MMHC algorithm applies a greedy hill-climbing search to orient the determined edges (as well as removes false positive edges).

The MMPC(T) algorithm (cf. algorithm 3) discovers the set of CPC (candidate parent-children, without distinguishing among both) for a target variable *T*. It is a combination of $\overline{MMPC}(T)$ algorithm and additional correction for symmetric test, where it removes variable X from CPC of a target variable T, if T is not in the set of CPC of X.

Algorithm $\overline{MMPC}(T)$ (cf. Algorithm 4) also has two phases. The first phase, *forward phase*, adds variables in CPC for a target variable *T*, using Max-Min heuristic (cf. figure 3.5).

$$MaxMin\,Heuristic = \max_{X \in \mathbf{X} \setminus CPC} \left(\min_{S \subseteq CPC} Assoc\,(X;\ T \mid S) \right) \tag{3.21}$$

Where function $Assoc(X;\ T \mid S)$ is an estimate of the strength of association (dependency) of X and T given S. The second, *backward phase*, removes the false positives variables from CPC.

Algorithm 4 $\overline{MMPC}(T)$

Require: *target variable T; data D*

Output: *a set of Candidate Parent and Children (CPC) of T*

**** *Forward Phase: MaxMinHeuristic* ****

$CPC = \varnothing$

Repeat

$< F, assocF > = \max_{X \in \mathbf{X} \setminus CPC} (\min_{S \subseteq CPC} Assoc(X; T \mid S))$

if $assocF \neq 0$ **then**

Add (CPC, F)

Endif

Until CPC has not changed

**** *Backward Phase:* ****

For all X ∈CPC

if $\exists S \subseteq CPC$, s.t. $Assoc(X; T \mid S) = 0$ **then**

$CPC = CPC \setminus X$

Endif

Return CPC

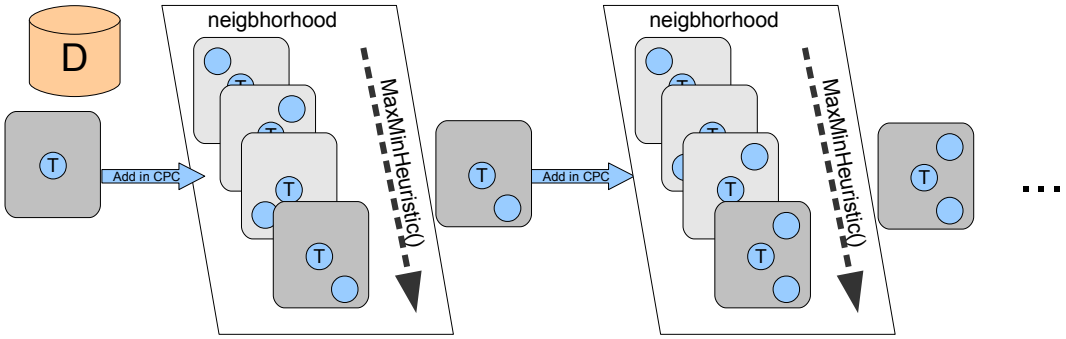


Figure 3.5: *Outline of* $\overline{MMPC}(T)$ *forward phase*

3.4.3 Evaluating structural accuracy

How can we measure the accuracy of a structure learning algorithm? One usual solution is choosing an existing network, gold standard (or randomly) and generate the dataset from its joint probability distribution. Further, we apply our algorithm upon generated dataset and learn the structure of the network. Later, we can compare the learned BN with the initial one.

In the literature, there exist different techniques based upon the one of above

methods e.g. Score based, Kullback-Leibler divergence based, Structural Hamming Distance (SHD) based along with sensitivity and specificity based methods [108].

Each method has pros and cons, some of them are complex to compute but they take into account Markov equivalent classes e.g. score and KL-divergence based methods. And others are simple but do not consider Markov equivalent class e.g. sensitivity and specificity based methods.

The SHD method is simple to compute. In a short, the SHD is a number of changes to be applied upon obtained network to convert it into the initial one. The original one does not consider the Markov equivalence, while the one proposed by Tsamardinos [138] takes it into account by comparing corresponding CPDAGs, therefore it does not penalize an algorithm for structural differences that cannot be statistically distinguished. If the distance is lower then the corresponding algorithm will be considered as better.

3.5 Problem of high dimensional domain

Learning small scale Bayesian networks is already well explored. But learning in high dimensional domains e.g. social networks or biological (bioinformatics, biomedical etc) domains are facing problem of high dimensionality. These domains produce datasets with several hundreds or hundreds of thousands of variables. This high dimensionality is a serious challenge in a Bayesian network field.

We can see the *score-and-search* based methods have problem of large search space, whereas the constraint based methods have limitation of conditional variables and the dataset size. Recent works [49, 70, 74, 137, 138] using hybrid approach are the only one to deal with high dimensionality problem. The problem is more challenging for learning algorithms when incoming data is generated by data stream environment.

3.6 Conclusion

This chapter gave some overview of probabilistic graphical models and discussed some relevant basic concepts about graph and probability theories because these are the building blocks for PGMs. We discussed two main subclasses, Markov network in general and Bayesian networks in detail.

In Bayesian network learning, there are two parts, first its parameter learning and second network structure learning. Parameters can be learned by maximum likelihood or maximum a posteriori methods. On the other hand, Bayesian network structure learning is a NP-hard (Non-deterministic Polynomial-time hard) problem. The network search space is super exponentially large.

There are three main categories for BN structure learning algorithms score based, constraint based and hybrid. We see the only hybrid methods could scale out for high dimensional domains. This motivates us to study the incremental Bayesian network structure learning using hybrid method. In next chapter we will present how Bayesian network structure can be learn from data stream.

4

Bayesian networks and data streams

Contents

4.1 Introduction 56
4.2 Learning from stationary domains 57
4.2.1 Score-and-search based methods 57
4.2.1.1 Buntine's approach 57
4.2.1.2 Lam and Bacchus's approach 60
4.2.1.3 Friedman and Goldszmidt's approach 64
4.2.1.4 Roure's approach 69
4.2.2 Hybrid method 75
4.3 Learning from non-stationary domains 78
4.4 Conclusions 83

4.1 Introduction

Since nineties, the Bayesian network learning has been receiving great attention from the researchers. One area of this research that has seen much activity is structure learning of the network. In Bayesian network structure learning process, the probabilistic relationship for variables are discovered or inferred from the observations of these variables (database). Most of work consider the database as a set of static observations.

Incremental Bayesian network structure learning is another part of this research area that has received very little attention. In this area the database is considered as an unbounded set of observations received sequentially as a stream of data and it builds a network structure step by step along the way. While, there are some articles in the literature that address incremental parameter learning [131, 111, 39].

In this chapter we present the previous work in this area of research, *incremental Bayesian network structure* learning, to the extent that the next part of this thesis can be understood. There are very limited works in the literature that addressed this issue. The detailed review of some of them can be found in Roure's work [124]. These approaches proposed to iteratively revise the structure of a Bayesian network. It can be classified in two categories, approaches dealing with the *stationary* domains and *non-stationary* domains. Furthermore we can divide these domains with respect to Bayesian network structure learning techniques such as *constraint* based, *score-and-search* based and *hybrid* etc.

This chapter is organized as follows: Section 4.2 describes the incremental Bayesian network structure learning approaches dealing with *stationary domains* and further it is divided in two subsections 4.2.1 and 4.2.2, first one gives an overview of the methods that use *score-and-search* based Bayesian network structure learning techniques and in later subsection 4.2.2 methods that use the hybrid approach of Bayesian network structure learning. Section 4.3 presents the approach that deals with non-stationary domains. Finally, the concluding statements is in section 4.4.

4.2 Learning from stationary domains

Algorithms that deal with *stationary domains* consider, the data is drawn from single underlying distribution. Therefore, it will not change with the passage of time. The models are also not very different when they are evaluated with respect to similar data sets [121]. If the new data changed the underlying distribution, then in this situation these incremental algorithms would not be able to find a model with high quality. We can divide these algorithms in further two categories with respect to their Bayesian network structure learning techniques as *scoere-and-search* based and *hybrid*.

4.2.1 Score-and-search based methods

4.2.1.1 Buntine's approach

In [15], Buntine proposed an incremental algorithm for refinement of Bayesian network on the arrival of new data over sliding window. It requires total ordering of all variables and dataset. It stores different alternative networks having score value within factor E, where E is an user defined parameter of the algorithm. Factor E is also used to specify the number of alternative BNs could be produced to keep the memory in the control. Buntine's approach is a generalization of $K2$ algorithm [32]. It is a batch algorithm that uses the *score-and-search* based Bayesian approach, later he proposed some guidelines for converting it into an incremental or online algorithm. He considered two conditions, if there is not enough time to update parent structure, the algorithm will only update posterior probabilities of the parent lattices. Otherwise both structure and parameters will be updated.

Before presenting Buntine's incremental approach, first we see how his batch algorithm works.

Buntine's batch approach

As we mentioned above, Buntine's batch approach is a generalization of $K2$ algorithm. It uses the same greedy search approach as proposed by Cooper and

Herskovits [32], and the partial theory given by the domain experts as a prior knowledge. This partial theory is a total ordering of the variables "$\prec$", defines parents and non-parents of each variable. It also includes the list of alternative parents. For a variable x the alternative parent sets $\prod_x$ will be the collection of subsets of $\{Y : Y \prec x\}$. The combination of these, gives a space of alternative networks. It is represented by Cartesian product of all parent sets of each variable $\otimes_{i=1}^{n} \prod_i$. The strength of potential parents measured by factor E. Here we also store the network parameters θ_{ijk} for each possible parent set. Buntine refers this space of alternative networks and parameters as a *combined Bayesian networks.*

Let P_X is a set containing sets of alternative parent variables for a variable X. Then, in order to reconstruct the network and to access alternative parent sets $\prod_X \in P_X$, they are stored in a lattice structure where each subset and superset parent structures are linked together in a web, denoted the *parent lattice* for X. The parent lattice P_X for the variable X can be defined as, the root node is an empty set and the leaves are parents $\prod_X$ having no superset in P_X.

The algorithm starts from an empty lattice and uses a simple beam search with three parameters C, D and E, such that $1 > C > D > E$. These parameters used as a threshold to classify parent sets as *alive*, *asleep* or *dead*. If threshold adjusts to 1, the algorithm behaves as a simple greedy search, otherwise it will be a beam search.

Buntine use three classifications of parent sets, *alive*, *asleep* and *dead*.

- *Alive* is the list of parent sets having posteriors within factor D and they are considered alive in the parent lattice.
- *Dead* parent sets are the sets having posteriors less than factor E. They are considered as unreasonable in further search procedure.
- *Asleap* is the list of parent sets considered as unreasonable now and may become alive later on.

Open-list is the list of parent sets to be further expanded.

Buntine's incremental approach

He considers two situations to adapt his batch algorithm for incremental purpose. First one is a rapid update of combined Bayesian network, if there is not enough time to update parent structure then the algorithm will update only posterior probabilities of the parent lattice. In the second situation, both structure and posteriors are only updated according to the new data.

First situation:

In this case, on the arrival of new examples we need to rapidly update parameters of the combined Bayesian network without changing the parent lattice. Therefore, for each variable X_i and for each alternative parent set $\prod_{X_i}$, it updates the posterior probabilities and counters N_{ijk}. It only updates the counters for those parent sets which are *alive* in the parent lattice.

Second situation:

On the available of new examples and additional time, the algorithm initiates incremental update process of the combined Bayesian network. In this process, it updates the posterior probabilities of all alive parent sets of the lattice. Then it updates the alive and asleep parents in order to ensure that they are representing best posterior of the parent sets. Furthermore, it expands the nodes from open list and continue with the search. During this update process, some nodes may oscillate on and off the alive list and open list because the posteriors ordering of the parent sets oscillate as the new example are added and posteriors are also modified.

Remarks:

Buntine's approach is a generalization of $K2$ algorithm, when parameters C, D and E adjust to 1 then it behave as $K2$. This approach considers total ordering of the variables, which is difficult to know in advance in real world applications. It handles incoming data stream over sliding window and maintains sufficient statistics

in a lattice structure of candidate parent set for each node. Therefore, it uses MAP approach (cf. Section 4.2.1.3) to retrieve these statistics. It produces a combined Bayesian network rather than a single network. He does not provide any experimental evaluation to validate the effectiveness of his approach. His approach used *score-and-search* based Bayesian network structure learning technique, which has its own limitations. It is not scalable for high dimensional domains.

4.2.1.2 Lam and Bacchus's approach

Lam and Bacchus [92] propose a different approach based on Minimal Description Length (MDL) principle. It is an extension of their batch algorithm proposed in [91]. The objective is to refine an existing Bayesian network structure on the arrival of new data in novel sliding window, which might refer only a subset of the variables. In the refinement process they implicitly assumed that the existing network is already a fairly accurate model of the previous data and new structure is very similar to the current one.

The MDL principle is based upon the idea that, the best representative model of the data is that model which minimizes the sum of the length of the encoding of *a*) the model and *b*) the data given the model [118]. The sum of these two terms is known as total *description length*.

In each iteration the description length of whole structure is improved by minimizing the description length of the subgraph, whose topology is changed by the new data. The algorithm, first learn both partial network structure from the new data, and existent network using the MDL learning method and then modifies locally the global old structure using the newly discovered partial structure.

Learning partial structure:

The sources of data for refinement process are new data and existent network structure $\mathcal{G}_n$. First objective is to learn a partial network $\mathcal{G}_p$ from these two sources of data. Therefore, we can find $\mathcal{G}_p$ using the MDL principle, minimizing the sum of the description length of following items:

1. The partial network structure $\mathcal{G}_p$
2. The new data given the partial network structure $\mathcal{G}_p$
3. The existent network $\mathcal{G}_n$ given the partial network structure $\mathcal{G}_p$

The sum of the last two items corresponds to the description length of the source data given the model. We assume that these two items are independent of each other given $\mathcal{G}_p$, therefore they can be evaluated separately. To compute the description length of first two items, the writers used their encoding scheme presented in [91].

To find the description length of third item, we have to compute the description length of existent network (of all n variables) $\mathcal{G}_n$ given $\mathcal{G}_p$. We already have a description length of $\mathcal{G}_p$ and we need to describe the differences between $\mathcal{G}_n$ and $\mathcal{G}_p$. Therefore, to compute the description length of the difference, the differences between $\mathcal{G}_n$ and $\mathcal{G}_p$ can be described as:

Reversed arcs: A list of arcs that are in $\mathcal{G}_p$ and also in $\mathcal{G}_n$ but with opposite directions.

Additional arcs: A list of arcs that are in $\mathcal{G}_p$ and that are not in $\mathcal{G}_n$

Missing arcs: A list of arcs that are in $\mathcal{G}_n$ but are missing in $\mathcal{G}_p$

The description length of third item will be the encoding length of the collection of above arc differences. Let r, a and m are the reversed, additional and missing arcs respectively, in $\mathcal{G}_n$ with respect to $\mathcal{G}_p$. And to encode an arc requires its source and destination node, also a node can be identified by $\log n$ from $\mathcal{G}_n$ structure. Thus we need $2 \log n$ bits to describe an arc. Then, the description length of $\mathcal{G}_n$ given $\mathcal{G}_p$ can be computed as:

$$(r + a + m)2 \log n \tag{4.1}$$

Now, we compute the description length for each node using equation 4.1. Each arc can uniquely be assigned to its destination node. Let r_i, a_i and m_i be the number of reversed, additional and missing arcs of a node X_i in $\mathcal{G}_n$ given $\mathcal{G}_p$. If $\mathcal{G}_n$ network contains a set of nodes $\mathbf{X}$ then equation 4.1 can be rewritten as:

$$\sum_{X_i \in \mathbf{X}} (r_i + a_i + m_i) 2 \log n \tag{4.2}$$

If $\mathcal{G}_p$ network structure contains the nodes X_p and $X_p \in \mathbf{X}$ and $X_q = \mathbf{X} \backslash X_p$ then the sum in previous equation can be written as

$$DL_3 = \sum_{X_i \in \mathbf{X}_p} (r_i + a_i + m_i) 2 \log n + \sum_{X_i \in \mathbf{X}_q} (r_i + a_i + m_i) 2 \log n \tag{4.3}$$

As we want to compare different partial structures $\mathcal{G}_p$ therefore second part of the above equation is constant over them and we need to compute only the first part of the equation.

The total description length of the network is then computed by simply adding node measures over all the nodes. To obtain the partial network structure, we have to apply a search procedure to find the structure with lowest total description length. For this purpose a batch algorithm [91] is used.

Refining the global structure:

Now, we can refine the original network $\mathcal{G}_n$ using the previously obtained partial network structure $\mathcal{G}_p$. The objective of this process is to find the structure with lowest description length with the help of $\mathcal{G}_p$ and $\mathcal{G}_n$.

Let $\mathbf{X}$ be a set of n nodes in original network $\mathcal{G}_n$, X_p be the set of nodes in partial network $\mathcal{G}_p$ such that $X_p \subset X$ and DL_{X_p} is its description length. Suppose we find new ways to connect the nodes in $\mathcal{G}_p$ such that it does not create any cycle when integrated in original structure. It will change the description length of the nodes. Let $DL_{x_p}^{new}$ be the sum of description length of all nodes in new topology after replacing the nodes and new resulting network is $\mathcal{G}_n^{new}$ then

Theorem 4. *if $DL_{x_p}^{new} < DL_{x_p}$ then $\mathcal{G}_n^{new}$ will have a lower total description length than $\mathcal{G}_n$ [92].*

Therefore according to the above theorem, the new organization will improve

the network.

Let say a node X_i in $\mathcal{G}_p$ has a set of parents $Pa_i(\mathcal{G}_p)$ (in $\mathcal{G}_p$) and its description length is DL_i. However in general, node X_i will have a different set of parents in $\mathcal{G}_n$ and different description length. If $Pa_i(\mathcal{G}_n) \not\subset X_p$, then these two description lengths are not comparable. In this situation, the node X_i has a parent in $\mathcal{G}_n$ which does not appear in the new data. Hence, the new data is unable to tell us about the effect of that parent on $\mathcal{G}_n$'s description length. As well as we find all the nodes whose parents are in $\mathcal{G}_p$ and also in $\mathcal{G}_n$, in this algorithm these nodes are called set of *marked* nodes.

Suppose for a certain *marked* node X_i, we decide to replace its parents in original network $\mathcal{G}_n$ with its parents found in $\mathcal{G}_p$ and new structure $\mathcal{G}_{n_1}$ is obtained. The total description length DL_{n_1} of new structure $\mathcal{G}_{n_1}$ can be computed by adding the total description length of the old structure $\mathcal{G}_n$ and the difference between the local description lengths of X_i in $\mathcal{G}_n$ and $\mathcal{G}_p$. The description length $\mathcal{G}_{n_1}$ is can be evaluated by examining if this parents replacement process of node X_i in $\mathcal{G}_n$ does not affect the local description lengths of any other node in $\mathcal{G}_n$. There is a situation when this process fails, when the parents of X_i in $\mathcal{G}_p$ contains a reversed arc as compared to $\mathcal{G}_n$. Consequently we need to consider the node X_r associated with the reversed arc. If X_r is a *marked* node then we need to re-evaluate its local description length since it will be affected by the replacement of X_i's parents. By applying this process on all nodes, we can identify affected nodes that are connected, called a *marked subgraph unit*.

Therefore, we can find all marked subgraph units in $\mathcal{G}_p$. The parent replacement process will be done for all nodes in the subgraph, if the subgraph is chosen for the refinement. So, refinement process is reduced to choosing appropriate subgraphs to find a refined structure with lowest description length.

The authors, used a best-first search to find the set of subgraph units which gives the best reduction in description length without generating any cycle. To help the search process they maintain a set of subgraphs with ascending order of the benefit gained if parent substitution was to be performed using that subgraph. The

refinement process iteratively continued until no more subgraph substitution can be performed.

Remarks:

Lam and Bacchus called their approach as a refinement process of the existing network. They consider the new data in different way. It may represent the subset of the variables therefore, they learn a partial network and then use it to refine the existing network using a MDL principle. They consider the existing network as a summary of sufficient statistics of previous data, so they are dealing with data stream over sliding window by considering current data and sufficient statistics of old data. Their way to retrieve sufficient statistics is similar to the MAP approach (cf. section 4.2.1.3, page 65). Therefore, it is also biasing the search procedure towards old one. There is an implicit assumption that the new network structure is very similar to the existent structure.

4.2.1.3 Friedman and Goldszmidt's approach

Friedman and Goldszmidt [48] propose an approach for incremental Bayesian network learning, both parameters and structure. It maintains a set of network candidates that is called *frontier* of the search process, which consists of all the networks compared in each iteration of the algorithm. In sliding window, when a new data example arrives, the procedure updates the information stored in memory, sufficient statistics. Further, on the completion of current window it invokes the search process to check whether one of the networks in the frontier is deemed more suitable than the current model. The frontier set is updated every time when algorithm updates the Bayesian network structure. They compared their approach with two different common approaches, first naive approach which stores all previously seen data, and repeatedly invokes a batch learning procedure on each new example. Second approach based on Maximum Aposteriori Probability (MAP). They claim that effective incremental BN approach is that which gives a tradeoff between the quality, learned network and the amount of informations maintained about the past

observations. First we describe the common approaches as naive and MAP.

Naive approach:

Naive approach for incremental learning uses classical batch algorithms and store all previously seen data. It repeatedly invokes batch algorithm as each new data example arrives. This approach uses all of the observed informations seen so far to generate a model. Consequently, it can produce an optimal model in terms of the quality. It requires to store all the data or the counters for the number of times each distinct representation of all variables observed. This approach requires a vast amount of memory in the long run and vast amount of time to performs the search from scratch every time as new data observed. It becomes infeasible when network is expected to performs for long period of time.

MAP approach:

The MAP approach is motivated by Bayesian analysis (cf. section 3.3.4, page 40), where we start from a prior probability over possible hypotheses (models and their quantifications) and compute the posterior given our observations. In principle, we can use this posteriori as our prior for next iteration in the incremental learning process. So, we are maintaining our *belief state* about the possible hypothesis after observing D_{n-1}. Upon receiving D_n, we compute posterior to produce a model as our current belief state.

At each step, it approximates the *maximum a-posteriori probability* (MAP) network candidate. That is, the candidate that will be considered most probably given the data seen so far. Later, it approximates the posterior in the next iteration by using the MAP network as a prior network with the equivalent sample size. Therefore, it is a space efficient method as it is using the existing network as a summary of the data seen so far and we only need to store current data.

Incremental approach:

The *incremental* approach is the middle ground between two extremes Naive and MAP approach. There is a tradeoff between quality of learned networks and amount of sufficient statistics maintained in the memory. It focuses on keeping track of just enough information to make the next decision. It maintains a set of alternative networks "*frontiers*", consists of all the networks compared in each iteration. It also maintains the sufficient statistics of the *frontiers*. On the arrival of new data it invokes a greedy hill climbing search procedure to check if there is a network in the frontiers better than current network.

Here are some notations used in his approach: Let $Suff(\mathcal{G})$ is a set of sufficient statistics of graph $\mathcal{G}$, such that, $Suff(\mathcal{G}) = \{N_{ijk} : 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant r_i, 1 \leqslant k \leqslant q_i\}$ where N_{ijk} denotes number of examples where set of $Pa(X_i)$ takes its j^{th} configuration while the variable X_i takes its k^{th} configuration in the database D (with r_i and q_i as respective cardinalities of X_i and $Pa(X_i)$). Similarly, given a set S of sufficient statistics records, let $Nets(S)$ be the network structures that can be evaluated using the records in S, that is, $Nets(S) = \{G : suff(\mathcal{G}) \subseteq S\}$.

For example, there is a choice between two graphs $\mathcal{G}$ and $\mathcal{G}^{'}$. As we know, to evaluate these two graphs we need to maintain their sufficient statistics, $Suff(\mathcal{G})$ and $Suff(\mathcal{G}^{'})$ respectively. Now suppose that, these graphs $\mathcal{G}$ and $\mathcal{G}^{'}$ differ with only one arc from X_i to X_j. Consequently, there is a large overlap between $Suff(\mathcal{G})$ and $Suff(\mathcal{G}^{'})$, such as, $Suff(\mathcal{G}) \cup Suff(\mathcal{G}^{'}) = Suff(\mathcal{G}) \cup \{N_{ijk}\}$ (for each variable X_i, corresponding set of parents in graph $\mathcal{G}^{'}$). Therefore, we can keep a set of sufficient statistics for both graphs by maintaining a slightly larger set of statistics.

Now, we see how this generalization works and reduces a considerable subset of the search space of greedy hill climbing search. As we know greedy search compares its current candidate $\mathcal{G}$ with all its *neighbors*, which are the networks that are one change away (i.e. addition, deletion or reversal of an arc) from $\mathcal{G}$. We can generalize this set of neighbors by maintaining a limited set of sufficient statistics. For example, if S consists of all the sufficient statistics for $\mathcal{G}$, $Suff(\mathcal{G})$, and its neighbors. Then, $Nets(S)$ will contain additional networks, including networks that add several arcs

Algorithm 5 *Friedman and Goldszmidt's algorithm*

Require: *data D with tuples u.* $D = \{u_1, ..., u_n\}$; *initial network* $\mathcal{G}$.
Output: *DAG*: a network $\mathcal{G}$ of high quality with parameter θ.

Let F be initial search frontier for $\mathcal{G}$.
Let $S = Suff(\mathcal{G}) \cup \bigcup_{B' \in F} Suff(\mathcal{G}')$
while(true)
  Read data u_n.
  Update each record in S *using* u_n.
  if $n \mod k = 0$ then
    Let $\mathcal{G} = \arg\max_{\mathcal{G}' \in Nets(S)} Score(\mathcal{G}'|S)$
    Update the set of frontier F *(using a search procedure)*
    $S = Suff(\mathcal{G}) \cup \bigcup_{B' \in F} Suff(\mathcal{G}')$
  Compute optimal parameters θ for $\mathcal{G}$ from S.
  Output $(\mathcal{G}, \theta)$

---

in distinct families in $\mathcal{G}$. Note that if $X \subset Y$, then N_X can be recovered from N_Y. Therefore, $Nets(S)$ contains many networks that are simpler than $\mathcal{G}$.

This incremental approach can be applied to any search procedure which can define a *search frontier*. The search frontier is a set of those networks, which it will compare in next iteration. It is denoted by F, and it defines that which sufficient statistics should be maintained in the memory. That's why, set S contains all the sufficient statistics, which are required to evaluate networks in the frontier F. In the next iteration of the incremental learning process, when new example(s) received then the algorithm, first updates the set of sufficient statistics S. Then use this set S to evaluate networks in F and choose having a best scoring value. Finally, it invokes the search procedure to re-compute the set of frontier F for next iteration, and also updates S accordingly. So, this process may introduce new informations and may also remove some sufficient statistics.

Now, we can describe the main loop of the algorithm (cf. algorithm 5). It ensures if there is enough information that are sufficient to take new decision in the search space. Therefore, after every *k* examples it performs its decision. It also reallocates its resources by computing new set of frontiers and updating the sufficient statistics

correspondingly. For greedy hill climbing search, the set of frontiers is consisting all the neighbors of current network $\mathcal{G}$. Other search procedures may explore some of the neighbors, so that they require smaller set of frontiers. In the next section we see his definition of the score function for this incremental approach.

Scoring function for sequential update:

There are standard scoring functions (cf. section 3.4.2.1, page 45) in the literature, that are used to evaluate a set of structures. These functions make an assumption that we are evaluating all candidates with respect to the same dataset. On the contrary (as we see in previous sections) Friedman and Goldszmidt's keeps sufficient statistics for different families at different times. Therefore, they are comparing networks with respect to different datasets. So, they can not apply standard scoring functions directly.

The problem in search procedure is to compare models, e.g. if M_1 and M_2 are two models such that model M_1 is evaluated with respect to D_1 while model M_2 evaluated with respect to D_2. Here, Friedman and Goldszmidt assume that both data sets D_1 and D_2 are sampled from the same underlying distribution.

The MDL and the Bayesian scores are inappropriate for this problem in their current form. The MDL score measures how many bits are needed to describe the data, based on the assumption that the model is faithful to the underlying distribution of data. Therefore, if D_2 is much smaller than D_1 then usually the description length of D_2 would also be shorter than D_1, regardless of how good the model M_2 is. The same problem occurs with Bayesian score.

So, to solve this problem Friedman and Goldszmidt proposed an average description length per instance, it can be used to compare data sizes of different lengths. They compute the average local MDL score, $f_{averageMDL}$ (cf. for MDL score section 3.4.2.1) as :

$$f_{averageMDL}(X_i, Pa(X_i)) = \frac{f_{MDL}(X_i, Pa(X_i))}{\sum_{X_i, pa(X_i)} N(X_i, pa(X_i))} \tag{4.4}$$

Where N is the number of instances of the dataset. When we compare models based on different datasets, this score normalizes each evaluation to measure the average effective compression rate per instance.

To ensure that the score is choosing a right model, they defined an *inherent error* of a model M with respect to a reference distribution P:

$$Error(M, P) = \min_{\theta} D(P|P_M)$$

The *inherent error* used following lemma 1:

Lemma 1. *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two network structures that are evaluated with respect to datasets D_1 and D_2 of size N_1 and N_2, respectively. They are sampled i.i.d. from an underlying distribution P. If $Error(\mathcal{G}_1, P) < Error(\mathcal{G}_2, P)$, then as both N_1 and N_2 go to infinity, $Score_{MDL}(\mathcal{G}_1|D_1) < Score_{MDL}(\mathcal{G}_2|D_2)$ with probability 1.*

Remarks

First two approaches *Naive* and *MAP* are at the extremes of the tradeoff between the quality of obtained model and amount of the resources required to store sufficient statistics. While the third *incremental* approach gives a flexible manipulation of the tradeoff. The *MAP* approach is similar to that of Lam and Bacchus's, both store a single network as a summary of past data and it is space efficient. It avoids the storing overhead of the previously seen data instances. The *incremental* approach is similar to the Buntine's approach both maintains a set of candidate networks in the memory. It deals with data stream over sliding window and maintains a set of sufficient statistics of current network.

4.2.1.4 Roure's approach

In [121], Roure proposed an incremental Bayesian network approach which deals with data streams over landmark window. It rebuilds the network structure from the branch which is found to be invalidated. He proposed two heuristics to change a batch Hill-climbing search into an incremental one. Further, he applied

incremental hill-climbing on some well known *score-and-search* based algorithms such as Chow and Liu (CL) [29], $K2$ [32] and Buntine (B) [15]. In section 3.4.2.1, We discussed batch greedy/hill-climbing search in section. Here we can see how Roure transformed the batch hill climbing search procedure into incremental one.

Incremental Hill Climbing Search ($iHCS$):

In $iHCS$ (cf. algorithm 6), Roure discussed two main problems: firstly when and which part needs to be update, secondly to compute and store sufficient statistics. At each iteration it repeats the search path by traversing the reduced space of model domain. He proposed two heuristics: *Traversal Operators in Correct Order* and *Reduced search space*.

These heuristics are based on the assumptions that all data in the stream are sampled from the same probability distribution. Therefore, search space is supposed not to change too much when few data are added in current dataset or new data can slightly change the underlying distribution. Therefore, the scoring function is imagined to be a continuous over the space of datasets, in order to state how the quality of models will change when they are evaluated with respect to dataset having additional data.

First heuristic:

"Traversal Operators in Correct Order" (TOCO) verifies the already learned model and its search path for new data. If the new data alter the learning (search) path then it is worth to update an already learned model.

Definition 28. (TOCO heuristic) *Let D be a data-set, M be a model, Score(M, D) be a scoring metric, HCS a hill-climbing searcher,* $\{op^1, \ldots, op^k\}$ *be a set of traverse operators which given an argument A and a model M returns a new model M′. Also, let* $\mathbf{OpA}_{\mathcal{N}(M)}$ *be the set of operators and argument pairs with which neighborhood* $\mathcal{N}$ *of model M is obtained. Let M be a model learned by HCS where* $M = op_n^{k_n}(\ldots(op_2^{k_2}(op_1^{k_1}(M_0, A_1), A_2)\ldots, A_n)$ *is built with the search path,*

Algorithm 6 *Incremental Hill Climbing Search ($iHCS$)*

Require : *data D; scoring function $Score(M,\ D)$, a set of operators $OP = \{op^1, \ldots, op^m\}$ and $\mathcal{B}_i$ is a set of k best operators and argument pairs for model M_i*, an integer $q \leqslant k$ which states the number of pairs in $\mathcal{B}$ to be used.

Output : *DAG : a model M of high quality*

** TOCO **\

Verify previous search path : After evaluation of scoring function over new data $D \cup D'$, let $(op_j,\ A_j)$ be a last pair where the new data agree with previously learned search path:
$M_{ini} = (op_j(...(op_1(M_0,\ A_1),\ A_2)...),\ A_j)$

if ($M_{ini} = M_{final}$ **)** **then**

Use $k = q$

Endif

i = 0

$M_i = M_{ini}$

** RSS **\

Repeat

$oldScore = Score(M_i,\ D)$

i++

$M_i = op(M_{i-1},\ A_i)$

** where

$op(M_{i-1},\ A_i) = arg\,max_{(op^m,\ A) \in \mathcal{B}_i}\ Score\,(op^m(M_{i-1},\ A_i),\ D)$

**\

if ($M_i \neq M_{final}$ **)** **then**

Recalculate $\mathcal{B}_k$

Endif

Until $oldScore \geqslant Score(M_i,\ D)$

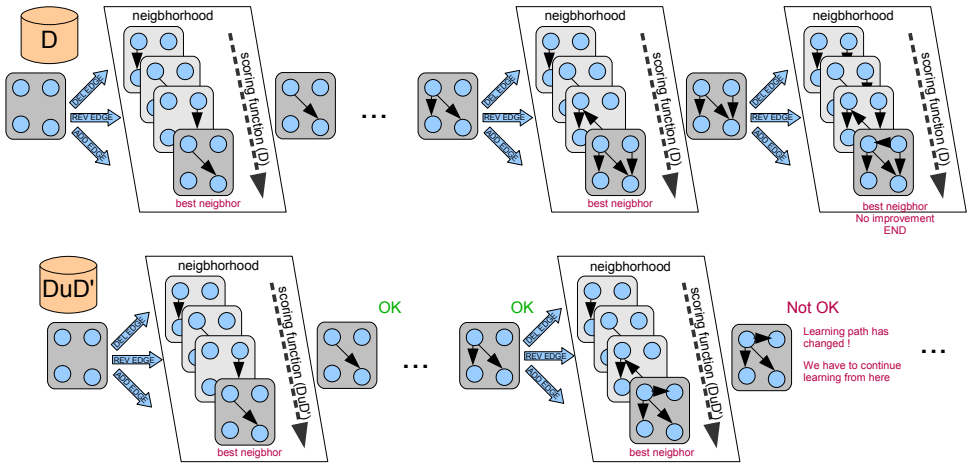


Figure 4.1: *Outline of TOCO heuristic*

$\mathcal{O}_{Op} = \{(op_1^{k_1}, A_1) \ldots, (op_n^{k_n}, A_n)\}$ *where*

$$\forall i \in [1, n] : (op_i^{k_i},\ A_i) \quad = \quad argmax_{(op_i^{k_i}, A) \in \mathbf{OpA}_{\mathcal{N}(M_{i-1})}} Score(op_i^{k_i}(M_{i-1},\ A_i),\ D)$$

Let D' be a set of new data instances, the TOCO heuristic states that HCS learns a new model $M' = op_{n'}^{k'_{n'}}(\ldots(op_{2'}^{k_{2'}}(op_{1'}^{k_{1'}}(M_{ini},\ A_{1'}),\ A_{2'})\ldots,\ A_{n'})$ corresponding to D_t=$\{D \cup D'\}$ where traverse operators and arguments are obtained as before

$$\forall i \in [1, n'] : (op_i^{k'_i},\ A_i) \quad = \quad argmax_{(op_i^{k'_i}, A) \in \mathbf{OpA}_{\mathcal{N}(M_{i-1})}} Score(op_i^{k'_i}(M_{i-1},\ A_i),\ D_t)$$

and where the initial model is $M_{ini} = op_j^{k_j}(\ldots(op_2^{k_2}(op_1^{k_1}(M_0,\ A_1),\ A_2)\ldots,\ A_j)$ where $\left(op_j^{k_j},\ A_j\right)$ is the last operator and argument pair which is in correct order, that is, the last pair that produces the model with the highest score among pairs in the search path $\mathcal{O}_{Op}$

$$(op_{j+1}^{k_{j+1}},\ A_{j+1}) \quad \neq \quad argmax_{(op_i^{k'_i}, A) \in \mathbf{OpA}_{\mathcal{N}(M_{i-1})} \cap \mathcal{O}_{Op}} Score(op_i^{k'_i}(M_{i-1},\ A_i),\ D_t)$$

Roure keeps the search path of the former learning step (cf. figure 4.1). On the arrival of new data instances, if the order of the previous search path is changed then we conclude two reasons. First reason, since we consider the quality function is continuous and $P(D)$ is significantly different than new data $P(D \cup D')$, therefore the newly arrived data provides new informations. Second reason is the search

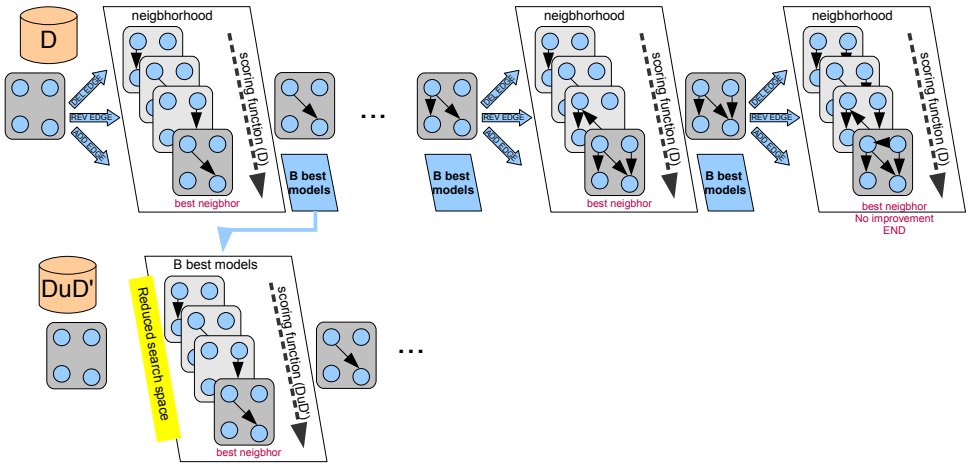


Figure 4.2: *Outline of RSS heuristic*

procedure follows a different search path over the model domain and the order of the search path is no more validated. Therefore, possibly it will produce a different model so, it will trigger the algorithm to revise the structure.

On the other hand, if the order still holds, we assume that $P(D)$ and $P(D \cup D')$ are very similar and hill climbing search will follow again the same path and obtain the same model.

When the order of search path is no more validated and the algorithm triggered to revised the structure, then the revision process will start from the point where algorithm triggered. So that, the path found before this triggered point will be considered as a valid and it will serve as a starting point (initial model) for the hill climbing search algorithm.

As a conclusion, TOCO heuristic allow to revise the model, only if the order of search path is invalidated and the re-learning process will not starts from scratch.

Second heuristic:

"*Reduced search space*" (RSS) applies when the current structure needs to be revised. At each step of the search path, it stores top *nRSS* models in a set $\mathcal{B}$ having the score closer to the best one. The *nRSS* is a parameter of the algorithm to control size of the search space. It avoids to explore those parts of the space where low quality models were found during former search steps.

Definition 29. (RSS heuristic) *Let D be a data-set, M be a model, Score(M, D) be a scoring metric, HCS a hill-climbing searcher,* $\{op^1, \ldots, op^k\}$ *be a set of traverse operators which given an argument A and a model M returns a new model M′. Also, let* $\mathbf{OpA}_{\mathcal{N}(M)}$ *be the set of operators and argument pairs with which neighborhood* $\mathcal{N}$ *of model M is obtained. Let M be a model learned by HCS where* $M = op_n^{k_n}(\ldots(op_2^{k_2}(op_1^{k_1}(M_0, A_1), A_2)\ldots, A_n)$ *is built with the search path,* $\mathcal{O}_{Op} = \{(op_1^{k_1}, A_1)\ldots, (op_n^{k_n}, A_n)\}$ *where*

$$\forall i \in [1, n] : (op_i^{k_i},\ A_i) \quad = \quad argmax_{(op_i^{k_i}, A) \in \mathbf{OpA}_{\mathcal{N}(M_{i-1})}} Score(op_i^{k_i}(M_{i-1},\ A_i),\ D)$$

and finally let $\forall i \in [1,\ n] : \mathcal{B}_{\left(op_i^{k_i},\ A_i\right)}$ *be a set of the k operator and argument pairs the score of which is closer to* $Score(op_i^{k_i}(M_{i-1},\ A_i),\ D)$*, where k is a given positive number. Keep a set* $\mathcal{B}$ *for each pair in* $\mathcal{O}_{op}$*.*

Let $D\prime$ *be a set of new data instances, the RSS heuristic states that HCS learns a new model* $M' = op_{n'}^{k'_{n'}}(\ldots(op_{2'}^{k_{2'}}(op_{1'}^{k_{1'}}(M_{ini},\ A_{1'}),\ A_{2'})\ldots,\ A_{n'})$ *corresponding to* D_t=$\{D \cup D'\}$ *where traversing operators and arguments are obtained using* $\mathcal{B}_{\left(op_i^{k_i}, A_i\right)}$

$$\forall i \in [1, n'] : (op_i^{k'_i},\ A_i) \quad = \quad \arg\max_{(op_i^{k'_i}, A) \in \mathbf{OpA}_{\mathcal{N}(M_{i-1})} \cap \mathcal{B}_{\left(op_i^{k'_i},\ A_i\right)}} Score(op_i^{k'_i}(M_{i-1},\ A_i),\ D_t)$$

RSS heuristic is outlined in the figure 4.2. Given a model M, hill climbing search proceeds in an iterative way and generates intermediate models in order to find a best final model. At each step it explores the neighborhood of the model and stores *nRSS* best intermediate models in the memory. When new data is available, the incremental hill climbing algorithm restricts the search space by exploring only its *nRSS* best neighborhood found in previous learning step. Again RSS is also based upon the same assumption that the data has been drawn from a same probability distribution.

The incremental hill climbing algorithm ($iHCS$) has two phases (cf. algorithm 6). First phase uses TOCO heuristic and checks if the learning path of previous learning step is in correct order or not. It also assigns the initial model M_{ini} the state, from which the search will be resumed. The second phase RSS, performs the search

to update the model using new data.

There are two different situations in which the $iHCS$ algorithm gets to the repeat statement. The first one happens when the TOCO heuristic finds an operator and argument pair incorrectly ordered $(M_{ini} \neq M_{for})$ and, in consequence, the $iHCS$ fires the revising process. The second one happens when the TOCO heuristic finds all operators in correct order $(M_{ini} = M_{for})$ and thus the former model is still valid and the $iHCS$ algorithm tries to use new operators just in case the former model can be improved.

Remarks:

Roure assumes the resulting model will be very similar to the current model. By reducing the size of *nRSS*, the search space will be very much reduced and the resulting model will be biased towards the current one. This incremental approach deals with data streams over landmark window, further Roure proposed a solution to store all possible sufficient statistics of data stream in the memory using an adaptation of AD-tree structure [123]. We recall our readers that *ADtree* data structure used to cache sufficient statistics. It reads through the dataset once and pre-computes the answer to every possible counting query. Therefore this way will wasts a lot of computational time to maintain those sufficient statistics that will not be used by the $iHCS$ algorithm.

Roure follows the same underlying idea of Friedman and Goldzmidts (cf. section 4.2.1.3), and Buntine's (cf. section 4.2.1.1) proposals, as they keep in memory a set of candidate networks to evaluate them in the light of new data.

4.2.2 Hybrid method

Approaches discussed in previous section deal with the stationary domains and use search and score methods to learn BN structure incrementally. As we discussed in section 3.4.2, search and score methods has their own limitations over hybrid methods. As far as we know, all the techniques proposed to learn a structure of the Bayesian network incrementally, are based upon the *score-and-search* method,

Algorithm 7 *Shi and Tan's algorithm*

Require: *Old data D_{old}, new data D_{new} and the current network structure $\mathcal{G}_c$.*
Output: *DAG*: a network $\mathcal{G}$ of high quality.

Let $D = D_{old} + D_{new}$

** Compute candidate parent sets **\
Foreach variable X_i

Build a candidate parent set $candP_{X_i}$, using some polynomial-time constraint-based technique.

end for

** Refining the current network structure **\
Apply greedy hill-climbing search on $\mathcal{G}_c$ using three operators; add_edge, remove_edge and reverse_edge. Apply add_edge $X_i \rightarrow X_j$ if and only if $X_j \in candP_{X_i}$.

Output *optimal network structure $\mathcal{G}$*

except there is only one proposal from Shi and Tan [129, 128] based on hybrid method and proposed in parallel of our own approach described in next chapter 5. In this section we discuss their approach and see how it works.

Shi and Tan's approach

Recently, Shi and Tan adapted a hybrid approach to incremental Bayesian network learning over landmark window. In first phase, they build a candidate parent set for each variable in dataset using constraint based technique. Later in second phase, they used a constrained hill climbing search procedure to refine the network using already obtained candidate parent sets.

We can see in the algorithm 7, in its first phase, they compute a set of expected parents for each variable in the dataset. Later, it finds an optimal network structure using greedy hill climbing search. They store all previously seen data and run incremental learning process over landmark window. They proposed two ways to compute a set of candidate parents.

First work:

In their first work [129], to learn a set of candidate parents for each variable, they used a constraint based method, which is very similar to PC algorithm [132]. Therefore, it has the same limitations as it also starts from a full DAG.

First, the candidate parent set $candP_{X_i}$ for variable X_i contains all other variables except itself (X_i). Next, it checks for independence, if there is a variable in candidate parent set that is found independent, remove this variable from candidate parent set. Let X_j is found independent of X_i, $(X_i \perp X_j)$, then X_j will be removed from the candidate parent set $candP_{X_i}$. They use a mutual information measure to find the conditional independence. For incremental process, it keeps record of those variables which are found independent in dataset D_{old}, therefore in next iteration of incremental process, first it checks for these variables if the independence is still exist in $D_{old} + D_{new}$.

For each variable X_j in $candP_{X_i}$ it applies a heuristic to find a set of variables which may d-separate X_i from X_j conditionally. For this purpose, it constructs a maximum spanning tree of current network using Chow and Liu's algorithm [29]. The current network may be an initial network or the network obtained in the previous incremental learning step. Further variables which are neighbors of X_i and found in the path of X_i to X_j are extracted. The extracted variables are combined together as C_X. According to the concepts of *directed separation and undirected separation* [107], if X_i and X_j are not connected directly then there may be some subset of $C_{\mathbf{X}\setminus X_i,X_j}$ that can conditionally separate X_i and X_j, then X_j will be removed from $candP_{X_i}$.

Second work:

In second work [128], they proposed two different techniques to compute the set of parent children. In *tree-shaped structure* based technique (*TSearch*), first they learn an undirected structure using *MWST* algorithm of Chow and Liu [29]. Then, for each variable X_i, it extracts neighbors and the neighbors of its neighbors in the undirected structure. In addition, the Markov Blanket and the parents of its parents

of same variable X_i are also extracted from the current Bayesian network. Further these extracted variables are merged together as a candidate parent set of variable X_i.

The second technique is a *feature selection* based, where they use a feature selection method proposed in [116] to extract a candidate parent set for a variable X_i. In [116], the authors propose three simple polynomial-time feature selection techniques to select candidate parents for each variable. Shi and Tan have not mentioned the name of particular technique they used for feature selection but they propose to use any polynomial-time technique.

Remarks:

As Roure, Shi and Tan's incremental methods also learn over landmark window. They do not store sufficient statistics any more. The first way to compute the candidate parents is very similar to PC algorithm [132] (cf. section 3.4.2.2, page 48). Therefore, it has the same limitations as PC. It starts from full DAG and keeps record of the independent variables. In the case of high dimensional domains, the list of independent variables may become very large.

The second way to compute the set of candidate parents is not incremental. It is a simple adaptation of respective batch algorithms using whole dataset.

4.3 Learning from non-stationary domains

All the methods seen so far assume that the data has been produced by a stationary domains so, underlying distribution remains same. In real world online applications this assumption is not feasible because some unknown events may occur e.g. network attack or non observable effects may change suddenly. On the other hand algorithms deal with non-stationary domains consider the underlying distribution of data may change with time so, drift or shift change may occur.

Nielsen and Nielsen's approach

There is only one proposal from Nielsen and Nielsen [109] addressing this issue for incremental Bayesian network structure learning as *structural adaptation*. They consider the non-stationary domains where concept shift may occur. Their method monitors the data stream in an adaptive sliding window where it checks if last, say *k*, observations fit to the current model. If it is not the case this means *shift* took place *k* observations ago. Therefore, if we relearn model from last *k* cases, we will lost informations learned before *k* cases. So, their method identifies the changed parts of the two distributions and relearn only those parts that are changed. Hence, this approach consists of two mechanisms, first monitoring and detecting when and where the model should be changed and second, relearning and integrating the parts of the model that conflict with the observations, using a *constraint based* local search strategy. In the next paragraph we discuss in detail.

Change detection:

The change detection part continuously processes the examples as it receives. It computes the *conflict measure* for each example *v* of data D and set of nodes *X*. *Conflict measure* checks that how well *v* fits with the local structure $\mathcal{B}$ around variable X_i using approach of Jensen et al [78]: *If the current model is correct or at least a good predictor of future observations, then we would in general expect that the individual elements of an observation v are positively correlated (unless v is a rare case, in which all bets are off)*:

$$\log \frac{P_{\mathcal{B}}(X_i)}{P_{\mathcal{B}}(X_i|X_j\ (\forall j \neq i))} < 0 \tag{4.5}$$

So, conflict measure returns the value given on the left-hand side of the equation 4.5. It records the history of conflict measures as $\mathbf{c}_{X_i}$ for node X_i.

We cannot use the value returned by conflict measure directly to determine whether a shift occur. Because a high value obtained may be caused by a rare case. Therefore, based on the $\mathbf{c}_{X_i}$ the method *ShiftInStream(c_{X_i}, k)* checks whether a shift

occurred before *k* observations or not. So, *ShiftInStream* method ensure if there is a trend towards the high/low values and it is not caused only by rare cases and a shift must have occurred.

Nielsen and Nielsen first calculate the negative of the *second discrete cosine transform* (DCT) component [117] for the last *2k* measures $c_1, ..., c_2k$ in c_{X_i}:

$$C_2 \equiv \sum_{j=1}^{2k} c_j \, cos\left(\frac{\pi(j+\frac{1}{2})}{2k}\right) \tag{4.6}$$

The component (eq. 4.6) calculated after reception of each observation tells how much there is a tendency for the last *2k* conflict measure values to be arranged on a line with positive slope. A high value therefore indicates that the last k cases are more in conflict with the model than those from before these.

When actual change/shift detection has taken place, then as a last step the detection module invokes the structure updating module for the set of nodes, for which it detected a change, together with the last *k* observations.

Learning and updating structure:

Now we have a set of shift involving nodes *C* and we have to adapt previously learned model $\mathcal{B}$ around these nodes. So that it fits the distribution of the data. Since we want to reuse the information encoded in $\mathcal{B}$ that has not been outdated by the previous phase. Therefore, we will update $\mathcal{B}$ to fit the distribution of new data D' based on the assumption that only nodes in *C* need updating of their probabilistic bindings.

In *learn fragment phase* it learns a partially directed fragments $\mathcal{G}_{X_i}$ for each nodes in *C*. Then corresponding fragments in current model $\mathcal{B}$ are also extracted for each node in *C*. All the fragments then merged into a single graph $\mathcal{G}'$, using four direction rules that preserve as much of $\mathcal{B}$'s structure as possible and without violating the newly discovered informations. Finally, it computes parameters for those nodes that have new parents in the final graph.

It uses the constraint based technique to find the graph fragments for each node

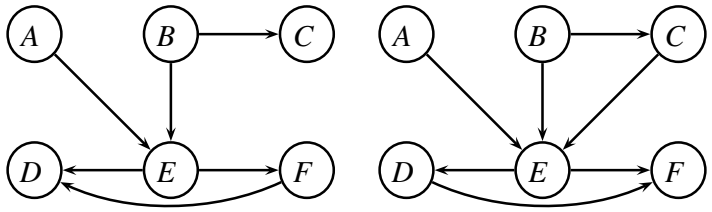
in $\mathbf{X}$. The variables that are conditionally independent of node X_i conditioning on some set of variables, are non-adjacent to X_i. Similarly, the variables found dependent of node X_i are adjacent to X_i in the graph fragments. For independence test, they used Pearson's χ^2 test [117]. Mostly constraint based learning methods discover only the direction of the arcs contributing in *v-structures* using independent tests and for the direction of remaining arcs, they apply some structural rules.

The *MergeFragments* module combines graph fragments of all variables in *X* using simple graph union, and preference given to arcs (directed) over links (undirected). Further, the remaining links in final graph directs using following four rules and ensure that they are not introducing new v-structures in the final graph:

1. If $A - B$ is a link, $C \rightarrow A$ is an arc, and B and C are not adjacent then direct the link $A - B$ as $A \rightarrow B$
2. If $A - B$ is a link and there is a directed path from A to B, then direct the link $A - B$ as $A \rightarrow B$.
3. If Rules 1 and 2 can not be applied, chose a link $A - B$ at random such that A in the nodes that are found unchanged in the first phase of the algorithm, and direct it as in current structure $\mathcal{B}$. During this rule try to preserve parent sets.
4. If Rules 1 and 3 cannot be applied, chose a link at random and direct it randomly.

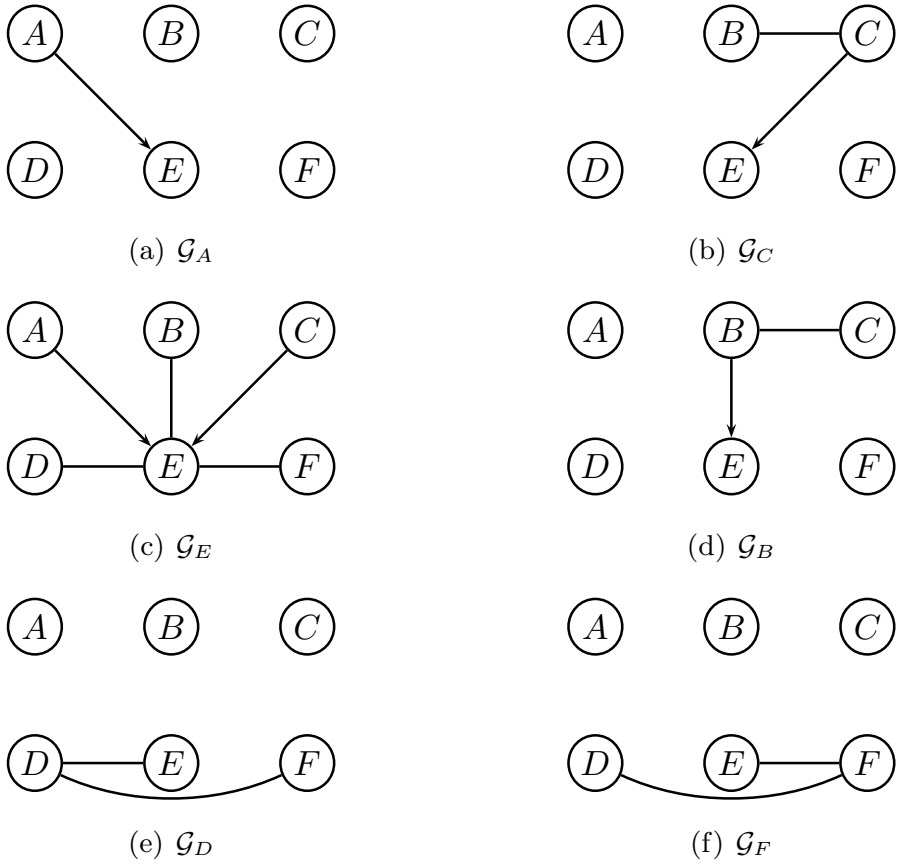
For more clarification, authors used an example of two networks $\mathcal{B}_1$ and $\mathcal{B}_2$ (cf. 4.3) with six nodes *A, B, C, D, E* and *F*. The sequence of data generated from these networks, first n_1 from $\mathcal{B}_1$ and then n_2 from $\mathcal{B}_2$. Adjust the value of *k* less than $n_1/2$ and n_2. For example, after reading *2k* data instances, when the algorithm read the *k'th* instance of the part of data, generated from $\mathcal{B}_2$. Therefore, a jump should be detected for nodes *A, C,* and *E*, as each has gotten a new Markov boundary in $\mathcal{B}_2$. Now, algorithm will update $\mathcal{B}$ around the nodes $C = \{A, B, C\}$ based on the last *k* instances (samples from $\mathcal{B}_2$).

UpdateNet module first learns fragments for A, C, and E, which are shown in figures 4.4(a), 4.4(b) and 4.4(c). These fragments are learned only from new data D'.



(a) $\mathcal{B}_1$ (b) $\mathcal{B}_2$

Figure 4.3: *Two BNs from which a DAG faithful sequence has been sampled* [109].

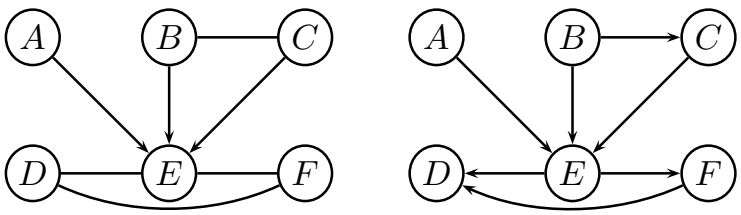


(a) $\mathcal{G}_A$ (b) $\mathcal{G}_C$

(c) $\mathcal{G}_E$ (d) $\mathcal{G}_B$

(e) $\mathcal{G}_D$ (f) $\mathcal{G}_F$

Figure 4.4: *Learned and extracted graph fragments matching* $\mathcal{B}_2$ [109].



(a) $\mathcal{G}_2'$ (b) $\mathcal{G}_2$

Figure 4.5: *(a) The merged graph fragment and (b) the fully directed version* [109].

Next, extract the nodes B, D and F from current model $\mathcal{B}$, these nodes was found unchanged in previous change detection phase. These are shown in the figures 4.4(d), 4.4(e) and 4.4(f). These learned and extracted fragments are merged into the graph in figure 4.5(a). The algorithm ends by directing the remaining links as shown in figure 4.5(b).

Remarks:

Nielsen and Nielsen in their proposal addressed a special case of incremental Bayesian network structure learning that is *structural adaptation*. They assume that the existing model structure is faithful to the previous data. The algorithm learns over sliding window with adaptive size, it keeps only k recent data examples and monitors whether it fits with the existing model structure or not. So, it triggers re-learning process for only those parts of the structure, which gives a considerable increase in the conflict measure. Therefore, the gradual change do not trigger the update process. To learn the conflicted fragments of the structure, it uses only recent k data instances. It is a *constraint* based approach for Bayesian network structure learning.

4.4 Conclusions

In the literature there are few approaches addressing incremental Bayesian network structure learning issue. These approaches are taking into account certain scenarios depending upon the nature of data and structure learning methodology. *Buntine's* approach (cf. section 4.2.1.1), *Lam and Bacchus's* approach (cf. section 4.2.1.2) along with the approach of *Friedman and Goldszmidt* (cf. section 4.2.1.3) learn from a data stream over sliding windows. They store sufficient statistics of previous data in a memory. They rely on the MAP approach (cf. section 4.2.1.3) for the computation of these statistics. Where as, *Roure* (cf. section 4.2.1.4) along with *Shi and Tan* (cf. section 4.2.2) use whole data as a landmark window. On the other hand, *Nielsen and Nielsen* adapt incremental process in different way. They use tumbling window with variable size.

As a conclusion, we see the most of the algorithms use the *scoring-and-search* based methods and treat the incremental process in different ways for *stationary domains*. Score-and-search based methods are not scalable for data stream mining, the judicious choice for handling a great number of variables is a local search hybrid approach. As mentioned in section 3.5, there is an encouraging trend to find an approach that might be applicable to the high dimensional domains, which cope with the accuracy of final model and the learning cost.

II

Propositions

5

Incremental Max-Min hill climbing

Contents

5.1 Introduction 89
5.2 Constraint based incremental local search 90
5.2.1 Principle 90
5.2.2 Objective 90
5.2.3 Incremental MMPC 90
5.2.4 Incremental $\overline{\mathbf{MMPC}}(\mathbf{T})$ 92
5.2.4.1 Forward phase of $\overline{MMPC}(T)$ as a greedy search 92
5.2.4.2 Algorithm 94
5.2.4.3 An illustrative example 97
5.2.5 Complexity 99
5.3 Incremental Max-Min Hill-Climbing 100
5.3.1 Principle 100
5.3.2 Objective 101
5.3.3 Incremental local search 101
5.3.4 Incremental global optimization 101
5.3.5 Comparison with existing methods 103
5.3.6 Optimizing frequency counting 103
5.3.7 Complexity analysis 105
5.4 Adaptation of iMMHC for sliding window 106
5.4.1 Principle 106
5.4.2 Objective 107
5.4.3 Update of sufficient statistics 107

5.4.4 Complexity 108
5.5 Adaptation of iMMHC for damped window 109
5.5.1 Principle 109
5.5.2 Objective 109
5.5.3 Forgetting mechanism 109
5.6 Conclusions 110

5.1 Introduction

Incremental Bayesian network learning from data stream includes both, structure learning of a model, $\mathcal{G}$, and estimating the set of model parameter values. Learning the parameters given a fixed model structure is a straightforward task (even in incremental environment). We are considering only incremental structure learning. Bayesian network structure learning is a NP-hard problem [22]. Traditional batch structure learning algorithms have size limit with hundreds of variables [35]. As we saw in chapter number 4, the existing state of the art in incremental structure learning involves only several tens of variables and methods do not scale well in high dimensional domains. The recent explosion of high dimensional datasets in the field of social networks, bioinformatics, telecommunication etc, having several hundreds and thousands of variables impose a serious challenge for existing incremental Bayesian network structure learning algorithms.

In this chapter, we introduce a constraint based incremental local search method called *incremental Max-Min Parent Children* ($iMMPC$) and later *incremental Max-Min Hill Climbing* ($iMMHC$) to obtain a final Bayesian network structure. These methods are inspired by the idea of *score-and-search* based technique *incremental Hill Climbing Search* (iHCS) and the local search hybrid approach *MMHC*. They handle data stream over landmark window. Finally, we adapt these algorithms on sliding and damped windows.

This chapter organized as follows, section 5.2 introduce proposed constraint based incremental local search approach. We also explain how a *constraint* based local search approach can be seen as a *score-and-search* based *greedy/hill-climbing* search. Later, we present a hybrid algorithm $iMMHC$ for incremental Bayesian network structure learning from landmark window in section 5.3. In section 5.4, we propose a frequency counter to store sufficient statistics over sliding window and section 5.5 introduces a forgetting mechanism to keep up to date model with the current data. Finally, we conclude in section 5.6.

5.2 Constraint based incremental local search

5.2.1 Principle

We saw in state of the art (cf. section 4.2.1, page 57) of incremental BN structure learning, most of the existing methods use *score-and-search* based techniques and treat the incremental process in different ways. The key advantage of local search methods over *score-and-search* is to scale up to hundreds of thousands of variables. Therefore, the judicious choice for handling a great number of variables is a local search approach. These are most scalable as they search for the conditional independence relationships among variables in a dataset and construct a local structure around the target. Therefore, we adapt a *local search* based approach, *MMPC* (cf. section 3, page 51) for skeleton discovery. This approach has been proved as a robust solution in order to limit the number of false positives [138]. So, here we propose a constraint based incremental local search method *iMMPC*. It deals with data stream over landmark window as defined in figure 2.5 on page 22.

5.2.2 Objective

The objective is to incrementally learn the local skeleton (undirected structure) for each variable by reducing the search space. Therefore, we store the most strong dependencies found in the search process as a candidate neighborhood to further use in next incremental learning process. By this way we reduces the search space for newly arrived data by considering only strong dependencies and resuming the search process from the point where this data invalidate the previous hypothesis.

5.2.3 Incremental MMPC

Here, we explain how developed constraint based incremental local search approach works, while it is outlined in the figure 5.1. The $iMMPC$ algorithm differs from original $MMPC$ algorithm (cf. algorithm 3, section 3.4.2.3) in such a way that we call $iMMPC$ algorithm for all variables in the dataset rather than only for a

Algorithm 8 $iMMPC(D_w)$

Require: Data of landmark window w_i (D_{w_i}), set of previously found top K best neighbors for each variable ($\mathcal{B}_{i-1}$)
Output: A skeleton $\mathcal{G}$

** ***Call to*** $\overline{\mathbf{iMMPC}}$ **\

$CPC(\mathbf{X}) = \varnothing$
For $j = 1$ *To* n **do**

$CPC(X_j) = \overline{iMMPC}(X_j, D_{w_i}, \mathcal{O}_{\mathcal{OP}}, \mathcal{B}_{i-1})$
Save search path $\mathcal{O}_{\mathcal{OP}}$ and a set $\mathcal{B}_i$ for variable X_j

Endfor

** ***Symmetrical correction*** **\

For each *variable* $Y \in CPC(X)$ **do**

if ($X \notin CPC(Y)$) **then**
$CPC(X) = CPC(X) \backslash Y$

EndIf

Endfor

** ***Construct a skeleton*** $\mathcal{G}$ **\

For all variables in $\mathbf{X}$, add an undirected edge $X_i - X_j$ in $\mathcal{G}$
if and only if $X_i \in CPC(X_j)$

Return Skeleton $\mathcal{G}$

single target variable. It also generates an undirected graph from the obtained CPCs.

First, it initializes the set of CPCs to empty sets for all variables in $\mathbf{X}$, then for each variable it calls $\overline{iMMPC}$ algorithm. So, $\overline{iMMPC}$ computes CPCs with the help of previously found K best variables and search path, $\mathcal{O}_{\mathcal{OP}}$, it also updates these two parameters for all variables as discuss in next section.

Next, it performs symmetrical corrections for all variables. Let us say variable Y is in the set of CPCs of variable X then it tests if X is also in the set of CPCs of variable Y. If it does not exist then it removes variable Y from the set of CPCs of X. Therefore the relation between parent and children remains symmetric.

Finally, it constructs an undirected graph (skeleton) for all variables using their sets of CPCs. So, it adds an undirected edge between two variables X_i and X_j, if X_i is found in the CPC of X_j.

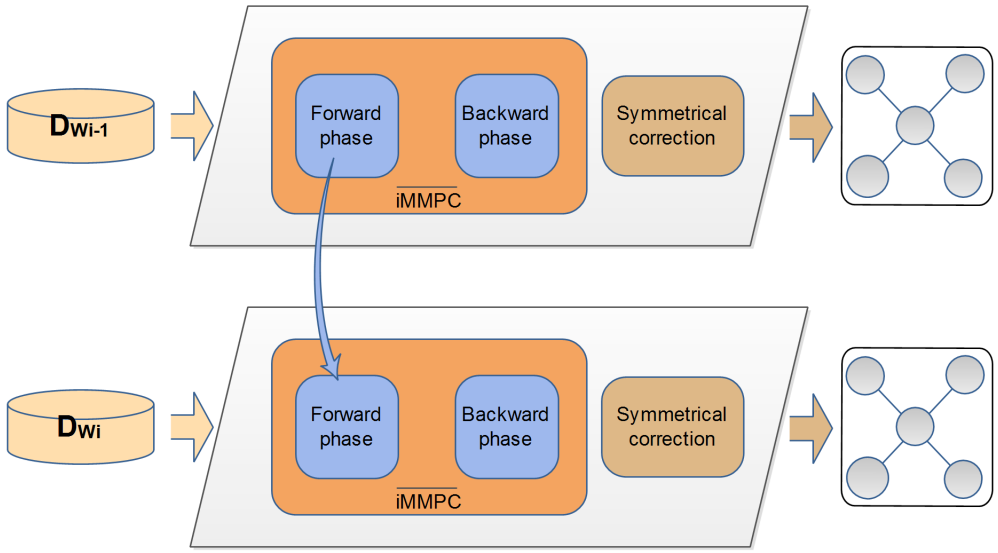


Figure 5.1: *Outline of $iMMPC$ algorithm*

5.2.4 Incremental $\overline{\text{MMPC}}(\text{T})$

$\overline{iMMPC}(T)$ algorithm tests for conditional independence for all remaining variables with the target. The conditionally dependent variables are added in the set of candidate parent children (CPC) in such a way that if X and T are dependent variables given a subset of variables $\mathbf{Z}$, ($X \not\perp T \mid \mathbf{Z}$), then X is added in the set of CPC of target variable T, CPC(T). On the other hand if X and T are found to be independent given a subset of variables $\mathbf{Z}$ then there will be no edge between X and T in the final graph. $\overline{iMMPC}(T)$ works in two phases (cf. algorithm 5.2.4.1). In the first forward phase, it sequentially adds variables in the CPC and later in second backward phase it removes the false positives from the set of CPC. In order to adopt Roure's TOCO and RSS heuristics (cf. section 4.2.1.4, page 69) first we need to show the forward phase works as a greedy search.

5.2.4.1 Forward phase of $\overline{MMPC}(T)$ as a greedy search

As discussed in section 3.4.2.3 the pivotal part of the *MMPC(T)* algorithm is the forward phase of $\overline{MMPC}(T)$. In this phase variables enter incrementally in the set of CPC(T) of a target variable T using *MaxMinHeuristic* (cf. equation 3.21). This heuristic first finds the minimum association of each variable X relative to

a subset S of already selected CPC, denoted as "$min_{S \subseteq CPC} Assoc\,(X;\, T \mid S)$". Later, it selects the variables as a CPC that maximizes the minimum association with target variable T. It requires to demonstrate this part acts as a hill-climbing process in a decreasing order. After consulting the current available literature and experimentation, mathematically it is difficult to demonstrate it, for example:

Example: Let's assume four variables X_1, X_2, X_3 and X_4. The target variable $T = X_1$ and we have to compute its set of CPC using mutual information as an association measure. At its *first iteration*, we compute $MI(X_1, X_2)$, $MI(X_1, X_3)$, $MI(X_1, X_4)$. Let's consider obtained MI are ordered in a decreasing way as:

$$MI(X_1, X_2) > MI(X_1, X_3) > MI(X_1, X_4)$$

Therefore, we choose X_2 and $CPC = \{X_2\}$. Now for *second iteration*, we compute $MI(X_1, X_3)$ and $MI(X_1, X_3|X_2)$, and we have to choose the minimum one. Similarly, we compute $MI(X_1, X_4)$ and $MI(X_1, X_4|X_2)$, and choose the minimum one. Then select the maximum from previously chosen minimums.

If the maximum is $MI(X_1, X_3|X_2)$, then we know that $MI(X_1, X_3|X_2) \leqslant MI(X_1, X_3) \leqslant MI(X_1, X_2)$. And if the maximum is $MI(X_1, X_4|X_2)$, then we know that $MI(X_1, X_4|X_2) \leqslant MI(X_1, X_4) \leqslant MI(X_1, X_2)$. Therefore, in both cases value of association measure decreased. We assume $X3$ is selected in this iteration and $CPC = \{X_2, X_3\}$.

In its third iteration, we compute $MI(X_1, X_4)$, $MI(X_1, X_4|X_2)$, $MI(X_1, X_4|X_3)$ and $MI(X_1, X_4|X_2, X_3)$ and keep the minimum one which is $\leqslant MI(X_1, X_4)$. It is easy to show that $MI(X_1, X_4|X?)$ [1] $< MI(X_1, X_2)$, but mathematically it is difficult to prove that $MI(X_1, X_4|X?) < MI(X_1, X_3|X?) < MI(X_1, X_2)$.

While during our experimentation we found that the value of objective function decreased between two iterations for selected variables with respect to the target variable i.e. $MI_{1^{st}\,iteration} > MI_{2^{nd}\,iteration} > ... > MI_{n^{th}\,iteration}$. It was observed in several experiments over different datasets. There is an obvious pattern that

1. $X?$ could be a X_2 or X_3 variable or nothing.

suggests the conjecture that:

Conjecture 1. *The forward phase of* $\overline{MMPC}(T)$ *is a greedy/hill-climbing search in a decreasing order using* $Assoc = MI$.

Therefore, in this study we make an assumption that the objective function decreased at each iteration of local search for a set of CPCs of a target variable.

As seen in section 3.4.2.1, the main components of greedy search are:

– a model

– a scoring function to measure the quality of models

– the search space

In local search environment, $\overline{MMPC}(T)$ finds the set of candidate parent-children for a target variable T step by step (without distinguishing among both parent and children variables). Here, we can say the local model for a target variable T consist of its CPCs.

We use the *MaxMinHeuristic* as a scoring function to measure the association between variables, therefore, new variables added in the set of CPCs of a target variable T on the basis of this function. Its value decreases between two iterations, as discussed in the example of section 5.2.4.1 and our assumption.

The search space for this local model is compromises of all variables V except the target variable. The variable found conditionally/unconditionally independent will remove from the search space.

5.2.4.2 Algorithm

As discussed in the previous section, the $\overline{MMPC}(T)$ algorithm (forward phase) works as a greedy search in decreasing order. Therefore, we can use the *score-and-search* based concept of Roure's TOCO and RSS heuristics (cf. algorithm 6) in our constraint based local search technique.

We assume the model as a set of CPCs and a target variable. Initially algorithm 5.2.4.1 starts with *initial model* M_0 containing only target variable T, corresponding

Algorithm 9 *Incremental $\overline{MMPC}$ Algorithm*

Procedure Forward Phase $\overline{MMPC}$(T, D, V, $\mathcal{O}_{\mathcal{OP}}$, $\mathcal{B}$):

Inputs: *target variable T; data D, set of variables in the network V, the search path $\mathcal{O}_{\mathcal{OP}}$ of the last learning step, the set $\mathcal{B}$; previously found best arguments (variables) for target variable and K is the number of variables to be store in $\mathcal{B}$.*

Output: *a set of Candidate Parent and Children (CPC) of T, the search path $\mathcal{O}_{\mathcal{OP}}$ and a set $\mathcal{B}$*

$i = 0$ and $CPC = \varnothing$
Model $M_{ini} = CPC \cup T$

**** *Forward Phase:* *****

if $\mathcal{O}_{\mathcal{OP}} \neq \varnothing$ **then**

Verify previous search path *over incoming data D. M_{ini} be a model for target variable where the new data agrees with previously learned search path.*
$M_{ini} = ((...((M_0 \cup A_1) \cup A_2)...) \cup A_j)$
where
$A_j = \arg\max_{X \in B} \left(\min_{S \subseteq CPC} MI\left(X; T \mid S\right)\right)$
if $assocA = \max_{X \in B} \left(\min_{S \subseteq CPC} MI\left(X; T \mid S\right)\right) \neq 0$

Update $\mathcal{B}$ accordingly.

Endif

$M_i = M_{ini}$

Repeat

$i++$
$M_i = (M_{i-1} \cup A_i)$
where
$A_i = \arg\max_{X \in V \setminus T} \left(\min_{S \subseteq CPC} MI\left(X; T \mid S\right)\right)$
if $assocA = \max_{X \in V \setminus T} \left(\min_{S \subseteq CPC} MI\left(X; T \mid S\right)\right) \neq 0$
Store top K variables in $\mathcal{B}_i$

Until $M_i \neq M_{i-1}$

**** *Backward Phase:* *****

For each variable $Y \in CPC$

if $\exists S \subseteq CPC \,\&\, MI(Y; T \mid S) = 0$ **then**

$CPC = CPC \backslash Y$

EndIf

EndFor

Return CPC, $\mathcal{O}_{\mathcal{OP}}$, $\mathcal{B}$

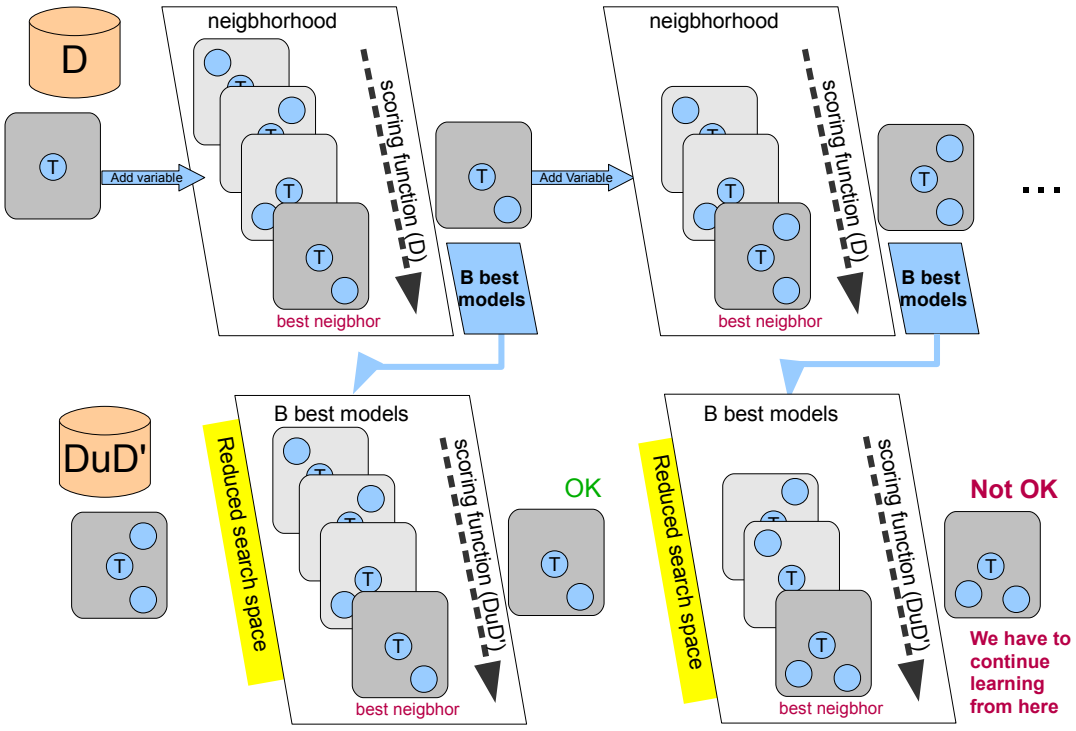


Figure 5.2: *Outline of $\overline{iMMPC}(T)$ algorithm (forward phase) for incremental data sets D and DuD'*

to an empty set of CPC for T. Then sequentially it adds the variables in it, having strong direct dependencies using the *MaxMinHeuristic* of $\overline{MMPC}(T)$ algorithm.

The algorithm proceeds as a search procedure to add the best variable in the model from its neighborhood search space. This iterative process continues until there is no more variable can be added. Therefore, all remaining variables are found with weak dependencies or independent of the target variable T given any subset of *CPC*. So, the *search path*, $\mathcal{O}_{\mathcal{OP}}$, in *iMMPC*$(T)$ is the sequence of variables (CPCs) added in the model M. At each step of the search path, we keep top K variables having the score value i.e. *MaxMinHeuristic*, closer to the variable that is added in the model and this set is know as $\mathcal{B}$. We maintain this set as a descending order.

On the arrival of new data, the forward phase of $\overline{iMMPC}(T)$ (cf. figure 5.2), first checks whether previously added variables in the list of *CPC* of a target *T* are valid for new available data $D \cup D'$. It also verify the previously learned *CPC* search path to define the initial model M_0 from which the search will be resumed. It continues to improve this model to introduce new variables in *CPC* in the final model. Meanwhile, variables found independent are deleted from the set of variables

V to avoid un-necessary computations and save the running time. The forward phase continues until the current model found as a final model and there is no more possible improvement.

In its backward phase, $\overline{iMMPC}(T)$ removes the false positive variables from CPC that may be considered in the first phase of the algorithm. For this purpose it tests for conditional independence of each variable X in CPC with target T given a subset S of CPC. If X and T are found independent given S then it removes X from the CPC.

5.2.4.3 An illustrative example

Now, we provide a simple example for the better understanding of incremental $\overline{MMPC}$(T) algorithm. The skeleton of underlying model is shown in figure 5.3. The data fed to the algorithm is sampled from the distribution of the original graph and we are interested in identifying set of CPC of variable $X6$, $CPC(X6)$.

Here, we present two cases corresponding to the two columns in figure 5.3. In the first one we handle data D and the other is to handle data $D \cup D'$. First case will start by searching the whole neighborhood space and storing the reduced search space (with $K = 4$) as a set $\mathcal{B}$. In second case the incremental algorithm will reduce the search space for new data by using only reduced search space $\mathcal{B}$.

In its first iteration, initial model M_0 contains only target variable and an empty set of *CPC*. It generates the neighborhood by considering all variables in V one by one, where $V = \{X1, X2, X3, X4, X5, X7, X8\}$. Then it calculates association ($minAssoc()$) for each model in the search space (between $X6$ and each of the seven other variables). Suppose the maximum value (Max of $minAssoc()$) is obtained for $X4$ variable, and $minAssoc(X6, X2) < minAssoc(X6, X7) < minAssoc(X6, X3) < minAssoc(X6, X4)$, so $X4$ is added in the $CPC(T)$. At this step we store the four best variables which have the association value closest to the maximum one so, for instance set $\mathcal{B}_0 = \{X4, X3, X7, X2\}$.

In second iteration, it starts from model M_1 containing target variable $X6$ and $X4$ that is added in CPC during previous iteration and generates its neighborhood by

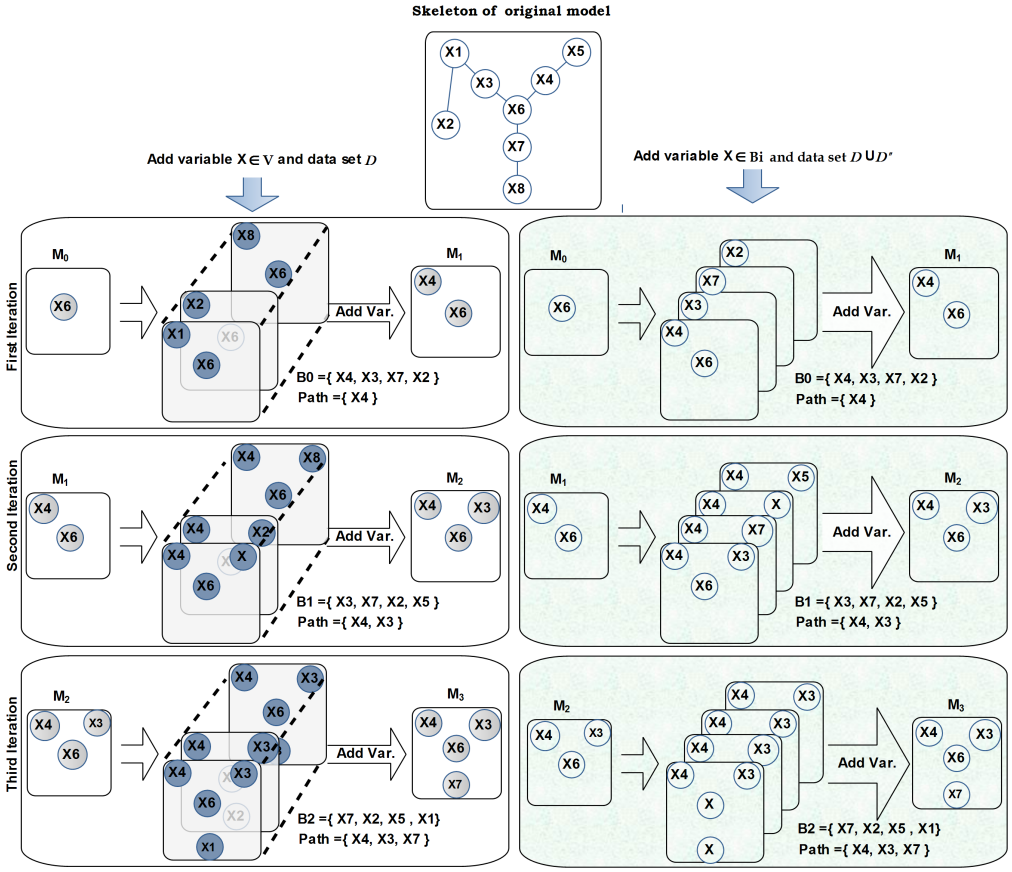


Figure 5.3: *An illustrative example for* $\overline{iMMPC}(T)$ *(forward phase) algorithm*

adding all remaining variables one by one (excluding variables found independent in previous iteration). Then, it computes $minAssoc()$ for each model, but in this iteration it choose the minimum association (MI) between conditional and unconditional variables for each model e.g. for model $\{X6, X4, X1\}$ contains a target variable $X6$, CPC variable $X4$ and a testing variable $X1$, it will compute $MI(X6, X1)$ and $MI(X6, X1|X4)$, the minimum value between these two becomes the score ($minAssoc()$) of this model. It repeats this process for all models in the search space. Finally, it selects the model having maximum score ($MaxMinHeuristic$) and adds the testing variable in the set of CPCs. For instance in figure 5.3 it selects $X3$ in its second iteration. Again it saves the search path (the sequence of variables added in CPC), which is $Path = \{X4, X3\}$ and the top four variables for this iteration e.g. $\mathcal{B}_1 = \{X3, X7, X2, X5\}$.

Similarly, in third iteration where $X7$ is added in the list of CPCs, also the set of

top four best variables/models is stored as $\mathcal{B}2$.

Further we can not proceed for this example because we assume the remaining variables have a zero association with the target variable given any subset of CPC.

On the arrival of new data D', our algorithm will relearn the model for data $D \cup D'$ using stored search paths and best neighborhoods. In this example we are storing top four variables therefore, for new data set, search space will be reduced to only four models.

So, again it starts from the initial model and verify the previous learning path in the light of set B for each iteration consequently search space is reduced. Again supposed the variable *X4* has a maximum association value so, it added in the list of *CPC*(T). We are also verifying the search path, if it is not in the same sequence then we need to recalculate the set of best variables/models.

This simple example illustrates the interest of our incremental approach. In high dimensional domain by using $K \ll n$ we can save a lot of *Assoc* computations.

5.2.5 Complexity

The complexity of *iMMPC* algorithm is based on the number of conditional independence calls. For first window of the incremental learning process, the complexity of the first phase remains same as its batch *MMPC* algorithm, where $\overline{MMPC}(T)$ performs conditional independence tests for target variable with all remaining variables, conditioning all subsets of CPC. So, for first phase of $\overline{MMPC}(T)$ the complexity is given as $O(n \times 2^{|CPC|})$. In the second phase, it performs tests for independence of any variable in the CPC with the target, conditioned on all subsets of the rest of the variables in the CPC. Therefore the computational complexity for second phase is $O(|CPC| \times 2^{|CPC|-1})$. If PC is the largest set of parents and children and maximum number of conditional variables are c and then the overall complexity to build the whole skeleton of the Bayesian network is $O(n^2 \times |PC|^{(c+1)})$ [138].

For succeeding windows, the first phase of $\overline{iMMPC}(T)$ is reduced by considering only K best variables. Therefore it is bounded by $O(K \times 2^{|CPC|})$. In the best case, to build the whole skeleton it requires $O(nK \times 2^{|CPC|})$ tests. In addition the

number of calls for the second phase of the $\overline{iMMPC}(T)$ algorithm.

In the worst case, the new data can change the learning path obtained in previous window, therefore, the complexity is considered as batch plus K, $O((n + K) \times 2^{|CPC|})$.

Now, we compute the complexity of the example presented in section 5.2.4.3 on page 97, for a single target variable. If we consider the best case presented in the example, then at first time forward phase needs $3(n - 2)$ comparisons and for next incremental stage it requires only $3K$ comparisons. In average case if new data changes then the complexity will be $O(2K + n)$. For worst case when distribution of the entire data is changed or model learning process starts from scratch then it needs maximum comparisons $O(3n)$.

5.3 Incremental Max-Min Hill-Climbing

5.3.1 Principle

In incremental Max-Min Hill Climbing ($iMMHC$) algorithm, we address the problem of Bayesian network structure learning from data stream, especially in high dimensional domains. As discussed in chapter 3, the learning of Bayesian network structure from a batch dataset is a *NP-hard* problem, therefore, if we take into account the challenges of data stream (cf. section 2.4.1, page 24) it becomes a more difficult task. There are few approaches addressing incremental Bayesian network structure learning problem as discussed in chapter 4. The recent work that claims to be outperformed over the state of the art is Roure's [121], which is based on *score-and-search* based structure learning technique.

The hybrid methods for Bayesian network structure learning are scalable for high dimensional domains such as *MMHC* (cf. section 3.5, page 53). Therefore, we propose a new approach $iMMHC$ that merges the idea of incremental greedy/hill-climbing search and of *MMHC* to deal with this problem.

5.3.2 Objective

The main idea of *incremental MMHC* ($iMMHC$) is to incrementally learn a high quality BN structure by reducing the learning time. It can be achieved by re-using the previous knowledge and reducing the search space. Like $MMHC$ method, $iMMHC$ is also a two phase hybrid algorithm as described in algorithm 10 and illustrated in figure 5.4. Both phases are incremental in their nature and handle data stream over landmark window.

5.3.3 Incremental local search

This first phase discovers the possible skeleton (undirected graph) $\mathcal{G}$ of the network for a given window, by using *iMMPC* method (cf. section 5.2.3). It learns a local structure around each variable by using the previous knowledge, and avoids exploring those parts of the search space which were previously found with low quality. For this purpose, it maintains a list of search paths (*order of the variables included in the set of candidate parent-children, CPC*). At the same time, it stores a set of top *K* best variables that have more chance to be considered in CPC. The minimum size of set *K* depends upon the average degree of the graph. When new data arrives, $iMMPC$ checks every search path. If they are yet validated, there is no need to re-learn the CPCs (local models). On the contrary, it triggers the re-learning process. The detail of the algorithm and the description of the parameters are defined in the previous section. Hence, we build the local structures incrementally by shrinking the search space, then we build an undirected graph $\mathcal{G}$ (skeleton) by merging these local structures.

5.3.4 Incremental global optimization

In the second phase, a greedy search is initiated for global optimization. A naive application of $MMHC$ would start this greedy search from an empty graph. Here, we propose an incremental optimization phase by initializing the greedy search procedure by the graph that is obtained in previous time window (like Shi

Algorithm 10 $iMMHC(D_w)$

Require: Data of landmark window w_i (D_{w_i}), previous top K best neighbors for each variable ($\mathcal{B}$), previous BN structure ($\mathcal{BN}_{i-1}$)
Output: a Bayesian network $\mathcal{BN}_i$

** ***Incremental local identification*** **\

$\mathcal{G} = iMMPC(D_{w_i}, \mathcal{B})$
Endfor

** ***Incremental greedy search*** **\

Starting from model M = empty graph **or** $\mathcal{BN}_{i-1}$.
Only try operators *add_edge* ($Y \rightarrow X$)
if there exist an edge $X - Y$ in $\mathcal{G}$
(no constraint for *remove_edge* or *reverse_edge operators*)

Update parameters θ for model M.

Return $\mathcal{BN}_i$

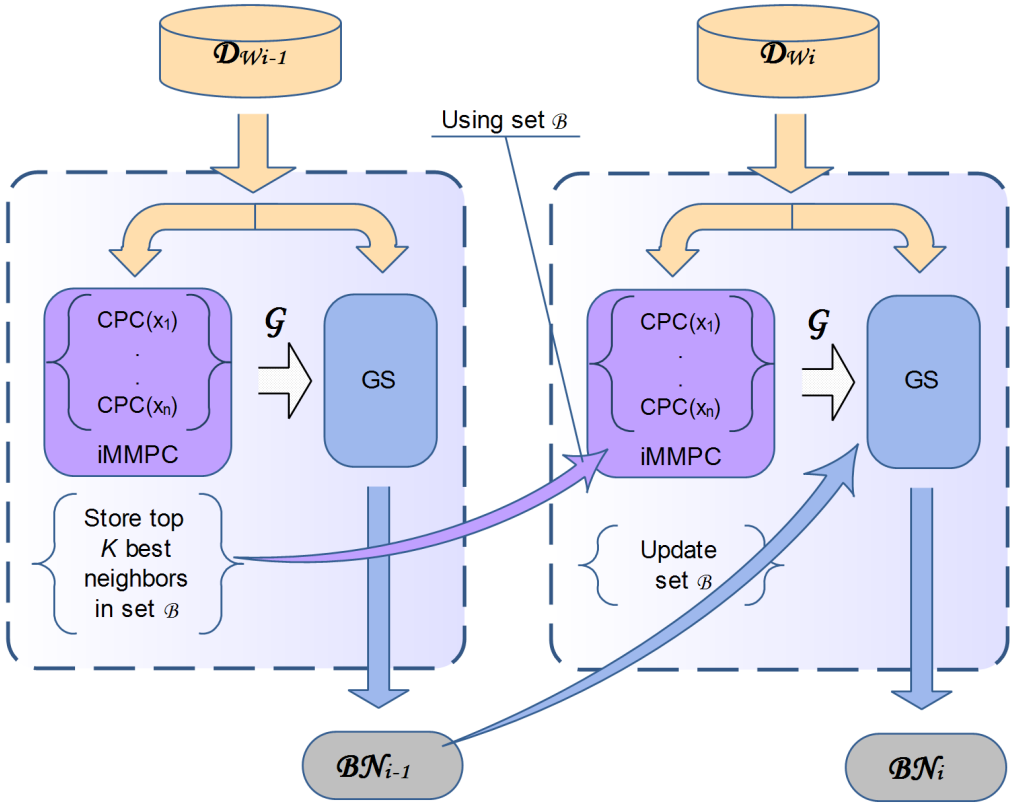


Figure 5.4: *iMMHC outline : dependencies between iMMHC execution for window w_i and previous results.*

and Tan, section 4.2.2). The greedy search considers adding those edges that are newly introduced in skeleton $\mathcal{G}$, but also removing the outdated edges. We apply the

operators *add_edge*, *remove_edge* and *reverse_edge*, where *add_edge* can add an edge $Y \rightarrow X$ if and only if Y is a neighbor of X in $\mathcal{G}$. By this way, $iMMHC$ keeps the sense of incremental learning by considering the previously learned model and revising the existing structure in the light of new data, as summarized in figure 5.4.

5.3.5 Comparison with existing methods

***iMMHC* vs *TSearch*:** $iMMHC$ is a two phase hybrid algorithm as $TSearch$ (cf. section 4.2.2, page 76) but both phases of $iMMHC$ are incremental while only the second phase of $TSearch$ deal with this problem. Another significant factor is the robustness of the first phase of the $iMMHC$, by using $iMMPC$, better than a simple tree approximation.

***iMMHC* vs *iHCS*:** In contrast to $iHCS$ (cf. section 4.2.1.4, page 69), $iMMHC$'s greedy search phase has two constraints, a previous graph and a CPC skeleton $\mathcal{G}$, therefore, it has already very limited search space. These constraints increase the accuracy of classic greedy search as well as reduce its search space to a great extent.

5.3.6 Optimizing frequency counting

The computational efficiency of Bayesian network structure learning algorithms is based on the number of tests required to measure the association between variables [132]. These tests could be conditional independence tests or association computations of any score function. In these tests algorithm has to read from an external source of data, may be a data file or database, and compute the frequencies for the given variables (contingency tables). Therefore, the most time consuming task of learning algorithms is to compute the frequencies. In contrast to the batch learning environments, in incremental or online learning, the running time is more important, where the model has to be updated in limited available time.

There are a number of optimization techniques proposed in the article of *MMHC* [138]. These techniques improve the efficiency of the local search phase of the algorithm. While, in *MMHC*, the calls for local scores in global optimization

(second) phase, are higher than the calls for independence test. For example, the authors created a tiled version of *Alarm* network with approximately 5000 variables and 6845 edges and sampled 5000 instances from its distribution. Then, MMHC reconstructed this tiled network in approximately 13 days total time. The local discovery phase took approximately 19 hours and the rest was spent on global optimization (edge orientation) [138]. Therefore, to speed up the second phase too, we introduce *frequency counter* a middle ground between the dataset and the algorithm because:

> *A large fraction of the learning time involves collecting the sufficient statistics from the data [49].*

Frequency counter is an intermediate layer between the dataset and the both phases of the $iMMHC$ (cf. fig. 5.5). It is a hash-table data structure maintained in the temporary memory as a cache.

$iMMHC$ has two phases, in its first phase it computes conditional independence tests like: $MI(X, Y|\mathbf{Z})$, where $\mathbf{Z}$ is the subset of already found CPC's and Y is the variable being tested for CPC. For each test it computes the contingency tables of N_{ijk} to find the ***MI***. Here we populate these frequencies in a hash-table to share with future calls. In hash-table the basic operations (lookup, insertion, deletion) take $O(1)$ time, with a fairly small constant (one hash calculation plus one pointer lookup). This makes hash tables very fast in many typical cases.

In the next phase of edge orientation, greedy search computes the local scores (cf. section 3.4.2.1, page 44). For local scores it re-computes the N_{ijk} values. Therefore, it will share the same frequency counter. First of all it will search into the cache, if it finds the frequencies for a required set of variables then returns it, otherwise it computes from the dataset. At the same time it populates these values in the cache.

By this approach we can save a significant amount of time, as a lot of frequency counter calls N_{ijk} are shared with both phases and within the phases too.

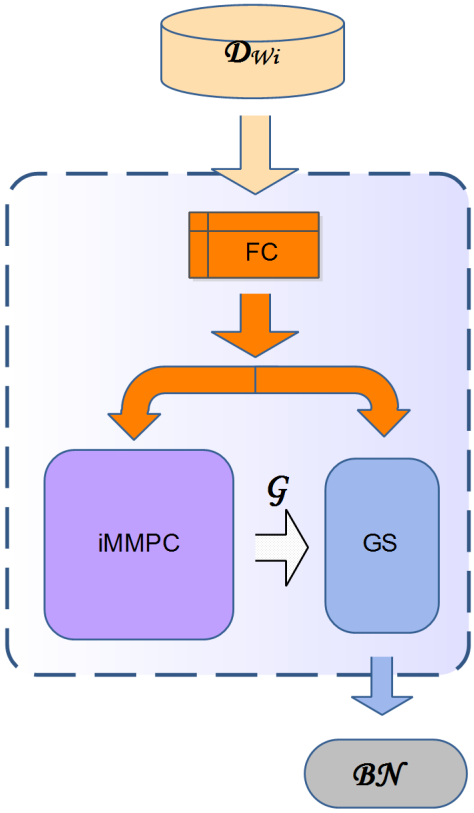


Figure 5.5: *Frequency counter "FC" outline in $iMMHC$ algorithm*

5.3.7 Complexity analysis

The first phase of the $iMMHC$ has the same complexity for each variable as *iMMPC* algorithm. Where it computes the set of CPCs for all variables. While the second phase uses a greedy search approach, where it starts from an empty graph and try to apply the operators *add_edge*, *reverse_edge* and *remove_edge* to find an optimal graph. So, it generates a search space of neighborhood structures by applying possible local changes. Let n be the number of variables in the current DAG and d is the maximum number of parents per node, the number of local changes are bounded by $O(n \times d^2)$, whereas it is bounded by $O(n^2)$ for *add_edge* operator and $O(n \times d)$ for *remove_edge* and *reverse_edge* operators.

Whereas, in succeeding windows of incremental process greedy search is initialized with previous graph, which is obtained in the last window. Therefore, it only tries for *add_edge* operator for those variables that are newly introduced in the sets of CPCs, rather than testing *add_edge* operator for all variable in CPC. If there are

average j unchanged variables in all set of CPCs, then the complexity for *add_edge* operator will be reduced to $O((n-j)^2)$. Moreover, by caching local score values and using frequency counter layer, it saves considerable amount of time to explore the search space.

5.4 Adaptation of iMMHC for sliding window

5.4.1 Principle

As discussed in chapter 2, in data stream environment, it is not feasible to store whole data or wait for data completion for mining tasks. There are different techniques used to store the summary of data seen so far. Buntine [15] stores sufficient statistics in a parent lattice structure (structure of alternative parents) and similarly Lam and Bacchus [92] consider the current network as a summary of the previous data. They use *maximum a-posteriori probability* (MAP) and consider the data instances being summarized are distributed according to the probability measure described by the model [48]. Friedman and Goldszmidt store set of candidate networks and their sufficient statistics in temporary memory. While Roure [121] stores all data counts in an adaptation of AD-tree data structure [123].

AD-tree [104] and its recent adaptations [123, 6, 83, 140] are the data structures that store counts for all possible combinations of the variables, therefore, they can serve for multiple data mining tasks. These structures are not cheap to update incrementally because each new record can match up to 2^n nodes in the tree in the worst case [104]. Therefore, it is more complicated for high dimensional data.

Here, we propose a middle ground between the two extreme approaches; *AD-tree* and *MAP*. So, we do not completely rely on MAP, neither we store counts for all possible combinations of variables like AD-tree.

5.4.2 Objective

The objective is to store only minimum possible sufficient statistics from the previous data, rather than storing the whole data stream. As compared to batch learning environments, we expect an approximate results from these summaries of old data [56].

5.4.3 Update of sufficient statistics

We have already discussed the adaptation of frequency counter in $iMMHC$ algorithm, in section 5.3.6. It was for landmark window, where whole previous data is available for learning algorithm and frequency counter serves as a cache to speed up the learning process. While in sliding window environment, it requires to store summary of old data as a sufficient statistic. Therefore, for this purpose, we use current frequency counter FC_i and the current model BN_i as a summary of previous data D_i. The frequency counter FC_i contains all N_{ijk} that are used to produce model at i^{th} window. For next window, w_{i+1}, this frequency counter FC_i will be served as FC_{old} (cf. figure 5.6).

$$N_{ijk_{(D)}} = N_{ijk_{(D_{w_i})}} + N_{ijk_{(D_{w_{i-1}})}} \tag{5.1}$$

where

$$N_{ijk_{(D_{w_{i-1}})}} = N_{ijk_{(FC_{old})}} \vee (N \times N_{ijk_{(MAP_{M_{i-1}})}}) \tag{5.2}$$

The frequency counts from the data D will be the sum of frequencies from current data, D_{w_i}, and the frequencies from previous data (seen so far), $D_{w_{i-1}}$. Therefore, in equation 5.1, to compute frequencies from whole dataset we require $N_{ijk_{(D_{w_i})}}$ that can be computed directly from current data. While for $N_{ijk_{(D_{w_{i-1}})}}$, most of counts will be found in FC_{old} (if the distribution of data is not changed and the new model will not be too much different from the old ones). The rest can be approximated by querying a joint distribution of the previous model, $MAP_{M_{i-1}}$ (cf. section 3.3.4, page 40) that is obtained in the previous learning step and multiplying it by the

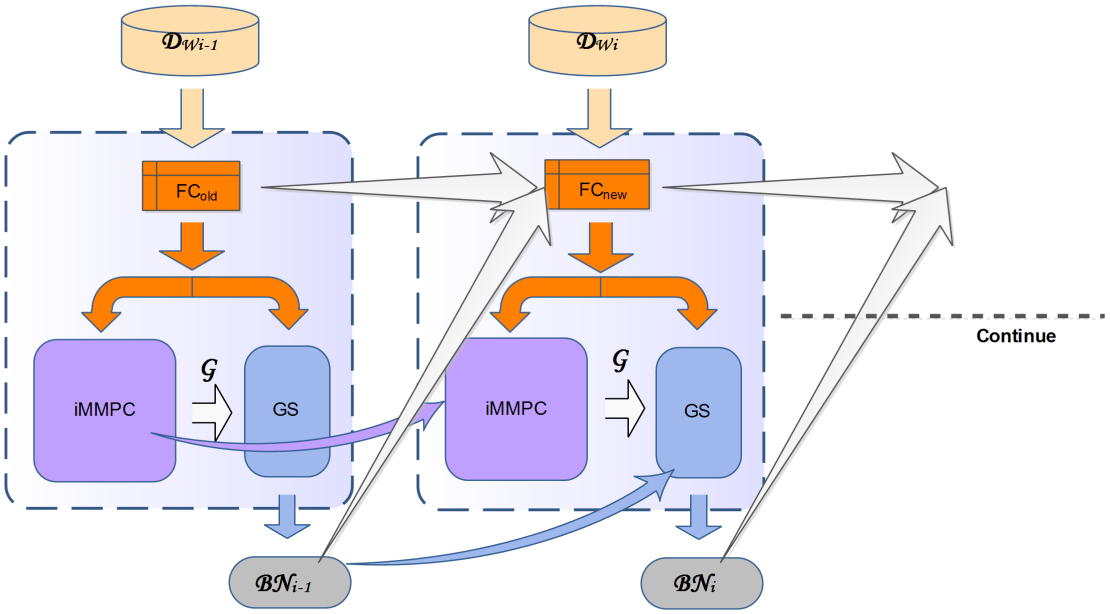


Figure 5.6: *Using an old frequency counter "FC" and BN as a summary of old data for each window "w".*

total number of examples in $D_{w_{i-1}}$, $(N \times N_{ijk_{(MAP_{M_{i-1}})}})$. In the final analysis, our approach do not completely rely on the MAP model for sufficient statistics and give a trade-off between two extremes.

5.4.4 Complexity

In this section we analyze how it is complex to update sufficient statistics. In equation 5.1 the complexity to find a particular frequency N_{ijk} depends upon two parts. In first part it reads all records in the dataset to find the frequencies for a specific combination of the variables. So, the complexity for one N_{ijk} computation is $O(1)$.

For second part, there are two situations, the best case is that the frequency is found in the FC cache, i.e. it performs $O(1)$ comparison. In its worst case if FC does not contain this value then it will be computed from Bayesian network by asking a probabilistic query. As we discussed in section 3.3.4 about different methods for inference in Bayesian network, the exact inference methods are not feasible in high dimensional domains if the structure is too complex (complexity exponential with respect to the size of the biggest clique). So the complexity of worst case totally

depends upon which approximate inference method is used. The stochastic sampling algorithms are anytime algorithms. The complexity of these algorithms depends upon the number of the samples. If there are d maximum number of parents for a variable so, the maximum number of variables in a query are $d + 1$. Therefore the complexity is $O(d)$.

5.5 Adaptation of iMMHC for damped window

5.5.1 Principle

In an incremental or online data mining, in other words data stream mining, the methods continuously revise and refine their models in order to accommodate new data as it arrives. To keep model up to date with new concepts, it is also required to eliminate the effect of data examples representing outdated concept. This issue has not been addressed by the researchers in Bayesian network structure learning community, except Nielsen and Nielsen [109] where they test if newly arrived example fit to the existing model or not.

5.5.2 Objective

Forgetting outdated data by adding a fading factor is a commonly used approach in data stream mining to adapt non-stationary data. So that the recent data got more importance than old one (cf. section 2.3.2.1, page 20).

5.5.3 Forgetting mechanism

In this section we describe a forgetting mechanism to emphasizes the importance of recent data. As in conceptual clustering, they are using aging methods to step-by-step forgetting the old concepts by decreasing the relative weight of samples, for example the FAVORIT system [85]. Also in incremental collaborative filtering [143], they are using fading factor at each new session so that older sessions become less important. For this purpose, we also introduce a fading factor α at frequency counter

so, all calls to the FC are weighted by a fading factor such as:

Definition 30. *(**Forgetting mechanism**) Let $N_{ijk_{(D_{w_i})}}$ be the frequency counts from data in current window w_i, and α is a fading factor such that $0 < \alpha < 1$. Then the frequency count, $N_{ijk_{(D)}}$, over the whole data stream can be obtained as:*

$$N_{ijk_{(D)}} = N_{ijk_{(D_{w_i})}} + \alpha[N_{ijk_{(D_{w_i-1})}} + \alpha[... + \alpha[N_{ijk_{(D_{w_0})}}]]]$$

$$= \sum_{w_i=CurrentWindow}^{0} N_{ijk_{(D_{w_i})}} \times \alpha^{w_i-1} \tag{5.3}$$

5.6 Conclusions

In a scenario where dataset is growing continuously and we cannot wait for its completion, the obtained model at a certain time will be outdated. So, revising the existing model by re-executing the batch learning algorithm will not only take a lot of resources as well as it is a costly venture. Therefore, in this chapter we proposed an incremental version of constraint based local search, Max-Min Parent-Children algorithm, $iMMPC$. It learns the skeleton of a Bayesian network. We applied idea of *score-and-search* based incremental greedy/hill-climbing approach to discover the set of candidate parent children (CPC) of a target variable *T*. It stores the most strong dependencies as a set of best arguments. Therefore, it reduces the search space for new data by considering only the strong dependencies. This approach improves the performance of the algorithm systematically and reducing the complexity significantly.

Furthermore, to learn a full Bayesian network structure we proposed incremental Max-Min Hill-Climbing $iMMHC$ algorithm, which can be scale up for high dimensional domains over landmark windows. The both phases of this hybrid approach, $iMMHC$, deal with data in an iterative way, by using (1) $iMMPC$ algorithm for incremental local structure identification and (2) a constrained greedy search for global model optimization starting from the previously obtained model. By this

way, we are able to take into account the previous knowledge in order to reduce the search space for incremental learning. As we have more and more data the test of independence gives more accurate values. We also introduce a frequency counter a middle layer between the dataset and the both phases of $iMMHC$. It gives a rapid access and re-usability to the sufficient statistics that are already computed. Finally, we proposed $iMMHC$ adaptation techniques over sliding and damped windows.

6

Experimentation

Contents

6.1 Introduction 114
6.2 Evaluation measures 114
6.2.1 Computational efficiency 114
6.2.2 Model accuracy 115
6.3 Experimental protocol 115
6.3.1 Benchmarks 115
6.3.2 Algorithms 116
6.4 Results and interpretations with landmark windows 117
6.4.1 Parameters analysis 117
6.4.2 iMMHC for incremental learning 120
6.4.3 Incremental versus batch MMHC 123
6.4.4 iMMHC for high dimensional domains 123
6.5 Results and interpretations with sliding window 124
6.6 Results and interpretations with damped window 126
6.6.1 Non-stationary domains: 126
6.6.2 Stationary domains: 128
6.7 Conclusions 128

6.1 Introduction

In this chapter, we carry out several experiments comparing $iMMHC$ with the most recent state of the art. Our goals are to evaluate its ability to deal with high dimensional data and characterizing the situation where it outperforms the other algorithms. In this chapter we present the results to show the flexibility of the algorithm and empirical evaluation of $iMMHC$ to justify its effectiveness.

The rest of the chapter is organized as follows. In section 6.2 we give a brief description of the evaluation criteria for the algorithms in our experiments and experimental protocols in section 6.3. In section 6.4 we empirically see, how *iMMHC* algorithm deals with landmark window and later for sliding window in section 6.5 and for damped window in section 6.6. Finally we conclude this chapter in section 6.7.

6.2 Evaluation measures

In this section we discuss two metrics that are used to evaluate our algorithm in terms of *computational efficiency* and *model accuracy*. The results presented in this chapter are the mean and standard deviation of each metric obtained over five datasets corresponding to a given benchmark model.

6.2.1 Computational efficiency

The main task in learning algorithms is to compute score functions. The complexity of the algorithm depends upon the number of total *score function calls* i.e. in our case, it is equal to the sum of the calls for MI independence tests and local score function calls during the greedy search. The total function calls are logarithmic scaled for better understanding. The confidence interval bars are computed as:

$$lower\,limit = \log_{10}(average(FunctionCalls) - \sigma(FunctionCalls)) \quad (6.1)$$

$$upper\,limit = \log_{10}(average(FunctionCalls) + \sigma(FunctionCalls)) \quad (6.2)$$

Where σ is a standard deviation.

6.2.2 Model accuracy

We used the Structural Hamming Distance (SHD) (cf. section 6.2) for *model accuracy*. It compares the distance between two CPDAGs. Therefore, first we converted the learned and original network structures into corresponding CPDAGs (cf. section 3.3.3.2). The interest of this measure is that it does not penalize an algorithm for structural differences that cannot be statistically distinguished. It computes the number of edges which are missing, extra or with wrong directions.

6.3 Experimental protocol

Here is an explanation of the benchmarks, algorithms and their parameters used in our experimentations.

6.3.1 Benchmarks

To evaluate our approach, we used following well-known networks from GeNIe/SMILE network repository [1]. These datasets are summarized in Table 6.1

Alarm: It is a network by medical experts for monitoring patients in intensive care.

Barley: This model is developed for the project "Production of beer from Danish malting barley grown without the use of pesticides".

Hailfinder: It is a model to forecast severe summer hail in northeastern Colorado.

Pathfinder: It is an expert system network that assists surgical pathologists with the diagnosis of lymph-node diseases.

Link: It is a model for the linkage among two genes to measure the distance between them.

To test the performance of our algorithm in high dimensional domain, we also generated a network with about one thousand of variables by using BN tiling method

1. http://genie.sis.pitt.edu/networks.html

Benchmark	#Vars	#Edges	Cardinality	#Instances	Av. Deg.	Max. Deg.
Alarm	37	46	2-4	20,000	2.5	6
Barley	48	84	2-67	20,000	3.5	8
Hailfinder	56	66	2-11	20,000	2.4	17
Pathfinder	109	195	2-63	20,000	1.7	106
Link	724	1125	2-4	10,000	3.1	17
Alarm28	1036	1582	2-4	10,000	3	8
Gene	801	972	3-5	5,000	2.4	11

Table 6.1: *Name of the benchmark, number of variables and edges, cardinality and degree of the each graph used for our experiments.*

of Tsamardinos et al [139] (implemented in Causal Explorer [2]). For this purpose, 28 copies of *Alarm* network are tiled, so that obtained network *Alarm28* containing 1036 variables. Subsequently, we sampled five datasets from all these Bayesian networks using GeNIE software and for *Alarm28* using Causal explorer. We also chose five *Gene* datasets from $MMHC$ source website [3].

The numerical characteristics of the networks and datasets are summarized in Table 6.1. We create an incremental scenario by feeding data to the algorithms with window sizes $\Delta_w = 1000$, 2000 and 3000. In this study we are presenting our results for windows size 2000 except for Gene data set where we used window size 1000.

6.3.2 Algorithms

We tested *iMMHC* algorithm in two different scenarios, as proposed in section 5.3:

1. $iMMHC_{\varnothing}$: starting the greedy search phase from an empty graph.
2. $iMMHC_{G}$: starting the greedy search phase from the previously obtained graph.

We compared our algorithm with batch *MMHC* and with the incremental $TSearch$ algorithm as described in section 4.2.2. We implemented the original algorithms as described in their articles, using **C++** language. We used **Boost graph** [4] library which provides an environment to manipulate graphs. We also used Bayesian library

2. http://www.dsl-lab.org/causal_explorer/
3. http://www.dsl-lag.org/supplements/mmhc_paper/mmhc_index.html
4. http://www.boost.org/

ProBT [5]. It provides Bayesian network related functionalities. The Bayesian network *StrucutreLearning* package is developed by our team as an extra layer for ProBT. It provides the functionalities to manipulate Bayesian networks and to learn their structures using different algorithms.

To keep the harmony between these algorithms, we used the same heuristic for all of them therefore $MMHC$ and $iMMHC$ are also using greedy search instead of the original TABU search. In greedy search, we used the BIC score function and we are caching these score values to avoid re-computational overheads for similar score function calls. Independence is measured with the help of *Mutual Information* (MI). The maximum number of conditioning variables are fixed to 10. The confidence level, α has traditionally been set to either 0.01 or 0.05 in the literature on constraint-based learning, so we have decided to adjust α to 0.05 in our experiments.

Experiments were carried out on a dedicated PC with Intel(R) Xeon(R) W3565 3.20 GHz CPU, 64 bits architecture, 8 GB. RAM memory and under Windows 7.

6.4 Results and interpretations with landmark windows

In this section, experiments are conducted over landmark window environment. As described in section 2.3.2.1, in this scenario dataset is growing continuously and we can not wait for data completion. Therefore, in this incremental environment the whole previous data seen so far is available for learning algorithms.

6.4.1 Parameters analysis

Here we analyze the behavior of user defined parameter K and window size w. We also discuss the effect of different initializations for greedy search phase of *iMMHC* algorithm. First of all we analyze the effectiveness of frequency counter cache. It is used in all the other experiments presented in this study.

5. http://www.probayes.com/index.php

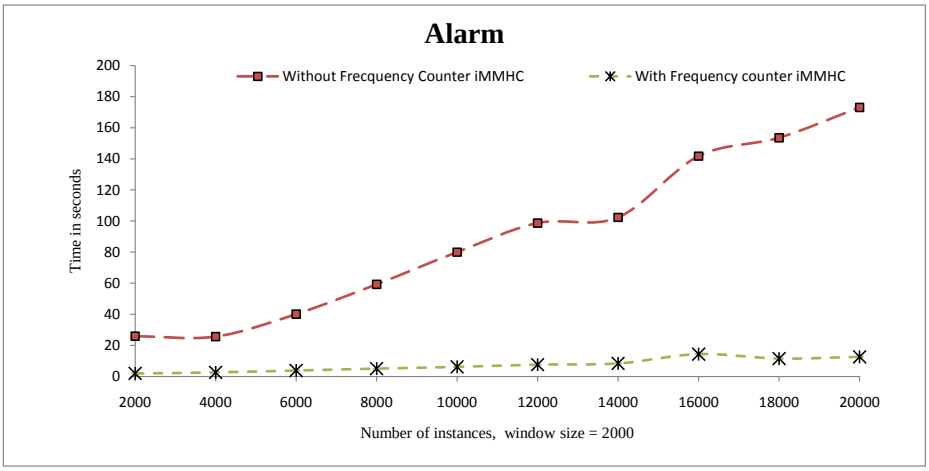


Figure 6.1: *Comparison of $iMMHC$ execution time with and without frequency counter for Alarm on landmark window.*

Influence of frequency counter cache: Here, we show how frequency counter cache proposed in section 5.3.6, reduces the learning time over incremental process. For this purpose, we executed $iMMHC_{\varnothing}$ with and without frequency counter (FC) cache. Figure 6.1 compares the execution time in seconds for Alarm network. We can observe that without frequency counter the execution time increases linearly on landmark window. While using the frequency counter, we saves a lot of execution time and it remains almost constant.

Parameter *K*: K is a user-defined parameter to specify the number of potential parent children that are to be stored in the memory at each step of $iMMPC$. So, it caches top *K*, most associated variables with the target variable to later use with novel data. Because they have more chances to become parent or children of a target variable.

The behavior of $iMMHC$ with different *K* values is shown in figure 6.2. We can see that the value of K could be between average and maximum degree of the theoretical graph. Logically, the complexity linearly increases with respect to K. This increase in the complexity is negligible as compared to the total function calls, whereas it is worth noting that $iMMHC$ learned a good model even with low value of K. Another reason is that we are considering landmark window and the data has been sampled from stationary domain, therefore the algorithm do not encounter drift.

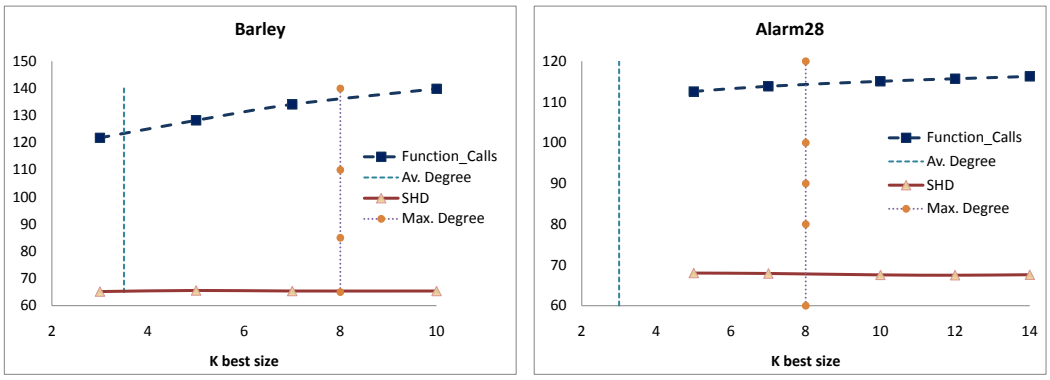


Figure 6.2: *Performance of $iMMHC$ algorithm for different K values (Function calls are divided by 10^2 for "Barley" and 10^5 for "Alarm28", and SHD by 10^1 for "Alarm28" for ease of exposition)*

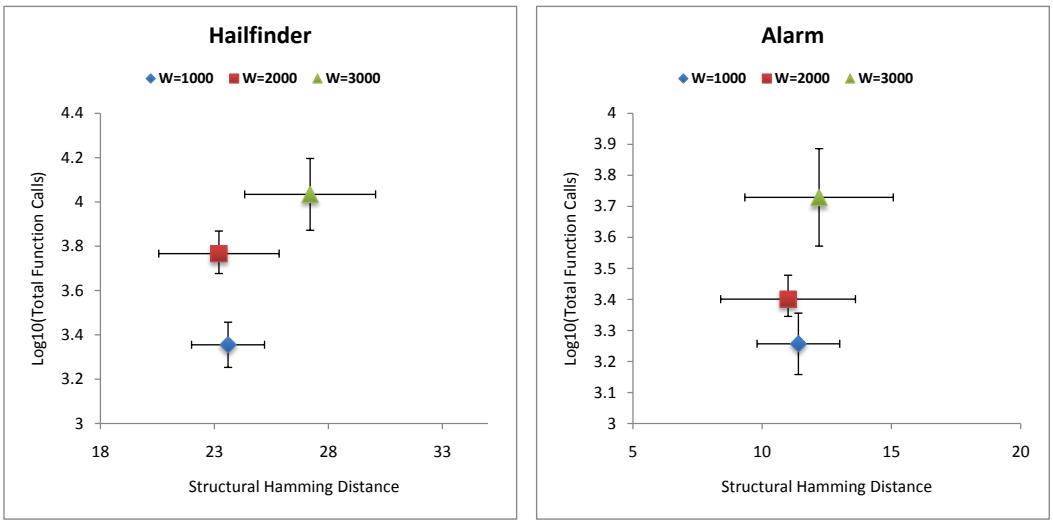


Figure 6.3: *Performance of $iMMHC$ algorithm for windows size 1000, 2000 and 3000 (average and standard deviation for final models)*

Window size *w*: Mostly, algorithms in incremental Bayesian network structure learning (cf. chapter 4) store incoming data in a temporary memory and wait until sufficient amount of data is available to trigger the re-learning process. Figure 6.3 shows the role of different window sizes on incremental learning process of $iMMHC$. We can observe that if window size is too large as 3000 then the algorithm will encounter a lot of changes in the local structures, so that it requires more function calls as in small window sizes. Correspondingly, it also affects the accuracy of the model. Therefore, the window should have enough amount of data, so that the algorithm can find a reliable set of K variables, having a small window size can negatively effect the learning process. We found that the ideal window size for the

benchmarks used here is about 2000.

Initialization of greedy search: From figure 6.4 we can observe the overall accuracy obtained by $iMMHC_{\varnothing}$ is very similar to the batch $MMHC$ algorithm except for datasets having high cardinality like *Barley* and *Hailfinder*. On the other hand $iMMHC_G$ got better accuracy than $iMMHC_{\varnothing}$ except for *Barely* and *Pathfinder* where batch $MMHC$ and $iMMHC_{\varnothing}$ learned a better model.

If we compare these two initializations with respect to the complexity then $iMMHC_G$ always has less function calls than $iMMHC_{\varnothing}$. As a conclusion we can say that initialization of greedy search with previously obtained graph is a tradeoff between the accuracy and complexity.

6.4.2 iMMHC for incremental learning

Figures 6.4 and 6.5 show that $iMMHC_G$ outperforms $TSearch$ with respect to complexity and accuracy. However $TSearch$ has a better accuracy than $iMMHC_{\varnothing}$ and batch $MMHC$ in *Hailfinder* and it has overall low complexity than $iMMHC_{\varnothing}$ and batch $MMHC$ except in *Pathfinder* and *Link*.

We observed that during $TSearch$ learning, the skeleton discovery phase ($MWST$) introduces lot of false-positive edges. Consequently, these errors propagate to the final structure. Since this structure is also used as an input for the next time window, the local errors in skeleton discovery mislead the incremental structure learning process. As another consequence of false-positive edges, the complexity of the global optimization phase is also increased.

The robustness of $iMMHC$ can be explained by the fact that it uses an incremental adaptation of $MMPC$ algorithm for skeleton discovery phase which has been proven as a robust solution, limiting the number of false-positives.

It is difficult to compare with Roure's results for $iHCS$ (cf. section 4.2.1.4) as we are not using the same experimental protocols and evaluation criteria. Roure compared the accuracy of his algorithm by the ratio of score obtained by incremental learning process and corresponding batch one (for final models), i.e. $\frac{score_{incremental}}{score_{batch}}$.

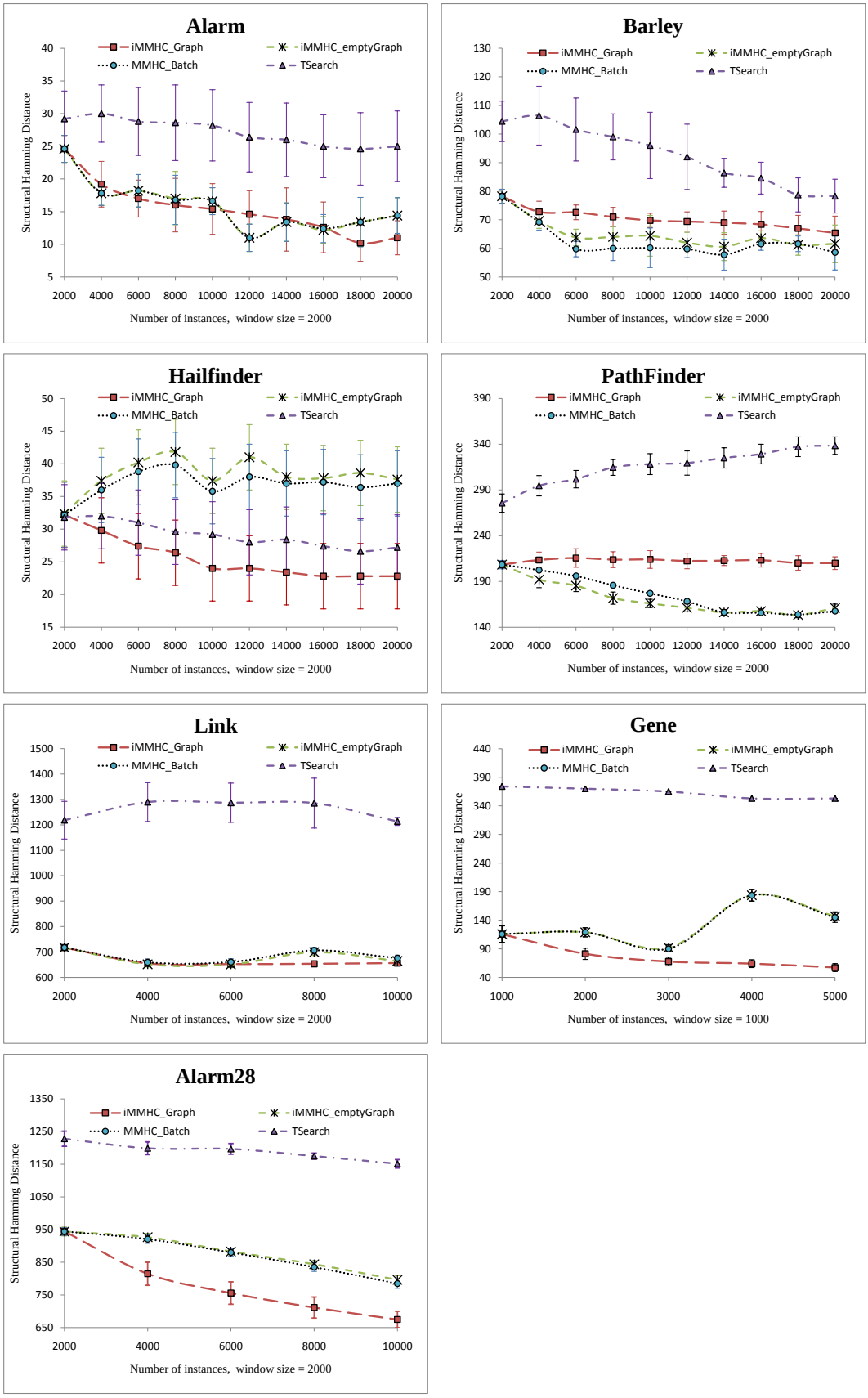


Figure 6.4: *Comparisons of Structural Hamming distance (SHD) for* $MMHC$, $iMMHC_{\emptyset}$, $iMMHC_{G}$ *and* $TSearch$ *algorithms using window of size 2000 (1000 for "Gene")*

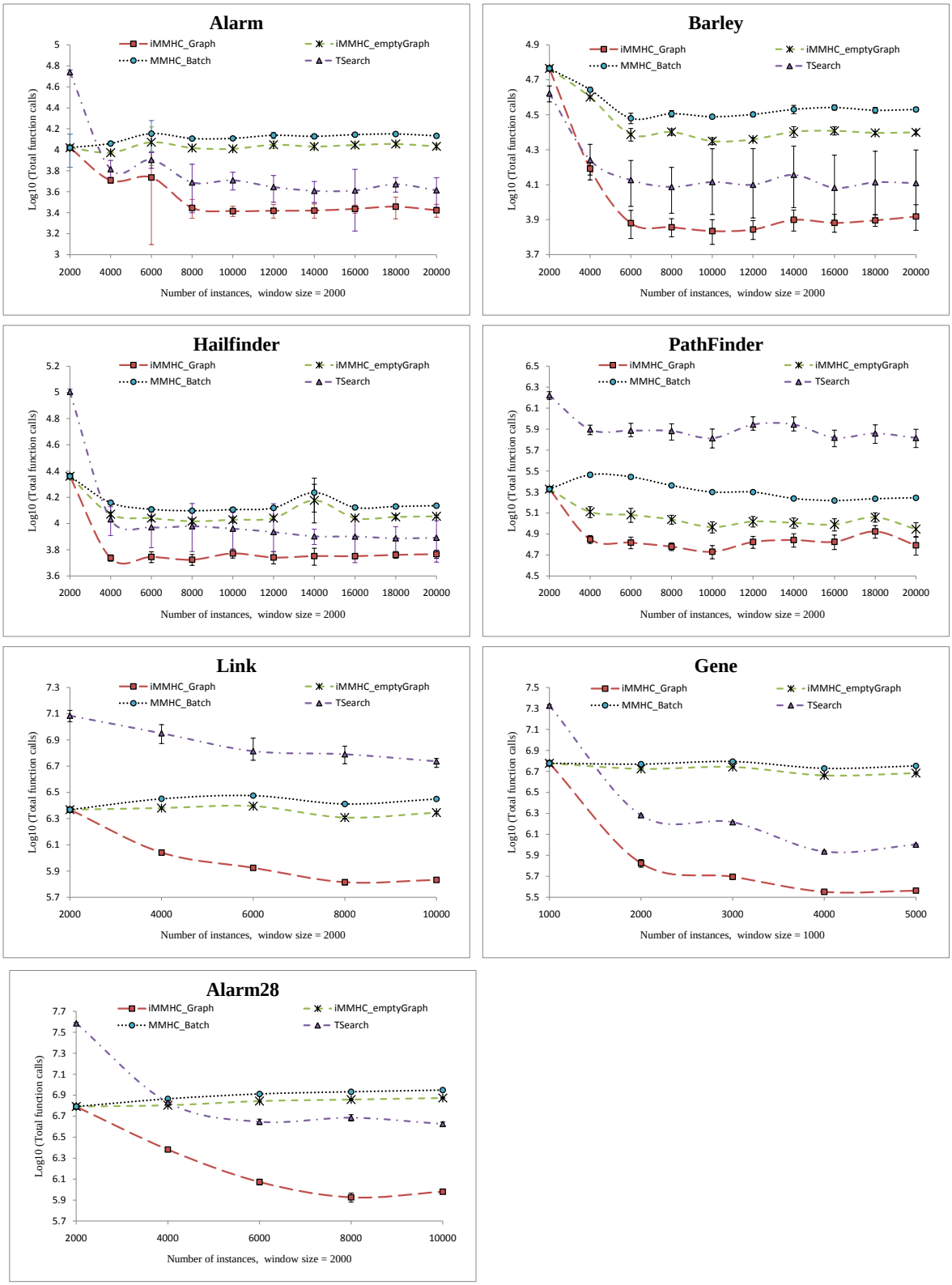


Figure 6.5: *Comparisons of Log10(total function calls) for $MMHC$, $iMMHC_{\varnothing}$, $iMMHC_G$ and $TSearch$ algorithms over an incremental process (using window of size 2000 (1000 for "Gene"))*

Further he defined an evaluation criteria as if this ratio is less than one, it means that an incremental algorithm is better than a batch one otherwise its counterpart is better. Therefore, he obtained "1.002" value for alarm network in [121]. In contrast, this ratio obtained by $iMMHC_G$ is "0.9561". Therefore, the ratio value smaller than one indicates that the model obtained by $iMMHC_G$ is better than a model obtained by batch $MMHC$.

6.4.3 Incremental versus batch MMHC

We can observe from the figure 6.4 that the model obtained by $iMMHC_G$ has a better quality than the model obtained by the batch $MMHC$, except for the datasets having high cardinalities (*Barley* and *Pathfinder*), while in *Link* dataset batch algorithm also gives an interesting accuracy. Whereas, the $iMMHC_\varnothing$ obtained overall same accuracy as batch $MMHC$ with a low complexity. These results are consistent with Roure's work in [121]. With respect to the high dimension of the search space, incremental algorithms can avoid being trapped in some local optima as could be their batch counterparts.

6.4.4 iMMHC for high dimensional domains

We have seen in chapter 4 that incremental Bayesian network structure learning were tested with benchmarks of about 37 variables. We used here two benchmarks; *Alarm28* with 1028 variables and *Gene* having 801 variables, to test the ability of our algorithm to deal with high dimensional domains. We can observe that the results of $iMMHC_G$ in figures 6.4 and 6.5 are much better than others. $iMMHC$ is an incremental algorithm where the first iteration has the same complexity as the batch algorithm but in the succeeding iterations, this complexity rapidly decreases. Therefore we can say $iMMHC$ is a good solution for incremental learning in high dimensional domains.

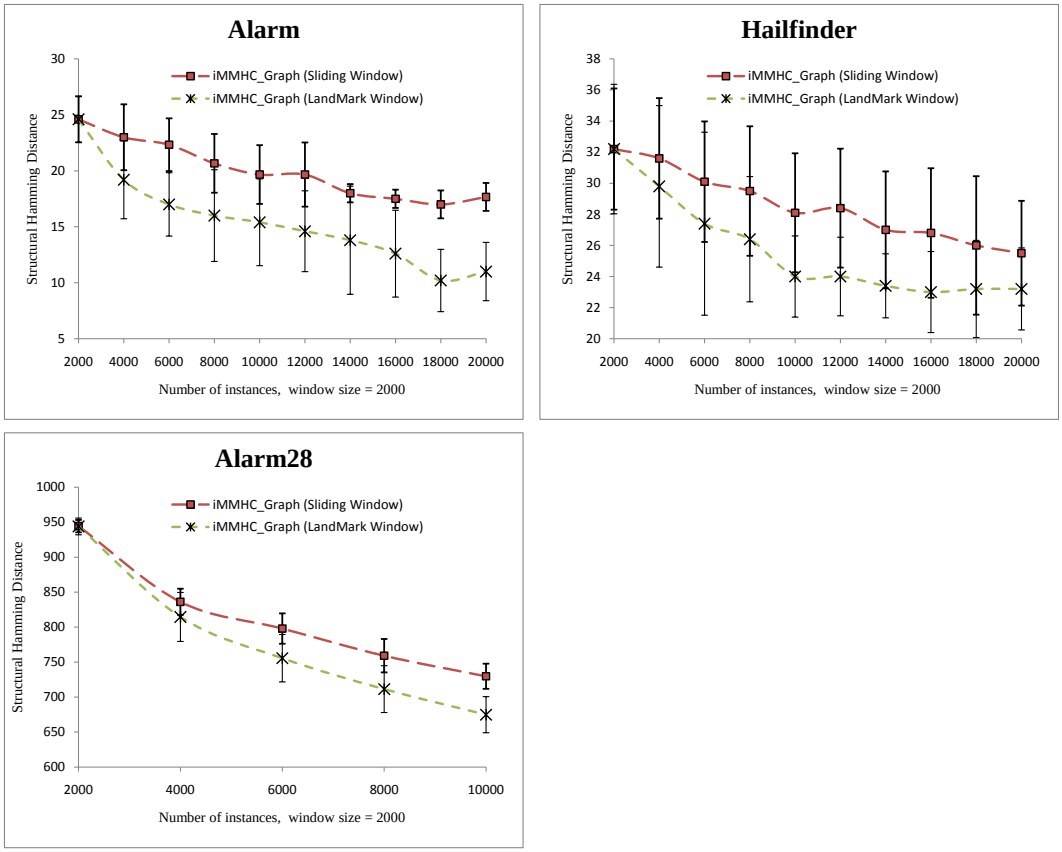


Figure 6.6: *Structural Hamming Distance (SHD) over sliding window.*

6.5 Results and interpretations with sliding window

Figure 6.6 compares the accuracy (SHD) obtained with sliding windows and landmark windows for datasets *Alarm*, *Hailfinder* and *Alarm28*. we can observe that using sliding windows the $iMMHC_G$ algorithm could not achieve the same accuracy as over landmark window. It is normal as discussed in section 2.2.1 about data stream processing that the results obtained over sliding window are approximated. In landmark window the algorithms have their access for whole previously seen data while in sliding window they have only approximated sufficient statistics as a summary of previous data but the SHD increase is not so bad, about 8% in Hailfinder and Alarm28 datasets and 40% in Alarm dataset.

Figure 6.7 shows the time gain in percentage over sliding window. It is calculated as $(1 - T_{SW}/T_{LW}) \times 100$, where T_{SW} and T_{LW} are the execution times for $iMMHC_G$ algorithm over sliding windows and landmark window, respectively. We can observe that sliding window saves significant amount of execution time

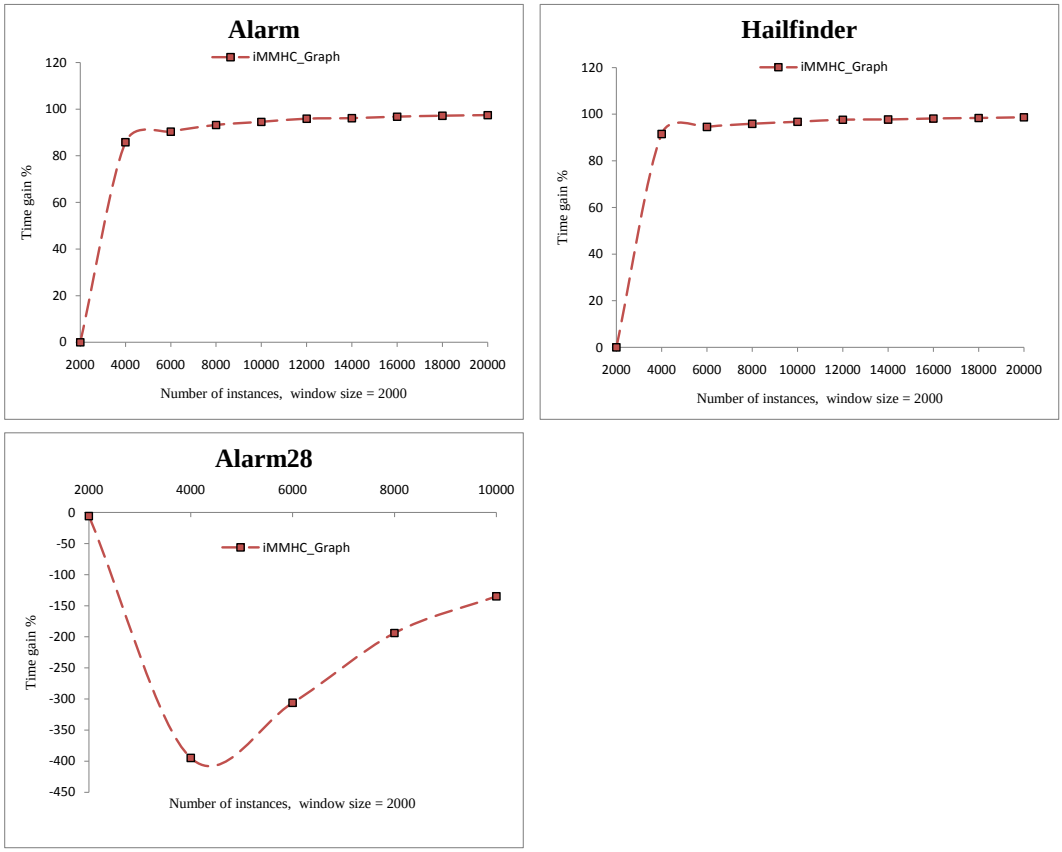


Figure 6.7: *$iMMHC_G$ execution time gain over sliding window.*

with respect to landmark window for datasets having small number of variables and simple networks. On the other hand, as discussed in section 3.3.4 regarding the complexity of probabilistic queries for datasets having large number of variables and complex networks so it takes more time as compare to landmark window, as seen in figure 6.7 for *Alarm28* dataset. It is more clear in first window of *Alarm28* while in succeeding windows it tries to recover this time as algorithm got more and more data. We are using an approximate inference technique using sampling method for probabilistic queries from large networks. The sampling size was adjusted to 500.

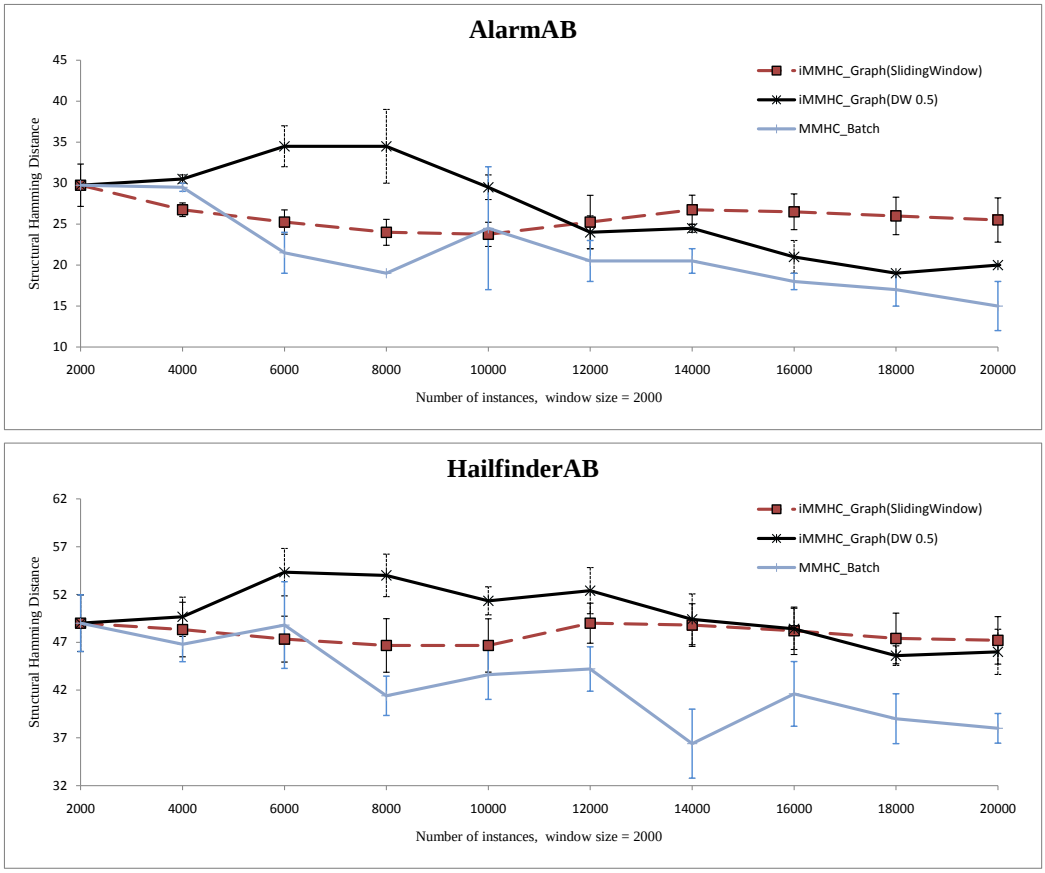


Figure 6.8: *Comparison of Structural Hamming Distance (SHD) for non-stationary domains over sliding, landmark and damped windows.*

6.6 Results and interpretations with damped window

6.6.1 Non-stationary domains:

We created a non-stationary environment by adding two datasets from two different networks such as: We generated data sets of 10,000 instances from original networks, *Alarm* and *Hailfinder* denoted by letter *A*. Then changed the original networks about 15% to 21% by deleting certain edges (10 out of 46 in *Alarm* and 10 out of 66 in *Hailfinder*) and generated datasets of 10,000 instances from the resulting networks called *B*. Later we merged these two datasets as *AlarmAB* and *HailfinderAB* such that the data generated from network *A* is placed before the data generated by network *B*. We sampled five datasets by this way and the results presented in this section are the mean and standard deviation of these datasets. We computed the SHD

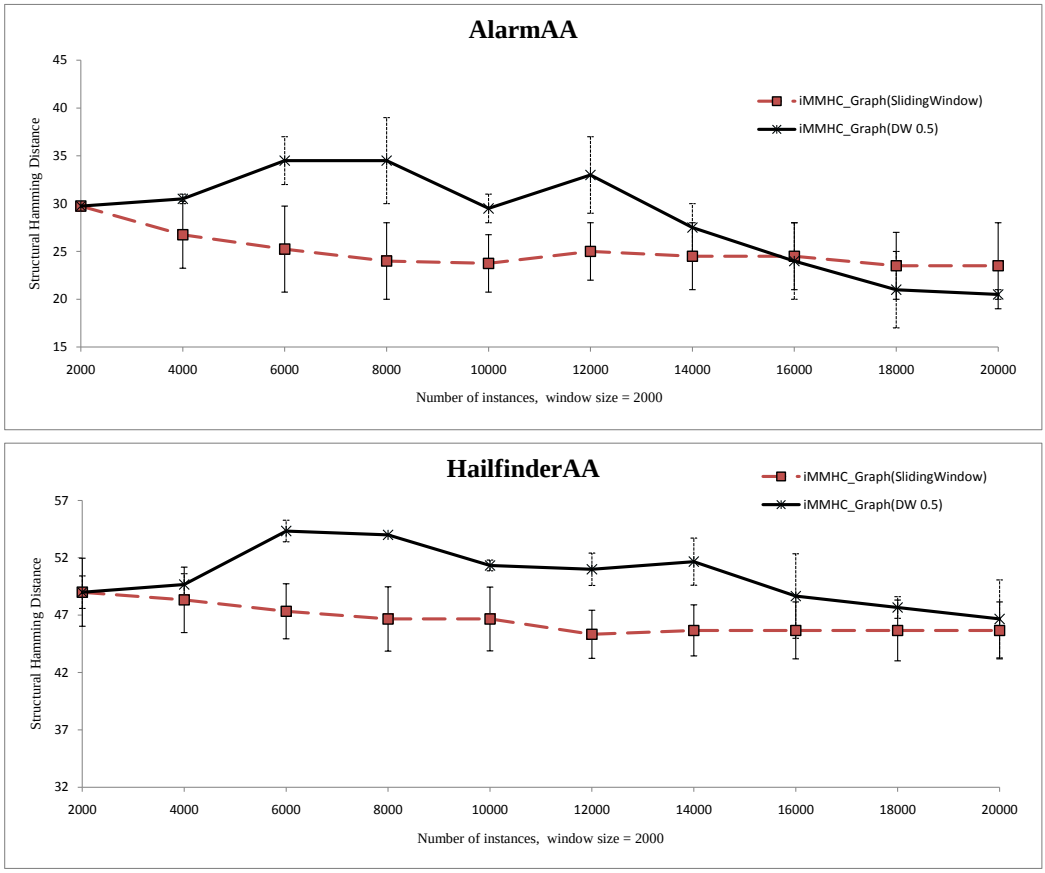


Figure 6.9: *Comparison of Structural Hamming Distance (SHD) for stationary domains over sliding and damped windows.*

by comparing the results of each window with their respective original networks.

Then we applied $iMMHC_G$ algorithm using sliding and damped windows. We also applied $MMHC$ batch algorithm for each network. For damped window we adjusted fading factor α to 0.5.

From figure 6.8, we can observe that for sliding windows, the SHD increases when drift occurs at fifth window (number of instances = 10,000) because the previous model used in order to estimate the sufficient statistics for past data is not relevant, so the SHD increases and even after the drift it could not recovered. While with damped window (DW 0.5) SHD increases in the beginning because we are applying forgetting factor, but after the drift forgetting factor helps to correct it and it starts decreasing. It is more visible for *AlarmAB* than for *HailfinderAB* datasets. Therefore, forgetting factor helps in discarding old data and dealing with non-stationary domains.

6.6.2 Stationary domains:

To test $iMMHC_G$ algorithm for stationary domains over damped window, we generated two datasets of 10,000 instances from *Alarm* and *Hailfinder* networks then merged them as a single dataset containing 20,000 data records, *AlarmAA* and *HailfinderAA*. Later, we applied $iMMHC_G$ algorithm for damped window with fading factor 0.5 and for sliding window.

Results of this section are presented in figure 6.9. We can observe that SHD increases in the beginning for damped window (DW 0.5) but later forgetting factor helps to recover it and it decreasing.

As seen before, damped windows helps for dealing with non-stationary domains. If the domain is constant (stationarity), using damped windows generates a decrease in term of accuracy during the beginning, but the difference is no more significant after a few iterations.

6.7 Conclusions

This chapter presents an empirical evaluation of our propositions described in chapter 5. Several experiments were conducted to assess the performance and the accuracy of the proposed methods. These experiments used datasets from benchmarks of Bayesian networks like *Alarm*, *Barley*, *Hailfinder*, *Pathfinder*, *Link*, *Gene* and a synthetic data set of *Alarm28*. Comparison were conducted with batch $MMHC$ algorithm and in recent state of the art algorithm $TSearch$. The experimental evaluations given in existing state of the art are very limited, in terms of number of variables and their cardinalities in the datasets.

Some of the results obtained in our experiments are presented in this chapter. The experiments show very encouraging results when we compare $iMMHC_G$ algorithm with similar incremental algorithm ($TSearch$) and the batch original one ($MMHC$) as well. Our algorithm achieved results that are comparable in performance and accuracy to the $TSearch$ and with batch $MMHC$ algorithm. We remind the readers that the skeleton discovery phase of $iMMHC$ algorithm

uses *constraint* based approach, which has already less complexity than *score-and-search* based techniques. As a consequence $iMMHC_G$ could also be an interesting alternative to batch learning for large databases.

As discussed in chapter 2 about data streams, some of the main constraints for learning algorithms are time and memory management, where algorithms have limited time to process the data and limited memory to store sufficient information about data. It is a difficult task to maintain a tradeoff between the accuracy, time and memory. Results presented here for sliding and damped windows show that our algorithm can handle these constraints with a limited cost of accuracy, but with a cost of computational time for complex networks.

III

Conclusion and perspectives

7

Conclusion and perspectives

7.1 Summary

In last the decade data stream mining became an active area of research. There are a lot of works that address different methods to analyze rapidly arrived data in real time. We can store and process the new and old data with a batch algorithms in order to learn a new model. But it requires lot of computing resources. The unboundness nature of data stream do not allow us to store all data in the memory. The recent explosion of high dimensional datasets in the field of social networks, bioinformatics, telecommunication etc, having several hundreds and thousands of variables. It poses a serious challenge for existing Bayesian network structure learning algorithms [138].

Bayesian network learning includes parameter learning and structure learning of the network. In this manuscript, we focused on structure learning part. Bayesian network structure learning aims at selecting a probabilistic model that explains a given set of data, which is a NP-hard problem. There are different ways to find the structure of the network including *score-and-search* based, *constraint* based

and *hybrid* methods. Each has its own pros and cons while hybrid method take advantage of both score and constraint based methods and give a better tradeoff between accuracy and complexity.

Bayesian network structure learning from static data is already well explored area of research. While incremental Bayesian network structure learning from data stream is not a mature area of research. As far as we know there exist few approaches that address this problem with the name of *incremental learning*, *sequential update*, *refinement of structure* and *structural adaptation*. If we separate the network update process from learning then we see most of the approaches deal with the update of existing network structure rather than learning. While they consider the existing network structure as a true representative of the current data.

We can divide these approaches in two main categories with respect to their assumption about incoming data and how they interact with the data, first stationary domain and second non-stationary domains. Most of the techniques use *score-and-search* based structure learning method, which could not scale up for high dimensional domains. As far as we know there exist only one *Shi and Tan's* proposal for incremental Bayesian network structure learning that is using hybrid technique. But they have not directly addressed the problem of high dimensionality. In our experiments we see their method includes lot of false positives in the set of CPCs of each variable during their first local search phase. As a consequence it enlarges the search space for the second phase known as global optimization. Therefore it does not accomplish its task in a limited time.

This thesis addresses the problem of Bayesian network structure learning from data streams, especially in high dimensional domains. We used hybrid technique to learn the structure of Bayesian network as it is scalable for large number of variables. Moreover, we proposed an incremental max-min hill-climbing method for Bayesian network structure learning from data stream. The goal of the learning task is to discover a new structure based on set of new data examples, existing network structure and available sufficient statistics.

In the first attempt we proposed an incremental constraint based local search

algorithm (iMMPC), that is learning local structures around each variable using incremental hill-climbing way. Later, we proposed an incremental MMHC algorithm to learn whole structure of the network. These algorithms use *landmark window* to deal with incoming data. We introduced a frequency counter that works as a middle layer between data and the algorithm. It serves as a cache to speed up the learning process. Further we have adapted $iMMHC$ for sliding window by storing sufficient statistics. Our way to store sufficient statistics is similar to Friedman's one (cf. section 4.2.1.3) as we also do not rely only on MAP and neither store counts for all combinations as in *AD-tree*. Finally to react with recent trends in data, we proposed a forgetting mechanism at frequency counter. The main objective of our approach is to spend less time than a batch learning scenario and to store limited amount of informations that are useful for incremental learning process.

Theoretical study and preliminary experiments show our approach improves the performance of the algorithm systematically, reducing the complexity significantly.

7.2 Perspectives and open problems

Despite the convincing results of our experiments, there are number of interesting points that need further investigation. These points could improve or extend the work presented in this manuscript.

Theoretical study

During our study we observed the behavior of forward phase of $\overline{MMPC}(T)$ and found that it works as a hill climbing search as mentioned in the conjecture 1 at page no. 94. We would like to proof this conjecture in order to justify this behavior.

Empirical study

We would like to conduct more experiments with $iMMHC$ for sliding window and damped window, especially for non-stationary domains. Our objective is to characterize different parameters that affect the efficiency of our algorithm in tracking

the changes in the data. We also want to conduct experiments in real data stream environment.

Improving iMMHC

It is possible to identify some specific changes in specific CPCs then greedy search phase can be restricted to these changes without applying for all variables in the network.

In learning with fading factor paradigm, the current model can represent the recent trends in the data and gradually forget the outdated informations. But in the case of sudden change where the data stream follow a totally new concept (distribution), the learning algorithm should discard the old concept and follow the new ones. This means changing window size when a concept drift is detected. Therefore we want to detect the sudden change and discard the previous data and to keep recording the new one only.

The major constraints for data stream learning algorithms are the time and memory. By properly managing the sufficient statistics update part, algorithm can react efficiently. As a consequence it can obtain a better accuracy. Therefore we want to study the impact of different approximate reasoning methods (cf. section 3.3.4.2) upon the accuracy for complex networks. We also want to extend scoring cache (cf. section 6.3.2) for incremental optimization. By doing this we will be able to store very concise information rather than raw information (i.e. counts) and it will also be helpful to solve these two constraints.

There is another interesting idea is that the users some time interested in only newly arrived and dispersed edges. Therefore we can highlight the changes in structure.

As Bayesian network structure learning is a NP-hard task, therefore the first phase of our algorithm can be parallelized the problem by executing on multiple computing resources. Where the set of CPC for each variable can be computed parallel, thereby giving an opportunity for the bottleneck to be alleviated.

Some other directions

In this study we are considering a constant number of variables, but we could consider an open world assumption in data stream that new variables can occur and other ones can disappear, etc. Therefore, it is also interesting to accommodate these assumption in incremental Bayesian network structure learning.

In order to improve scalability of incremental Bayesian network structure learning methods for high dimensional domains, we can consider a simple Bayesian network structure learning models such as trees or mixtures-of-trees in order to have learning algorithms with simple (log-linear) complexity [5].

Another interesting direction of research is the development of incremental Bayesian network structure learning method for distributed data stream. There is a similar approach by Chen et al [18], they first learn a local Bayesian network at each site using the local data. Another Bayesian network is learnt at the central site using the data transmitted from the local site. Then local and central Bayesian networks are combined to obtain a collective Bayesian network.

Recently, Probabilist Relational Models (PRMs) [47] are used for *collaborative filtering* and *information filtering* in recommendation systems [60]. It is interesting to online learning the PRMs. We can also consider to learn PRMs incrementally from multi-dimensional data streams [75].

There is another possible direction to consider time series data for online dynamic Bayesian network learning [145, 28].

A

Comparison table for different data stream mining techniques

This annex gives complementary material to the state of the art provided in section 2.4.2.

Algorithm	Mining Task	Advantages	Disadvantages
GEMM and FOCUS [58]	Decision tree and frequent item sets	– Concept drift detection – Incremental mining models	– Time consuming and costly learning
OLIN [93]	Uses info-fuzzy techniques for building a tree-like classification model	- Dynamic Update	– Low speed – Storage memory problem – Time consuming and costly learning
VFDT and CVFDT [21]	Decision Trees	– High speed – Need less memory space	– Non-adaption to concept drift – Time consuming and costly learning
LWClass [51]	Classification based on classes weights	– High speed – Need less memory space	– Non-adaption to concept drift – Time consuming and costly learning
CDM [88]	Decision tree and Bayes network	Suitable factor for distance measurement between events	– User defined information complexity

On-demand stream classification [3]	Using micro-clusters ideas that each micro-cluster is associated with a specific class label which defines the class label of the points in it.	– Dynamic update – High speed – Need less memory space	– High cost and time need for labeling
Ensemble-based Classification [144]	Using combination of different classifiers	– Single pass – Dynamic update – Concept drift adoption – High accuracy	– Low speed – Storage memory problem – Time consuming and costly learning
ANNCAD [95]	Incremental classification	- dynamic update	– Low speed – Storage memory problem – Time consuming and costly learning
SCALLOP [46]	Scalable classification for numerical data streams.	- dynamic update	– Low speed – Storage memory problem – Time consuming and costly learning

Table A.1: *Comparison of data stream mining techniques for Classification [80]* .

Algorithm	Mining Task	Advantages	Disadvantages
STREAM and LOCAL SEARCH [110]	K-Medians	- Incremental learning	– Low clustering quality in high speed – Low accuracy
VFKM [42, 43, 73]	K-Means	– High speed – Need less memory space	- Multi-pass
CluStream [2]	The concepts of a pyramidal time frame in conjunction with a micro-clustering approach.	– Time and space efficiency – concept drift detection – High accuracy	Offline clustering
D-Stream [19]	Density-based clustering	– High quality and efficiency – Concept drift detection in realtime data stream	- High complexity
AWSOM [113]	Prediction	– Efficient pattern detection – Need less memory space - dynamic update Single pass	- High complexity

HPStream [21]	projection based clustering	– Efficient for high dimensional data stream – Incremental update – High scalability	- High complexity

Table A.2: *Comparison of data stream mining techniques for clustering [80]* .

Algorithm	Mining Task	Advantages	Disadvantages
Approximate Frequent Counts [100]	Frequent item sets	– Incremental update – Simplicity – Need less memory space – Single pass	- Approximate output with increasing error range possibility
FPStream [61]	Frequent item sets	– Incremental and dynamic update – Need less memory space	- High complexity

Table A.3: *Comparison of data stream mining techniques for frequency counting and Time Series Analysis [80].*

B

Résumé en langue française

B.1 Introduction

De nombreuses sources de données produisent de l'information en continu : flux de clicks de clients sur les sites internet, appels téléphoniques, données de transactions de différentes natures, etc. Ces flux de données sont souvent définis comme des séquences continues et ordonnées d'items qu'il n'est pas possible de stocker dans leur totalité ni de manipuler facilement. La fouille de flux de données revient donc à extraire des informations pertinentes, des connaissances, d'un tel type de données. Les grandes familles de méthodes de fouille de données comme la recherche d'associations, la classification ou le clustering ne peuvent généralement pas s'appliquer directement aux flux de données dans lesquels il est impossible et irréaliste de parcourir plusieurs fois les données afin d'en extraire des informations pertinentes. De plus, les caractéristiques du flux de données peuvent évoluer au cours du temps, et cette évolution doit aussi être capturée [55]. Les spécificités des flux sont telles qu'il faut aussi considérer avec attention le problème d'allocation de ressources. De part le volume des données et la vitesse du flux d'informations, les algorithmes de fouille doivent très rapidement faire face à des situations de surcharge. Dans un tel contexte, la construction d'algorithmes de fouille de flux de données est un défi stimulant.

Les réseaux bayésiens (RB) sont des modèles graphiques probabilistes définis par un graphe orienté sans circuit, aussi appelé la structure du modèle, décrivant un ensemble de propriétés de dépendances ou d'indépendances conditionnelles entre les nœuds du graphe, et des paramètres, i.e. un ensemble de tables de probabilités conditionnelles décrivant la force de ces dépendances. L'aspect graphique de ces modèles et le fait de pouvoir construire ce graphe à l'aide d'expertise et/ou d'algorithmes d'apprentissage font de ces modèles des outils de plus en plus utilisés en extraction et en gestion des connaissances. L'apprentissage de la structure des RB est un problème NP-difficile [22] qui a mené à la proposition de nombreuses heuristiques que l'on peut classer sommairement en trois familles. Les méthodes *à base de recherche d'indépendances conditionnelles* [132] exploitent les propriétés

d'indépendance probabiliste (estimées dans les données par des tests statistiques classiques) et les propriétés d'indépendance graphique pour reconstruire un graphe qui soit le plus représentatif des indépendances trouvées dans les données. Ces méthodes sont vite limitées par le manque de fiabilité du test lorsque le nombre de variables concernées augmente.

Les méthodes *à base de score* [26] cherchent à optimiser un critère défini en parcourant de manière heuristique l'espace des solutions. Malheureusement, l'espace de recherche étant de taille super-exponentielle par rapport au nombre de variables, il devient très difficile de trouver une bonne solution en grande dimension. Les méthodes *hybrides* combinent une phase de recherche locale et une autre phase d'optimisation globale. La recherche locale consiste à effectuer des tests d'indépendance conditionnelle pour identifier la structure locale autour d'une variable cible, par exemple l'ensemble des parents-enfants candidats (PC) ou la couverture de Markov (MB), à l'aide d'heuristiques comme *MMPC* [136] ou *MBOR* [120]. L'optimisation globale sert alors à identifier la structure totale du modèle par utilisation d'une méthode à base de score contrainte par les voisinages identifiés lors de la recherche locale. Ainsi l'algorithme *MMHC* [138] combine *MMPC* pour découvrir une enveloppe supérieure du squelette du graphe et recherche gloutonne pour affiner ce squelette et orienter le modèle.

Dans de nombreux domaines d'application comme la bio-informatique ou les réseaux sociaux, il est assez courant d'avoir de l'ordre de 10.000 à 100.000 variables disponibles. Les méthodes d'apprentissage à base de recherche d'indépendance conditionnelle ou de scores sont alors très nettement dépassées. Seules les méthodes d'apprentissage hybrides arrivent à travailler dans de telles dimensions.

B.2 Approches existantes

L'apprentissage incrémental de la structure d'un réseau bayésien a donné lieu à de nombreux travaux ne traitant pas forcément la non-stationnarité liée aux flux de données. Une première famille d'approches traite le cas *stationnaire* où le système

sous-jacent n'évolue pas au cours du temps. L'autre famille d'approches, plus réduite, traite le cas *non stationnaire* en prenant en compte les dérives ou décalages potentiels.

Dans le cas stationnaire, [15] propose un algorithme séquentiel de recherche à base de score puis décrit ensuite comment le convertir en un algorithme d'apprentissage incrémental. Il considère deux conditions : tout d'abord s'il n'y a pas assez de temps pour mettre à jour la structure du graphe, l'algorithme ne mettra à jour que les paramètres (tables de probabilité conditionnelles) du modèle. Dans le cas contraire, l'algorithme mettra à jour à la fois la structure et les paramètres. L'approche de [92, 90] est aussi une approche séquentielle, basée sur le principe de longueur de description minimale (MDL). L'idée est d'apprendre une structure partielle à partir des nouvelles données et de la structure existante puis de modifier localement cette structure existante à l'aide de la nouvelle structure partielle. [48] proposent trois approches différentes. La première approche, la plus naïve, stocke toutes les données précédentes et appelle le même algorithme d'apprentissage à chaque itération. La seconde approche se base sur l'estimation de la probabilité a posteriori du modèle. La troisième approche, appelée incrémentale par ses auteurs, maintient un ensemble de structures candidates nommé *frontière* de la procédure de recherche. Lorsque de nouvelles données arrivent, la procédure met à jour les informations stockées en mémoire et déclenche la procédure de recherche pour vérifier si un des modèles de la frontière est devenu plus intéressant que le modèle actuel. [121], de son côté, mémorise la séquence ayant permis de construire le modèle actuel et reconstruit la structure du nouveau modèle à partir de la portion de séquence qui n'est plus optimale avec les nouvelles données obtenues. Il propose ainsi deux heuristiques permettant de transformer une recherche gloutonne classique en une méthode incrémentale. La section B.3.2 décrit plus précisément ces heuristiques. [124] discute alors plus généralement des méthodes d'apprentissage de structure incrémentales. Toutes les approches précédentes sont des adaptations d'algorithmes d'apprentissage à base de score, et traitent le cas stationnaire. Récemment, [129] a adopté une approche incrémentale hybride proche de celle que nous proposons ici.

Concernant le cas non stationnaire, [109] proposent une approche prenant en compte la dérive du système sous-jacent. Cette approche consiste en deux mécanismes, le premier surveille et détecte où et quand le modèle doit être mis à jour, et le second réapprend localement les parties du modèle en conflit avec les nouvelles données. [16] proposent eux aussi un algorithme adaptatif dans un environnement non stationnaire.

En conclusion, nous voyons d'un côté que la plupart des méthodes existantes utilisent des méthodes à base de score en traitant différemment l'aspect incrémental. Malheureusement ces méthodes ne passent pas à l'échelle en grande dimension, comme le font les méthodes hybrides. D'un autre côté, les méthodes non stationnaires ont mis en évidence l'efficacité d'une reconstruction locale des parties du modèle ayant évoluées.

Pour cette raison, nous proposons de tirer avantage des approches de recherche locale en proposant l'algorithme *iMMPC* algorithme incrémentale de recherche locale pour des problèmes stationnaires de grande dimension dans la section B.4. Ensuite l'algorithme incrémentale Max- Min hill climbing *(iMMHC)* pour obtenir une structure global de réseau bayésien finale dans la section B.5. Enfin, on adapter ces algorithmes sur les fenêtres glissement et à amortissement pour cas non-stationnaire dans les section B.6 et section B.7.

B.3 Contexte

B.3.1 MMPC(T): recherche locale pour l'apprentissage de la structure d'un réseau bayésien

L'algorithme *MMPC(T)*, Max-Min Parent Children [138]) construit progressivement un ensemble CPC(T) (parents-enfants candidats, sans distinction) pour une variable T. C'est la combinaison de la procédure $\overline{MMPC}(T)$ (cf. Algo. 4) et d'une correction additionnelle non décrite ici permettant de conserver la symétrie de la relation parent-enfant.

$\overline{MMPC}$(T) se déroule en deux phases. Dans la *phase forward* il ajoute progressivement dans CPC(T) des variables candidates les plus directement liées à T et dans la *phase backward* il supprime un certain nombre de faux positifs ajoutés dans la première phase. Le cœur de l'algorithme *MMPC* est ainsi la phase *forward* de *$\overline{MMPC}$(T)*, qui nécessite le plus de calculs de mesures d'association.

La fonction *min(Assoc)* utilisée dans la phase *forward* mesure l'association directe entre T et une variable candidate X conditionnellement à un ensemble CPC. Sa valeur est nulle si les deux variables sont indépendantes conditionnellement à au moins un sous ensemble de CPC. Ainsi des variables ayant une mesure d'association nulle pour un ensemble CPC donné ne pourront jamais entrer dans CPC(T) et pourront être supprimées de l'espace de recherche des candidats pour les itérations suivantes. Le niveau d'association peut être estimé par des mesures comme le χ^2 ou l'information mutuelle (MI), estimées à partir des données. La variable ajoutée à l'ensemble CPC est alors celle qui maximise cette mesure d'association *min(Assoc)* conditionnellement au CPC courant, d'où le nom de l'heuristique *MaxMinHeuristic* et de l'algorithme *MMPC* proposés par [**?**]. La phase *forward* s'arrête lorsque il n'y a plus de variable directement dépendante avec T sachant le CPC courant.

B.3.2 Adaptation incrémentale des méthodes d'apprentissage à base de score

Les travaux de [121] proposent deux heuristiques permettant de transformer un algorithme classique de recherche gloutonne (*HCS*, Hill-climbing search) comme celui proposé par exemple par [26] en un algorithme incrémental. Nous allons donc décrire tout d'abord l'algorithme *HCS* non incrémental puis les heuristiques proposées pour la version incrémentale *iHCS*.

Hill Climbing Search (HCS) L'algorithme HCS (cf. Algo. 1) parcourt l'espace des graphes de manière itérative, de proche en proche, en examinant des changements locaux du graphe et en sélectionnant à chaque étape le graphe qui maximise une fonction de score donnée. Le **voisinage** d'un modèle *M* est défini par l'ensemble

de tous les graphes voisins générés à l'aide d'**opérateurs** *op* et des arguments A associés, où l'opérateur peut être *Ajout Arc*, *Suppression Arc* ou *Inversion Arc*. Une **fonction de score** $f(M, \mathcal{D})$ sert à mesurer la qualité du modèle pour un ensemble de données fixé.

Soit M_0 le modèle initial et M_f le modèle final obtenu par l'algorithme *HCS* avec $M_f = op_n(...op_2((op_1(M_0\, A_1),\, A_2), ...,\, A_n)$ où chaque opérateur (et arguments) mène au modèle ayant le meilleur score dans le voisinage. Le ***chemin de recherche*** ou de "traversée" est la séquence d'opérateurs (et d'arguments) utilisés pour construire le modèle final M_f, $\mathcal{O}_{op} = \{(op_1,\, A_1)\,,\, (op_2,\, A_2)\,, \ldots\, (op_n,\, A_n)\}$ utilisée pour construire M_f. Il permet ainsi de reconstruire la séquence de modèles intermédiaires ayant mené à ce modèle. Notons que le score des modèles intermédiaires augmente de manière croissante le long de ce chemin.

Incremental Hill Climbing Search (iHCS) Avec l'algorithme *iHCS* (cf. Algo. 6) Roure propose d'utiliser le chemin de recherche précédemment défini pour développer deux heuristiques basées sur la stationnarité des données. Le chemin de recherche est supposé ne pas trop changer lorsque de nouvelles données arrivent, ou lorsque ces données s'écartent très peu de la distribution initiale, ce qui permet à la fonction de score d'être continue dans l'espace des ensembles de données.

La première heuristique *Traversal Operators in Correct Order* (TOCO) vérifie l'adéquation du chemin de recherche existant aux nouvelles données. Si ces données altèrent ce chemin, il est alors nécessaire de mettre à jour le modèle.

La seconde heuristique *Reduced search space* (RSS) est utilisée pour la mise à jour du modèle. De manière générale, l'algorithme ne stocke pas que le modèle intermédiaire optimal obtenu à chaque étape de la recherche gloutonne, mais l'ensemble $\mathcal{B}$ des k meilleurs modèles. L'utilisation de cet ensemble $\mathcal{B}$ permet de réduire l'espace de recherche en évitant d'explorer des parties de l'espace qui menaient précédemment à des modèles de faible qualité.

B.4 Une approche locale pour l'apprentissage incrémental

Nous présentons ici notre algorithme incrémentale de recherche locale *iMMPC(T)* dans le cas stationnaire. Comme discuté précédemment le cœur de l'algorithme *MMPC(T)* est la phase *forward* de $\overline{MMPC}$*(T)* dans laquelle les variables sont ajoutées progressivement dans CPC(T). Nous allons montrer que cette phase peut être décrite comme une recherche gloutonne.

B.4.1 Interprétation "gloutonne" de la phase *forward* de $\overline{MMPC}$*(T)*

Comme indiqué précédemment, la partie pivotante de la *MMPC(T)* est l'algorithme la phase avant de $\overline{MMPC}(T)$.. Dans cette phase de variables entrent en l'ensemble des CPC (T) d'une variable cible T progressivement, en utilisant *MaxMinHeuristic* (cf. équation 3.21). Cette heuristique trouve d'abord l'association minimale de chaque variable X par rapport à un sous-ensemble S de déjà sélectionné CPC, noté "$min_{S \subseteq CPC} Assoc\,(X;\, T \mid S)$". Plus tard, il sélectionne les variables comme un CPC qui maximise l'association minimum avec variable cible T. Il nécessite de démontrer cette pièce agit comme un processus de recherche gloutonne dans un ordre décroissant. Après consultation de la littérature disponible actuelle et l'expérimentation, mathématiquement, il est difficile de le prouver, par exemple:

Exemple: Supposons quatre variables X_1, X_2, X_3 et X_4. La variable cible $T = X_1$ et nous devons calculer son ensemble de CPC en utilisant l'information mutuelle comme une mesure d'association. Lors de sa première itération, on calcule $MI(X_1, X_2)$, $MI(X_1, X_3)$, $MI(X_1, X_4)$. MIpX1; X4q. Considérons obtenu en MI sont ordonnés de façon décroissante en:

$$MI(X_1, X_2) > MI(X_1, X_3) > MI(X_1, X_4)$$

Donc, nous choisissons X_2 et $CPC = \{X_2\}$. Maintenant, pour la deuxième itération, nous calculons $MI(X_1, X_3)$ et $MI(X_1, X_3|X_2)$, et nous devons choisir celui minimum. De même, nous calculons $MI(X_1, X_4)$ et $MI(X_1, X_4|X_2)$, et choisissez celui qui minimum. Sélectionnez ensuite le maximum, de minimum préalablement choisis.

Si le maximum est $MI(X_1, X_3|X_2)$, alors nous savons que $MI(X_1, X_3|X_2) \leqslant MI(X_1, X_3) \leqslant MI(X_1, X_2)$. Et si le maximum est $MI(X_1, X_4|X_2)$, alors nous savez que $MI(X_1, X_4|X_2) \leqslant MI(X_1, X_4) \leqslant MI(X_1, X_2)$. Par conséquent, dans les deux cas, la valeur de la mesure d'association diminué. Ici, nous supposons $X3$ est sélectionné dans ce itération comme un CPC donc, $CPC = \{X_2, X_3\}$.

Dans sa troisième itération, nous calculons $MI(X_1, X_4)$, $MI(X_1, X_4|X_2)$, $MI(X_1, X_4|X_3)$ et $MI(X_1, X_4|X_2, X_3)$ et de garder une minimum qui est $\leqslant MI(X_1, X_4)$. Ce est facile de montrer que $MI(X_1, X_4|X?)$ [1] $< MI(X_1, X_2)$, mais mathématiquement, il est difficile de prouver que $MI(X_1, X_4|X?) < MI(X_1, X_3|X?) < MI(X_1, X_2)$. Alors que, pendant notre expérimentation nous avons constaté que la valeur de la fonction objectif diminué entre deux itérations pour les variables sélectionnées par rapport à la cible variables i.e. $MI_{1^{st}\,iteration} > MI_{2^{nd}\,iteration} > ... > MI_{n^{th}\,iteration}$. On a observé dans plusieurs expériences sur différents jeux de données. On a observé dans plusieurs expériences sur différents jeux de données. Il y a une tendance évidente qui suggère la conjecture (cf. conjecture 1, page 94).

Par conséquent, dans cette étude, nous faisons l'hypothèse que la fonction objectif a diminué à chaque itération de la recherche locale pour un ensemble de CPC d'une variable cible.

Les principales composantes de recherche gloutonne sont:

– un modèle
– Une fonction de notation pour mesurer la qualité des modèles
– L'espace de recherche

1. $X?$ pourrait être un X_2 ou X_3 variable ou rien.

Dans l'environnement de la recherche locale, $\overline{MMPC}(T)$ trouve l'ensemble des candidats parents-enfants pour une étape de T variable cible par étape (sans faire de distinction entre les parent et enfants des variables). Ici, nous pouvons dire que le modèle local pour une variable cible T se composent de ses CPC. Nous utilisons le $MaxMinHeuristic$ en fonction de notation pour mesurer l'association entre les variables, par conséquent, de nouvelles variables ajoutées dans l'ensemble des CPC d'une cible T variable sur la base de cette fonction. Sa valeur diminue entre deux itérations, comme on le verra dans l'exemple de la section B.4.1 et notre hypothèse.

L'espace de recherche pour ce modèle local est compromis de toutes les variables V, sauf la variable cible. La variable trouvé conditionnelle / inconditionnelle indépendante se retirer de l'espace de recherche.

B.4.2 Notre proposition : *MMPC(T)* incrémental

En démontrant le caractère "glouton" de la phase *forward* de *$\overline{MMPC}$(T)*, il nous est désormais possible d'adapter les heuristiques TOCO et RSS qui rendaient incrémentale une recherche gloutonne classique (Algo. 6), Nous supposons que le modèle comme un ensemble de CPC et une variable cible. Notre *modèle initial* M_0 correspond à un ensemble CPC vide.

Puis il ajoute séquentiellement les variables en elle, ayant des dépendances directes fortes en utilisant l'algorithme de MaxMinHeuristic MMPCpTq.

Notre version incrémentale de *MMPC(T)* démarre avec ce *modèle initial* M_0 et cherche le meilleur voisin. Ce processus itératif se poursuit tant que les variables ajoutées ont une mesure d'association significative avec la variable cible en utilisant de *MaxMinHeuristic* de l'algorithme *$\overline{MMPC}$(T)*. Ainsi le *chemin de recherche* de *iMMPC* correspond à la séquence des variables ajoutées au fur et à mesure dans CPC(T). A chaque étape sur ce chemin de recherche, nous conservons dans un ensemble $\mathcal{B}$ les k variables du voisinage ayant une mesure d'association le plus proche de la variable optimale.

A l'arrivée de nouvelles données *iMMPC(T)* vérifie d'abord le chemin de recherche courant pour identifier le modèle M_i à partir duquel il faudra relancer

la recherche gloutonne en appliquant les heuristiques TOCO et RSS présentées précédemment.

Il continue d'améliorer ce modèle pour introduire nouvelles variables dans la CPC dans le modèle final. Pendant ce temps, les variables ont trouvé indépendante sont supprimés à partir de l'ensemble des variables V pour éviter des calculs non nécessaires et d'économiser la durée de fonctionnement. La phase *forward* poursuit jusqu'à ce que le modèle en cours trouvée comme un modèle final et il n'y a plus d'amélioration possible.

La phase *backward* de $\overline{MMPC}(T)$ entraîne beaucoup moins de calculs de mesure d'association. Nous proposons donc de conserver cette phase à l'identique dans notre version incrémentale *iMMPC*.

B.4.3 Exemple jouet

Nous proposons ici un exemple jouet permettant de mieux comprendre le fonctionnement de notre algorithme incrémental. Supposons que les données provenant du flux suivent le modèle non dirigé de la Figure 5.3. Les données d'entrée sont échantillonnées à partir de la distribution du graphe original. Concentrons-nous sur l'identification locale de $CPC(X6)$, voisinage de $X6$.

Présentons deux situations correspondant aux deux colonnes de la figure 5.3. Dans la colonne de gauche, nous travaillons avec un premier ensemble de données $\mathcal{D}$ et dans la colonne de droite nous travaillons avec $\mathcal{D} \cup \mathcal{D}'$. A gauche, nous commençons par rechercher dans le voisinage complet et générons l'espace de recherche réduit $\mathcal{B}$ (avec $k = 4$) $\mathcal{B}$.

iMMPC va démarrer du modèle initial vide M_0 contient seule variable cible et un vide ensemble de CPC et va générer un voisinage en considérant toutes les variables de V un par un où $V = \{X1, X2, X3, X4, X5, X7, X8\}$.

à l'aide de l'opérateur $AddUndirectedEdge(M_0,\ X)$ et va calculer la mesure d'association ($minAssoc()$) entre $X6$ et chacune des 7 autres variables. Supposons que l'association maximale (Max de $minAssoc()$) soit obtenue pour $X4$ et que $minAssoc(X6, X2) < minAssoc(X6, X7) < minAssoc(X6, X3) <$

$minAssoc(X6, X4)$, alors $X4$ est ajoutée dans $CPC(T)$. A cette étape nous allons mémoriser l'ensemble des $k = 4$ meilleures variables par rapport à la mesure d'association, par exemple $\mathcal{B}0 = \{X4, X3, X7, X2\}$ par ordre décroissant.

L'itération suivante démarre du modèle M_1 contenant variable cible$X6$ et $X4$ qui est ajouté dans la CPC lors de l'itération précédente et génère son voisinage en ajoutant toutes les variables restant un par un (à l'exclusion des variables trouvées indépendant itération précédente). Et répète le même processus.

Ensuite, il calcule pour chaque modèle $minAssoc()$, mais dans cette itération il choisir l'association de minimum (MI) entre conditionnelle et inconditionnelle variables pour chaque modèle e.g. pour le modèle $\{X6, X4, X1\}$ contient une variable cible $X6$, variable CPC $X4$ et une variable de test $X1$, il va calculer $MI(X6, X1)$ et $MI(X6, X1|X4)$, la valeur minimale entre les deux devient le score ($minAssoc()$) de ce modèle.

Il répète ce processus pour tous les modèles dans l'espace de recherche. Enfin, il sélectionne le modèle ayant le score maximum ($MaxMinHeuristic$) et ajoute la variable de test dans l'ensemble de la CPC. Supposons que dans la figure 5.3 donc $X3$ est ajoutée à $CPC\,(T)$.

Nous conservons ici aussi le chemin de recherche (la séquence de variables ajoutées dans la CPC), qui est $Path = \{X4, X3\}$ et les quatre meilleures variables par exemple $\mathcal{B}1 = \{X3, X7, X2, X5\}$. Idem à l'itération suivante où $X7$ est ajoutée et l'ensemble des 4 meilleurs variables est conservée dans $\mathcal{B}2$.

Considérons maintenant qu'il n'y ait plus aucune variable permettant d'améliorer notre mesure d'association. Nous avons donc obtenu à partir de $\mathcal{D}$ un ensemble de CPC pour $X6$ et un ensemble $\mathcal{B}$ résumant les variables caractéristiques à chaque étape de cette recherche.

A l'arrivée de nouvelles données, $\mathcal{D}'$, notre algorithme se remet en marche avec pour objectif d'apprendre un nouvelle modèle pour les données $\mathcal{D} \cup \mathcal{D}'$.

Il repart du modèle initial et vérifier le parcours d'apprentissage précédente mais uniquement aux variables X stockées dans l'ensemble des k meilleures variables à cette itération à l'instant précédent. Ainsi l'espace de recherche est réduit à k

modèles. Supposons que la variable $X4$ soit encore la meilleure, alors elle est ajoutée à $CPC(T)$. Et ainsi de suite le long du chemin de recherche. Si le chemin s'écarte du précédent, alors il faut recalculer l'ensemble des k meilleurs opérateurs pour ce nouveau modèle, et ainsi de suite.

Cet exemple simple nous permet de voir que, dans le meilleur des cas la complexité de la phase de recherche gloutonne locale est contrainte par la valeur k, diminuant ainsi le nombre de mesures d'association à recalculer. Dans le pire des cas, il faudrait alors recalculer un nombre de mesures d'association identique à la situation initiale. Dans des situations plus réalistes où $k \ll n$, ce contrôle de la complexité est un atout important limitant fortement le nombre de mesures d'association à calculer.

B.5 Notre proposition : *MMHC* incrémental

L'objectif de MMHC incrémentale (iMMHC) est d'apprendre une haute qualité de la structure BN progressivement, en réduisant le temps d'apprentissage. Elle peut être obtenue en ré-utilisant les connaissances précédente et la réduction de l'espace de recherche. Comme méthode MMHC, iMMHC est également un algorithme hybride en deux phases, comme décrit dans l'algorithme 10 et illustré à la figure 5.4.

B.5.1 Une approche locale pour l'apprentissage incrémental:

La première phase découvre le squelette possible (graphe non orienté) $\mathcal{G}$ du réseau pour une fenêtre donnée, en utilisant $iMMPC$ méthode (cf. section 5.2.3). Il apprend une structure locale autour de chaque variable en utilisant la connaissance précédente, et évite d'explorer les parties de l'espace de recherche qui étaient auparavant trouvé de faible qualité. A cet effet, il tient une liste des chemins de recherche (*ordre des variables incluses dans l'ensemble des candidats parents-enfants, CPC*). Dans le même temps, il stocke un ensemble de haut *K* meilleures variables qui ont plus de chance d'être considéré dans la CPC. La taille minimale des ensemble *K* dépend du degré moyen du graphe. Lorsque de nouvelles données arrivent, $iMMPC$ vérifie chaque chemin de recherche. Si elles ne sont pas encore validés, il n'y a pas

besoin de ré-apprendre les CPC (modèles locaux). Au contraire, il déclenche le processus de réapprentissage. Le détail de l'algorithme et la description des paramètres sont définis dans la section précédente. Par conséquent, nous construisons les structures locales de façon incrémentielle en réduisant l'espace de recherche, puis nous construisons un graphe non orienté $\mathcal{G}$ (squelette) en fusionnant ces structures locales.

B.5.2 La construction incrémentale d'un réseau bayésien global:

La deuxième phase, une recherche gloutonne est initiée pour l'optimisation globale. Une application naïve de $MMHC$ commencerait cette recherche gloutonne d'un graphe vide. Ici, nous proposons une phase d'optimisation incrémentale par l'initialisation de la procédure de recherche gloutonne par le graphe que l'on obtient dans la fenêtre de temps précédent (comme Shi et Tan, section 4.2.2). La recherche gloutonne considère l'ajout de ces arêtes qui sont nouveaux introduites en skeleton $\mathcal{G}$, mais aussi enlever les arêtes obsolètes. Nous appliquons la opérateurs *add_edge*, *remove_edge* et *reverse_edge*, où *add_edge* peut ajouter un arête $Y \rightarrow X$ si et seulement si Y est un voisin de X dans $\mathcal{G}$. De cette manière, $iMMHC$ garde le sens de l'apprentissage incrémental en considérant le modèle appris précédemment et la révision de la structure existante à la lumière de nouvelles données, comme résumé dans la 5.4.

B.5.3 Optimisation de comptage de fréquence

L'efficacité computationnelle d'algorithmes de l'apprentissage structures du réseau bayésien sont basées sur le nombre de tests nécessaires pour mesurer l'association entre les variables [132]. Ces tests pourraient être des tests d'indépendance conditionnelle ou de calculs d'association de toutes fonction de score. Dans ces tests algorithme avoir à lire de la source de données externe, peut être un fichier de données ou base de données, et de calculer les fréquences pour les certaines variables (tableaux de contingence). Par conséquent, la tâche fastidieuse de la plupart des algorithmes d'apprentissage consiste à calculer les fréquences. En contraste avec les

environnements d'apprentissage batch, en incrémental ou d'apprentissage en ligne le temps d'exécution est plus important, où le modèle doit être mis à jour dans le temps limitées qui sont disponibles.

Il ya plusieurs techniques d'optimisation proposées dans l'article de *MMHC* [138]. Ces techniques améliorent l'efficacité de la phase de recherche locale de l'algorithme. Alors que, dans *MMHC*, les appels pour les scores locaux dans l'optimisation globale (deuxième) phase, sont plus élevés que les appels pour le test de l'indépendance. Par exemple, l'auteur a créé une version carrelée de réseau d'alarme avec environ 5000 variables et 6845 arêtes et échantillonné 5000 exemples de sa distribution. Ensuite, *MMHC* reconstruit ce réseau dans environ 13 jours le temps total. La phase de découverte locale a pris environ 19 heures, et le reste a été consacré à l'optimisation globale (orientation de arête) [138]. Par conséquent, afin d'accélérer la deuxième phase aussi, nous introduisons *compteur de fréquence* entre l'ensemble de données et l'algorithme car:

> *Une grande partie du temps d'apprentissage consiste à recueillir statistiques suffisantes à partir des données[49].*

Compteur de fréquence est une couche intermédiaire entre données et les phases deux de la $iMMHC$ (cf. fig. 5.5). Il est une structure de données de table de hachage maintenue dans la mémoire temporaire comme mémoire cache. Il ya deux phases dans $iMMHC$, dans sa première phase, il calcule les tests de l'indépendance conditionnelle comme: $MI(X, Y|\mathbf{Z})$, où $\mathbf{Z}$ est le sous-ensemble de déjà trouvé CPC et Y est la variable testée pour le CPC. Pour chaque test, il calcule les tableaux de contingence de N_{ijk} pour trouver le MI. Ici, nous remplissons ces fréquences dans une table de hachage pour partager avec de futurs appels.

Dans la phase suivante pour l'orientation de arête, recherche gloutonne calcule les scores locaux (cf. section 3.4.2.1, page 44). Pour les scores locaux, il recalcule les valeurs N_{ijk}. Par conséquent, il se partager le même compteur de fréquence. Il sera tout d'abord chercher dans le cache, si elle trouve les fréquences pour un ensemble de variables nécessaire alors il les rend, sinon il calcule à partir de l'ensemble de données. En même temps, il remplit ces valeurs dans la mémoire cache.

De cette manière, nous pouvons économiser une quantité importante de temps, comme beaucoup de compteur de fréquence appelle N_{ijk} sont partagés avec les deux phases et dans les phases aussi.

B.6 L'adaptation *MMHC* incrémental sur "Sliding Window"

Dans l'environnement de flux de données, il est impossible de stocker des données totale ou attendre l'achèvement de données pour les tâches d'exploration. Il existe différentes techniques utilisées pour stocker le résumé des données vu jusqu'à présent, comme MAP [92] et AD-tree [104]. Mais il est plus compliqué de traiter des données de grande dimension.

Ici, nous proposons un juste milieu entre ces deux approches extrêmes. Donc, nous ne nous appuyons pas complètement sur MAP, ni nous stockons comptes pour toutes les combinaisons possibles de variables comme AD-tree.

L'objectif est de stocker aussi minimum que possible, de statistiques suffisantes à partir des données précédentes, lieu de stocker l'ensemble de flux de données. Par rapport à des environnements d'apprentissage batch, nous nous attendons à un des résultats approximatifs de ces résumés des anciennes données [56].

B.6.1 Mise à jour des statistiques suffisantes

Nous avons déjà discuté de l'adaptation du compteur de fréquence dans l'algorithme $iMMHC$, dans la section 5.3.6. Il était pour la fenêtre de landmark où toute donnée précédente est disponible pour l'apprentissage algorithme et compteur de fréquence utilisée comme cache pour accélérer le processus d'apprentissage. Alors que dans glissement environnement de fenêtre, elle nécessite de stocker résumé des données vieilles comme une statistique suffisante. Par conséquent, à cet effet, nous utilisons actuelle compteur de fréquence FC_i et le modèle actuel BN_i comme un résumé de précédent de données D_i. Le compteur de fréquence FC_i contient toutes N_{ijk} qui

sont utilisées pour produire le modèle au i^{me} fenêtre. Pour la prochaine fenêtre, w_{i+1}, ce compteur de fréquence FC_i sera servi comme FC_{old} (cf. figure 5.6).

$$N_{ijk_{(D)}} = N_{ijk_{(D_{w_i})}} + N_{ijk_{(D_{w_{i-1}})}} \tag{B.1}$$

où

$$N_{ijk_{(D_{w_{i-1}})}} = N_{ijk_{(FC_{old})}} \vee (N \times N_{ijk_{(MAP_{M_{i-1}})}}) \tag{B.2}$$

Les comtes de fréquence à partir des données D sera la somme des fréquences à partir des données actuelles, D_{w_i}, et les fréquences à partir de données précédentes (vu jusqu'à présent), $D_{w_{i-1}}$. Par conséquent, dans l'équation B.1, pour calculer les fréquences de données ensemble, nous exigeons $N_{ijk_{(D_{w_i})}}$ qui peut être calculé directement à partir des données actuelles. Alors que pour $N_{ijk_{(D_{w_i-1})}}$, la plupart des comptes sera trouvé dans FC_{old} (si la distribution des données ne sont pas changé et le nouveau modèle ne sera pas beaucoup différent des anciens). Les autres peut être approximée par l'interrogation d'une distribution conjointe du modèle précédent, MAP_{M_i-1} (cf. section ref sec: 3.3.4, page 40) qui est obtenu à l'étape de l'apprentissage antérieur et en le multipliant par le nombre total d'exemples à $D_{w_{i-1}}$, $N \times N_{ijk_{(MAP_{M_{i-1}})}}$. En dernière analyse, notre approche ne dépend pas entièrement sur le modèle du MAP pour les statistiques suffisantes et de donner un compromis entre deux extrêmes.

B.7 L'adaptation *MMHC* incrémental sur "Damped Window"

Pour garder le modèle à jour avec de nouveaux concepts, il est nécessaire d'éliminer l'effet des exemples de données représentant concept dépassé. Cette question n'a pas été adressée par les chercheurs dans l'apprentissage de la structure du réseau bayésien, sauf Nielsen et Nielsen [109] où ils testent si les nouveaux exemples correspondent au modèle existant ou non.

Oubliant données obsolètes en ajoutant un facteur fondu est une approche couram-

ment utilisée dans l'extraction de flux de données pour adapter les données non stationnaires. Alors que les données récentes ont plus d'importance que vieux (cf. section 2.3.2.1, page 20).

B.7.1 Mécanisme de l'oubli

Dans cette section, nous décrivons un mécanisme d'oubli pour insiste sur l'importance des données récentes. Comme dans le clustering conceptuel, ils utilisent des méthodes de vieillissement pour l'étape-par-étape oublier les vieux concepts en réduisant le poids relatif des échantillons, par exemple le système de FAVORIT [85]. Dans incrémentale filtrage collaboratif [143] aussi, ils utilisent fading facteur à chaque nouvelle session pour que les séances âgés deviennent moins importantes. A cet effet, nous introduisons aussi une fading facteur α au compteur de fréquence conséquent, tous les appels au FC sont pondérées par un facteur fading tels que:

Definition 31. ***(Mécanisme de l'oubli)*** *Soit $N_{ijk_{(D_{w_i})}}$ les comptes de fréquence à partir des données dans la fenêtre courante de w_i et $alpha$ est un facteur fading tels que $0 < \alpha < 1$. Ensuite, le nombre de fréquences, $N_{ijk_{(D)}}$, sur l'ensemble du flux de données peut être obtenu comme:*

$$N_{ijk_{(D)}} = N_{ijk_{(D_{w_i})}} + \alpha[N_{ijk_{(D_{w_i-1})}} + \alpha[... + \alpha[N_{ijk_{(D_{w_0})}}]]]$$

$$= \sum_{w_i=CurrentWindow}^{0} N_{ijk_{(D_{w_i})}} \times \alpha^{w_i-0} \tag{B.3}$$

B.8 Etude expérimentale

B.8.1 Protocole expérimental

Dans cette section, nous discutons sur les benchmarks, des algorithmes et leurs paramètres utilisés dans nos expérimentations. Nous discutons également les mesures qui sont utilisés pour évaluer notre algorithme. Les résultats présentés ici sont la

moyenne et l'écart-type de chaque mesure obtenu sur cinq ensembles de données correspondant à un modèle de référence donnée.

Benchmarks: Pour évaluation de notre approche, nous avons utilisé des benchmarks classique qui sont disponibles sur la plateforme GeNIe/SMILE [2]. Ils sont résumées dans le tableau 6.1.

Pour tester les performances de notre algorithme dans le domaine de grande dimension, nous avons également généré un réseau avec environ un millier de variables en utilisant méthode de carrelage "BN tiling" de Tsamardinos et al [139] (disponible en Explorer Causale [3]). Dans ce but, 28 exemplaires de réseau d'alarme sont carrelés afin que le réseau obtenu *Alarm28* contenant 1036 des variables. Par la suite, nous avons échantillonné cinq ensembles de données à partir de tous ces réseaux bayésiens en utilisant le logiciel GeNIe (pour *Alarm28* utilisant l'explorateur de causalité). Nous avons également choisi cinq jeux de données *Gene* de site Web de $MMHC$ [4].

Les caractéristiques numériques des réseaux sont résumées dans le tableau 6.1. Nous créons un scénario supplémentaire par l'alimentation des données aux algorithmes avec des tailles de fenêtres $\Delta_w = 1000$, 2000 et 3000. Dans cette étude, nous présentons nos résultats pour les fenêtres de taille 2000, sauf pour données *Gene* où nous avons utilisé la taille de la fenêtre 1000.

Algorithmes: Nous avons testé l'algorithme *iMMHC* dans deux scénarios différents, tel que proposé dans la section 5.3:

1. $iMMHC_{\varnothing}$: Démarrage de la phase de recherche gluton à partir d'un graphe vide.
2. $iMMHC_{G}$: Démarrage de la phase de recherche gluton à partir d'un graphe obtenu précédemment.

2. http://genie.sis.pitt.edu/networks.html
3. http://www.dsl-lab.org/causal_explorer/
4. http://www.dsl-lag.org/supplements/mmhc_paper/mmhc_index.html

Nous avons comparé notre algorithme avec batch *MMHC* et avec le $TSearch$ algorithme incrémental comme indiqué dans la section 4.2.2. Nous avons implémenté les algorithmes originaux comme décrit dans leurs articles, en utilisant langage **C++**. Nous avons utilisé bibliothèque de **Boost graph** [5] qui fournit un environnement de manipuler des graphiques. Nous avons également utilisé bayésienne bibliothèque ProBT. Il offre des fonctionnalités liées au réseau bayésiens. Le paquet strucutre apprentissage de réseau bayésien est développé par notre équipe comme une couche supplémentaire pour ProBT. Il fournit les fonctionnalités pour manipuler les réseaux bayésiens et à apprendre de leurs structures en utilisant différents algorithmes.

Pour garder l'harmonie entre ces algorithmes, nous avons utilisé la même heuristique pour tous donc $MMHC$ et iMMHC utilisons également la recherche de glutone à la place de la recherche TABU. A la recherche avide, nous avons utilisé la fonction de score BIC et nous sommes la mise en cache de ces valeurs de score pour éviter les frais généraux re-calcul des appels de fonction de partition similaires. L'indépendance est mesurée à l'aide de textit de d'information mutuelle (MI). Le nombre maximum de variables de conditionnement sont fixés à 10. Le niveau de confiance, $alpha$ ajusté à 0,05 dans nos expériences.

Des expériences ont été réalisées sur un PC dédié avec processeur Intel (R) Xeon (R) W3565 GHz CPU 3.20, architecture 64 bits, 8 Go. Une mémoire RAM et sous Windows 7.

Mesures d'évaluation: Notre premier objectif est de vérifier si l'approche incrémentale de notre algorithme mène à des résultats proches de sa version batch, tout en restreignant le nombre de calculs. Ceci est évalué par d'efficacité de calcul et la précision du modèle. Les résultats présentés dans cette section sont la moyenne et l'écart type de chaque métrique obtenue sur cinq ensembles de données correspondant à un modèle de référence donnée.

Efficacité de calcul La tâche principale dans les algorithmes d'apprentissage est calculer des fonctions de score. La complexité de l'algorithme dépend du

5. http://www.boost.org/

nombre total de des appels de fonction de score, c'est à dire dans notre cas, il est égal à la somme des appels à MI critères d'indépendance et de la fonction de score local appelle lors de la recherche avide. Les appels de fonctions totales sont réduites logarithmique pour une meilleure compréhension. Les barres d'intervalles de confiance sont calculés comme suit:

$$lower\,limit = \log_{10}(average(FunctionCalls) - \sigma(FunctionCalls)) \tag{B.4}$$

$$upper\,limit = \log_{10}(average(FunctionCalls) + \sigma(FunctionCalls)) \tag{B.5}$$

Où σ est un écart-type.

Précision du modèle Nous avons utilisé le "structurel Hamming Distance" (SHD) (cf. section 6.2) pour *précision du modèle*. Il compare la distance entre deux CPDAGs. C'est pourquoi, nous avons transformé la première des structures de réseau apprises et originales dans CPDAGs correspondant (cf. section 3.3.3.2). L'intérêt de cette mesure est qu'elle ne pénalise pas un algorithme pour différences structurelles qui ne peuvent être statistiquement distingué. Il calcule le nombre d'arêtes qui manquent, supplémentaires ou avec de mauvaises directions.

B.8.2 Résultats et interprétations:

B.8.2.1 Analyse des paramètres

Ici, nous analysons le comportement des paramètres définis par l'utilisateur K et taille de la fenêtre w. Nous discutons aussi de l'effet de différentes initialisations pour phase de recherche gloutonne de *iMMHC* algorithme. Tout d'abord, nous analysons l'efficacité du compteur de fréquence cache. Il est utilisé dans toutes les autres expérimentations présentées dans cette étude.

Influence de la fréquence compteur cache: Pour montrer comment compteur de fréquence cache proposé dans la section B.5.3, réduit le temps d'apprentissage

sur le processus incrémental. Nous avons exécuté $iMMHC_{\varnothing}$ avec et sans compteur de fréquence (FC) de cache. Figure 6.1 compare le temps d'exécution en secondes pour le réseau d'alarme. Nous pouvons observer que sans compteur de fréquence les temps d'exécution augmente linéairement sur la fenêtre de landmark. Alors, compteur de fréquence permet d'économiser beaucoup de temps d'exécution et il reste presque constante.

Paramètre *K*: K est un paramètre défini par l'utilisateur pour spécifier l'ensemble des parents-enfants potentiels qui doivent être stockées dans la mémoire à chaque étape de $IMMPC$. Ainsi, il met en cache top *K*.

Le comportement de $iMMHC$ avec différents *K* valeurs est indiquée dans la figure 6.2. Nous pouvons voir que la valeur de K pourrait être entre le degré moyenne et maximale de la courbe théorique. Logiquement, la complexité augmente linéairement par rapport à K. Cette augmentation de la complexité est négligeable en comparaison des appels de fonction au total, alors qu'il est significatif que $iMMHC$ appris un bon modèle même avec une faible valeur de K.

Taille de la fenêtre *w*: Figure 6.3 montre le rôle des différentes tailles de fenêtre sur le processus d'apprentissage progressif de $iMMHC$. Nous pouvons observer que si la taille de la fenêtre est trop grand que 3000 puis l'algorithme va rencontrer beaucoup de changements dans les structures locales, de sorte qu'il nécessite plus d'appels de fonction que dans les petites tailles de fenêtres. En conséquence, il affecte également la précision du modèle. Par conséquent, la fenêtre doit avoir suffisamment volume de données parce que l'algorithme peut trouver un ensemble fiable de variables K, présentant une petite taille de la fenêtre peut négativement affecter le processus d'apprentissage. Nous avons constaté que la taille de fenêtre idéale pour les points de référence utilisés ici est d'environ 2000

Initialisation de recherche gloutonne: Dans la figure 6.4, si l'on compare ces deux initialisations par rapport à la complexité alors $iMMHC_G$ a toujours moins d'appels de fonction que $iMMHC_{\varnothing}$. En conclusion, nous pouvons dire que

l'initialisation de la recherche gloutonne, avec le graphe obtenue précédemment est trouver un compromis entre la précision et de la complexité.

B.8.2.2 iMMHC pour l'apprentissage incrémentale

Figures 6.4 et 6.5 montrent que $iMMHC_G$ surpasse $tsearch$ par rapport à la complexité et la précision. Mais $tsearch$ a une meilleure précision que $iMMHC_\varnothing$ et batch $MMHC$ en *Hailfinder* et il a globalement faible complexité de $iMMHC_\varnothing$ et batch $MMHC$ sauf dans *Pathfinder* et *Link*.

Nous avons remarqué que, pendant l'apprentissage de $Tsearch$, la phase de découverte de squelette ($MWST$) introduit beaucoup de arêtes faux positifs. Par conséquent, ces erreurs propager à la structure finale. Comme cette structure est également utilisé comme une entrée pour la fenêtre de temps suivant, les erreurs locales dans la découverte de squelette tromper le apprentissage de la structure incrémentale. Il est également à augmenter la complexité de la phase d'optimisation globale.

Le $iMMHC$ est un algorithme robuste parce qu'il utilise une adaptation incrémentielle de l'algorithme $MMPC$ pour la phase de découverte squelette qui a été prouvé comme une solution robuste. $MMPC$ limite le nombre de faux positifs.

B.8.2.3 Incrémentale contre MMHC batch

On peut observer sur la figure 6.4 que le modèle obtenu par $iMMHC_G$ a une meilleure qualité que le modèle obtenu par les lots $MMHC$, sauf pour les jeux de données avec hauts cardinalités (*Barley* et *Pathfinder*), alors que dans *Link* dataset algorithme de batch aussi donne une précision intéressante. Considérant que, le $iMMHC_\varnothing$ obtenu même précision globale comme $MMHC$ batch avec une faible complexité. Ces résultats sont cohérents avec le travail de Roure dans [121]. En ce qui concerne la dimension élevée de l'espace de recherche, des algorithmes supplémentaires peut éviter d'être pris au piège dans une certaine optima locaux que pourraient être leurs contreparties batch.

B.8.2.4 iMMHC pour les domaines de grande dimension

Nous avons vu dans le chapitre 4 que l'apprentissage de la structure du réseau Bayésienne incrémentale a été testé avec benchmarks d'environ 37 variables. Nous avons utilisé ici deux benchmarks; *Alarm28* avec 1028 variables et *Gene* a 801 variables, afin de tester la capacité de notre algorithme pour traiter avec des domaines de grande dimension. Nous pouvons observer que les résultats de $iMMHC_G$ en figures 6.4 et 6.5 sont beaucoup mieux que d'autres. Nous pouvons donc dire $iMMHC$ est une bonne solution pour apprentissage incrémental dans les domaines de grande dimension.

B.8.2.5 Résultats et interprétations avec "sliding window"

Figure 6.6 compare la précision (SHD) obtenu avec des "sliding window" et "landmark window" pour les ensembles de données *Alarme*, *Hailfinder* et *Alarm28*. nous pouvons observer que l'utilisation de "sliding window" $iMMHC_G$ algorithme pourrait pas obtenir la même précision que sur "landmark window". Il est normal comme discuté dans la section 2.2.1 sur le traitement des flux de données que les résultats obtenus au cours de "sliding window" sont approchées. Dans la "landmark window" les algorithmes ont leur accès aux données entiers vu précédemment. Mais dans la "sliding window" ils ont seulement des statistiques suffisantes approchées comme un résumé des données précédentes.

Figure 6.7 montre le gain de temps en pourcentage dans la "sliding window". Il est calculé comme $(1 - T_{SW}/T_{LW}) \times 100$, où T_{SW} et T_{LW} sont les temps d'exécution pour $iMMHC_G$ algorithme plus "sliding window" et "landmark window" respectivement. Nous pouvons observer que "sliding window" permet d'économiser beaucoup de temps d'exécution par rapport à la "landmark window" pour les ensembles de données ayant petit nombre de variables et des réseaux simples.

D'autre part, comme indiqué dans la section 3.3.4 concernant la complexité des requêtes probabilistes pour les ensembles de données ayant un grand nombre de variables et des réseaux complexes donc il prend plus de temps que la fenêtre repère, comme on le voit dans la figure 6.7 pour *Alarm28* ensemble de données. Il est plus

clair dans la première fenêtre de *Alarm28* alors que dans les fenêtres réussir, il essaie de récupérer ce temps que l'algorithme a de plus en plus de données. Nous utilisons une technique d'inférence approximative en utilisant la méthode d'échantillonnage pour les requêtes probabilistes de grands réseaux. La taille de l'échantillon été ajusté à 500.

B.8.2.6 Résultats et interprétations avec"damped window"

Domaines non-stationnaires: Nous avons créé un environnement non stationnaire en ajoutant deux ensembles de données de deux réseaux différents comme: nous avons généré des ensembles de données de 10.000 cas de réseaux originaux, *Alarm* et *Hailfinder* notée par lettre *A*. Puis changé les réseaux originaux environ 15% à 21% en supprimant certains arêtes (10 sur 46 à *Alarm* et 10 sur 66 à *Hailfinder*) et généré des ensembles de données de 10.000 cas de la réseaux disant *B* résultant. Plus tard, nous avons fusionné ces deux ensembles de données comme *AlarmAB* et *HailfinderAB* que les données générées à partir du réseau *A* est placé avant les données générées par le réseau *B*. Nous avons échantillonné les cinq ensembles de données de cette manière. Nous avons calculé le SHD en comparant les résultats de chaque fenêtre avec leurs réseaux respectifs d'origine.

Puis nous avons appliqué $iMMHC_G$ algorithme utilisant "sliding window" et "damped window". Nous avons également appliqué algorithme $MMHC$ batch pour chaque réseau. Pour "damped window" nous avons ajusté fondu facteur α à $0,5$.

De la figure 6.8, nous pouvons observer que pour les "sliding window", les SHD augmente lorsque drift se produit à la cinquième fenêtre (nombre de cas = 10 000), parce que le modèle précédent utilisé pour estimer les statistiques suffisantes pour les données passées, qui ne sont pas pertinents, donc le SHD augmente et même après la drift, il ne pouvait pas récupéré. Alors avec "damped window" (DW 0.5) SHD augmente au début parce que nous appliquons facteur d'oubli, mais après le facteur d'oubli de drift aide à corriger et il commence à diminuer. Il est plus visible pour les *AlarmAB* que pour *HailfinderAB* ensembles de données. Par conséquent, facteur d'oubli contribue à jeter les anciennes données et de traiter avec des domaines

non-stationnaires.

Domaines stationnaires: Pour tester $iMMHC_G$ algorithme pour domaines stationnaires sur "damped window", nous avons généré des deux ensembles de données de 10 000 cas de réseaux *Alarm* et *Hailfinder* puis les fusionner en un seul ensemble de données contenant 20 000 enregistrements de données, *AlarmAA* et *HailfinderAA*. Plus tard, nous avons appliqué $iMMHC_G$ algorithme de "damped window" avec fading facteur $0,5$ et pour "sliding window".

Les résultats de cette section sont présentés dans la figure 6.9. Nous pouvons observer que SHD augmenté dans le début de la fenêtre amortie (DW 0.5), mais facteur d'oubli plus tard aide à récupérer et il a diminué.

Comme vu précédemment, "damped window" amorties aide pour traiter des domaines non-stationnaires. Si le domaine est constante (stationnarité), en utilisant des "damped window" génère une diminution de la durée de précision pendant le début, mais la différence est plus significative après quelques itérations.

B.9 Conclusion

Dans ce travail, nous avons proposé une version incrémentale de l'algorithme de recherche locale *MMPC*. L'utilisation de deux heuristiques proposées initialement pour des algorithmes d'apprentissage à base de score permet de réduire fortement l'espace de recherche lors de l'arrivée de nouvelles données en ne considérant que les plus fortes dépendances découvertes lors de l'étape précédente. Une étude théorique et résultats expérimentaux montrent que cet algorithme incrémental permet effectivement de réduire considérablement le temps de calcul en arrivant à un modèle équivalent.

De même que l'algorithme *MMPC(T)* n'est qu'une étape dans la construction d'un réseau bayésien global, la prochaine étape de notre travail i*MMHC* utilise de notre algorithme i*MMPC(T)* dans la construction incrémentale d'un réseau bayésien.

Nous avons introduit un compteur de fréquence qui fonctionne comme couche

intermédiaire entre les données et l'algorithme. Il sert comme un cache pour accélérer le processus d'apprentissage. Ensuite, nous avons adapté $iMMHC$ pour fenêtre coulissante en stockant statistiques suffisantes. Enfin, pour réagir aux tendances récentes dans les données, nous avons proposé un mécanisme oublier au compteur de fréquence. Le principal objectif de notre démarche est de passer moins de temps que d'un scénario d'apprentissage de lot et de stocker quantité limitée d'informations qui sont utiles pour le processus d'apprentissage progressif.

Etude théorique et expériences préliminaires montrent notre approche améliore les performances de l'algorithme systématiquement, réduisant considérablement la complexité.

Abbreviations & Acronyms

AIC	Akaike information criterion
BN	Bayesian network
BIC	Bayesian information criterion
CPC	Candidate parent and children
CPDAG	Completed partially directed acyclic graph
DAG	Directed acyclic graph
FC	Frequency counter
GS	Greedy search
HCS	Hill-climbing search
iHCS	Incremental hill-climbing search
iMMPC	Incremental max-min parent children
iMMHC	Incremental max-min hill-climbing
MAP	Maximum a-posteriori probability
MDL	Minimum description length
MMHC	Max-min hill-climbing
MWST	Maximum weight spanning tree
MMPC	Max-min parent children
PDAG	Partially directed acyclic graph
PGM	Probabilistic graphical model
RSS	Reduced search space
TOCO	Traversal operators in correct order

List of Tables

2.1 *Differences between traditional and stream data processing [56]* . . 15

3.1 *Number of possible DAG patterns generated by n variables [122].* . 44

6.1 *Name of the benchmark, number of variables and edges, cardinality and degree of the each graph used for our experiments.* 116

A.1 *Comparison of data stream mining techniques for Classification [80]* .141

A.2 *Comparison of data stream mining techniques for clustering [80]* . . 143

A.3 *Comparison of data stream mining techniques for frequency counting and Time Series Analysis [80].* 143

List of Figures

1.1 *Scope of the thesis* 7

2.1 *An Example of drift in classification with recently observed data examples in different time windows and the associated decision boundaries.* 16

2.2 *Some change types in data stream* 16

2.3 *Classification of data stream processing methods (adapted from [80]).* 17

2.4 *Horizontal and vertical summarizing techniques for a data stream.* . 18

2.5 *Different window models* 22

3.1 *Conditional independences and factorization of the joint distribution by probabilistic graphical models: first row - graphical representation; second row - independences induced by the structure of graph; third row - factorization driven from the structure of graph. (a) an example of DAG. (b) an example of Markov random field (MRF) [5].* 37

3.2 *An example of Bayesian network model; A DAG with five nodes and parameters representing the probability distribution.* 38

3.3 *An example of Markov equivalent class for three variables,* X, Y, Z *and four DAGs . First three DAGs (a), (b) and (c) have the same independence structure and (d) corresponds to another set of independencies.* 40

3.4 *Outline of greedy/hill-climbing search* 48

3.5 *Outline of* $\overline{MMPC}(T)$ *forward phase* 52

4.1 *Outline of TOCO heuristic* 72
4.2 *Outline of RSS heuristic* 73
4.3 *Two BNs from which a DAG faithful sequence has been sampled* [109]. 82
4.4 *Learned and extracted graph fragments matching* $\mathcal{B}_2$ [109]. 82
4.5 *(a) The merged graph fragment and (b) the fully directed version* [109]. 82

5.1 *Outline of* $iMMPC$ *algorithm* 92
5.2 *Outline of* $\overline{iMMPC}(T)$ *algorithm (forward phase) for incremental data sets D and DuD'* 96
5.3 *An illustrative example for* $\overline{iMMPC}(T)$ *(forward phase) algorithm* 98
5.4 *iMMHC outline : dependencies between iMMHC execution for window* w_i *and previous results.* 102
5.5 *Frequency counter "FC" outline in* $iMMHC$ *algorithm* 105
5.6 *Using an old frequency counter "FC" and BN as a summary of old data for each window "w".* 108

6.1 *Comparison of* $iMMHC$ *execution time with and without frequency counter for Alarm on landmark window.* 118
6.2 *Performance of* $iMMHC$ *algorithm for different K values (Function calls are divided by* 10^2 *for "Barley" and* 10^5 *for "Alarm28", and SHD by* 10^1 *for "Alarm28" for ease of exposition)* 119
6.3 *Performance of* $iMMHC$ *algorithm for windows size 1000, 2000 and 3000 (average and standard deviation for final models)* 119
6.4 *Comparisons of Structural Hamming distance (SHD) for* $MMHC$, $iMMHC_{\varnothing}$, $iMMHC_G$ *and* $TSearch$ *algorithms using window of size 2000 (1000 for "Gene")* 121
6.5 *Comparisons of Log10(total function calls) for* $MMHC$, $iMMHC_{\varnothing}$, $iMMHC_G$ *and* $TSearch$ *algorithms over an incremental process (using window of size 2000 (1000 for "Gene"))* 122
6.6 *Structural Hamming Distance (SHD) over sliding window.* 124
6.7 $iMMHC_G$ *execution time gain over sliding window.* 125

6.8 *Comparison of Structural Hamming Distance (SHD) for non-stationary domains over sliding, landmark and damped windows.* 126

6.9 *Comparison of Structural Hamming Distance (SHD) for stationary domains over sliding and damped windows.* 127

List of Algorithms

1 *Greedy Search (GS)* 48

2 $MMHC(D)$ 50

3 $MMPC(T, D)$ 51

4 $\overline{MMPC}(T)$ 52

5 *Friedman and Goldszmidt's algorithm* 67

6 *Incremental Hill Climbing Search ($iHCS$)* 71

7 *Shi and Tan's algorithm* 76

8 $iMMPC(D_w)$ 91

9 *Incremental $\overline{MMPC}$ Algorithm* 95

10 $iMMHC(D_w)$ 102

Bibliography

[1] Charu C. Aggarwal. *Data streams: models and algorithms*, volume 31. Springer, 2007. 14

[2] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In *VLDB '2003: Proceedings of the 29th international conference on very large data bases*, pages 81–92. VLDB Endowment, 2003. 25, 142

[3] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. On demand classification of data streams. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge Discovery and Data mining*, KDD '04, pages 503–508, New York, NY, USA, 2004. ACM. 141

[4] Hirotugu Akaike. Statistical predictor identification. *Annals of the Institute of Statistical Mathematics*, 22(1):203–217, 1970. 46

[5] Sourour Ammar. *Probabilistic graphical models for density estimation in high dimensional spaces: application of the Perturb & Combine principle with tree mixtures*. PhD thesis, Université de Nantes, December 2010. 37, 137

[6] Brigham Anderson and Andrew Moore. AD-Trees for fast counting and for fast learning of association rules. In *Proceedings Fourth International Conference on Knowledge Discovery and Data Mining*, pages 134–138. ACM Press, 1998. 106

[7] Arvind Arasu, Mitch Cherniack, Eduardo Galvez, David Maier, Anurag S. Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. Linear road: a stream data management benchmark. In *Proceedings of the Thirtieth*

international conference on very large data bases - Volume 30, VLDB '04, pages 480–491. VLDB Endowment, 2004. 14

[8] Brain Babcock, Mayur Datar, Rajeev Motwani, and Liadan O'Callaghan. Maintaining variance and k-medians over data stream windows. In *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 234–243, New York, NY, USA, 2003. ACM. 25

[9] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16, New York, NY, USA, 2002. ACM. 14, 19, 20, 28

[10] Brian Babcock, Mayur Datar, and Rajeev Motwani. Load shedding techniques for data stream systems. In *Proceedings of the 2003 Workshop on Management and Processing of Data Streams (MPDS)*, volume 577, 2003. 20

[11] James Bailey and Elsa Loekito. Efficient incremental mining of contrast patterns in changing data. *Information processing letters*, 110(3):88–92, 2010. 28

[12] Yang Bei and Huang Houkuan. TOPSIL-miner: an efficient algorithm for mining top-k significant itemsets over data streams. *Knowledge and Information Systems*, 23(2):225–242, 2010. 28

[13] Yael Ben-Haim and Elad Tom-Tov. A streaming parallel decision tree algorithm. volume 11, pages 849–872. JMLR. org, 2010. 26

[14] Albert Bifet and Ricard Gavaldà. Adaptive learning from evolving data streams. In *IDA '09: Proceedings of the 8th International Symposium on Intelligent Data Analysis*, pages 249–260, Berlin, Heidelberg, 2009. Springer-Verlag. 26

[15] Wray Buntine. Theory refinement on Bayesian networks. In *Proceedings of the seventh conference (1991) on Uncertainty in artificial intelligence*, pages

52–60, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc. 57, 70, 106, 148

[16] Gladys Castillo and João Gama. Adaptive Bayesian network classifiers. *Intelligent Data Analysis*, 13:39–59, January 2009. 149

[17] Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 30–39, New York, NY, USA, 2003. ACM. 25

[18] R. Chen, K. Sivakumar, and H. Kargupta. An approach to online Bayesian learning from multiple data streams. In *Proceedings of Workshop on Mobile and Distributed Data Mining, PKDD*, volume 1, pages 31–45. Citeseer, 2001. 137

[19] Yixin Chen and Li Tu. Density-based clustering for real-time stream data. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge Discovery and Data mining*, KDD '07, pages 133–142, New York, NY, USA, 2007. ACM. 142

[20] Jian Cheng and Marek J. Druzdzel. AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks. *Journal of Artificial Intelligence Research*, 13(1):155–188, 2000. 42

[21] Yun Chi, Haixun Wang, and Philip S. Yu. Loadstar: load shedding in data stream mining. In *VLDB '05: Proceedings of the 31st international conference on very large data bases*, pages 1302–1305. VLDB Endowment, 2005. 20, 140, 143

[22] David M. Chickering. Learning Bayesian networks is NP-complete. In *Proceedings of AI and Statistics, 1995.*, pages 121–130, 1995. 44, 89, 146

[23] David M. Chickering. A transformational characterization of equivalent Bayesian network structures. In *Proceedings of the 11th Annual Conference on Uncertainty in Artificial Intelligence (UAI-95)*, pages 87–98, San Francisco, CA., 1995. Morgan Kaufmann. 47

[24] David M. Chickering. Learning equivalence classes of Bayesian-network structures. *Journal of Machine Learning Research*, 2:445–498, February 2002. 40

[25] David M. Chickering and Craig Boutilier. Optimal structure identification with greedy search. *Journal of Machine Learning Research*, 3:507–554, 2002. 45, 47

[26] David M. Chickering, D. Geiger, and D. Heckerman. Learning Bayesian networks: Search methods and experimental results. In *Proceedings of Fifth Conference on Artificial Intelligence and Statistics*, pages 112–128, 1995. 147, 150

[27] David M. Chickering and David Heckerman. Efficient approximations for the marginal likelihood of incomplete data given a Bayesian network. In *Proceedings of the twelfth international conference on uncertainty in artificial intelligence*, pages 158–168. Morgan Kaufmann Publishers Inc., 1996. 45

[28] Hyun Cheol Cho, Kwon Soon Lee, and M Sami Fadali. Online learning algorithm of dynamic Bayesian networks for nonstationary signal processing. *International Journal of Innovative Computing, Information and Control*, 5(4):1027–1042, 2009. 137

[29] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968. 45, 46, 70, 77

[30] Antonio S. Cofiño, Rafael Cano, Carmen Sordo, and Jose M. Gutierrez. Bayesian networks for probabilistic weather prediction. In *15th Eureopean Conference on Artificial Intelligence, ECAI*, pages 695–700. Citeseer, 2002. 5

[31] Gregory F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial intelligence*, 42(2):393–405, 1990. 41, 42

[32] Gregory F. Cooper and Edward Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347, 1992. 45, 46, 57, 58, 70

[33] Graham Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. *ACM Trans. Database Syst.*, 30(1):249–278, 2005. 23, 27

[34] Corinna Cortes, Kathleen Fisher, Daryl Pregibon, and Anne Rogers. Hancock: a language for extracting signatures from data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge Discovery and Data mining*, KDD '00, pages 9–17, New York, NY, USA, 2000. ACM. 14

[35] Rónán Daly, Qiang Shen, and Stuart Aitken. Review: learning Bayesian networks: Approaches and issues. *The Knowledge Engineering Review*, 26(2):99–157, 2011. 42, 89

[36] Adnan Darwiche. Bayesian networks. *Commun. ACM*, 53(12):80–90, December 2010. 4

[37] Luis M. de Campos. A scoring function for learning Bayesian networks based on mutual information and conditional independence tests. *J. Mach. Learn. Res.*, 7:2149–2187, December 2006.

[38] Sérgio Rodrigues De Morais and Alexandre Aussem. An efficient and scalable algorithm for local Bayesian network structure discovery. In *Machine Learning and Knowledge Discovery in Databases*, pages 164–179. Springer, 2010. 49, 50

[39] Francisco Javier Diez. Parameter adjustment in Bayes networks. the generalized noisy or-gate. In *Proceedings of the Ninth international conference on Uncertainty in artificial intelligence*, pages 99–105. Morgan Kaufmann Publishers Inc., 1993. 56

[40] Jianguo Ding. Probabilistic inferences in Bayesian networks. *arXiv preprint arXiv:1011.0935*, 2010. 41

[41] Alin Dobra, Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Processing complex aggregate queries over data streams. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 61–72, New York, NY, USA, 2002. ACM. 19

[42] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge Discovery and Data mining*, pages 71–80, New York, NY, USA, 2000. ACM. 16, 25, 26, 142

[43] Pedro Domingos and Geoff Hulten. A general method for scaling up machine learning algorithms and its application to clustering. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 106–113. Morgan Kaufmann, 2001. 19, 25, 142

[44] Kazuo J. Ezawa and Til Schuermann. Fraud/uncollectible debt detection using a Bayesian network based learning system: A rare binary outcome with mixed data structures. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 157–166. Morgan Kaufmann Publishers Inc., 1995. 5

[45] M.T. Fennell and Richard P. Wishner. Battlefield awareness via synergistic sar and mti exploitation. *Aerospace and Electronic Systems Magazine, IEEE*, 13(2):39–43, 1998. 5

[46] Francisco Ferrer-Troyano, Jesús S. Aguilar-Ruiz, and José C. Riquelme. Discovering decision rules from numerical data streams. In *Proceedings of the 2004 ACM symposium on Applied computing*, SAC '04, pages 649–653, New York, NY, USA, 2004. ACM. 141

[47] Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *IJCAI*, pages 1300–1309. Springer-Verlag, 1999. 137

[48] Nir Friedman and Moises Goldszmidt. Sequential update of Bayesian network structure. In *Proceedings of the 13th Conference on Uncertainty in Artificial*

Intelligence (UAI 97), pages 165–174, 1997. 64, 106, 148

[49] Nir Friedman, Iftach Nachman, and Dana Peér. Learning Bayesian network structure from massive datasets: The "sparse candidate" algorithm. In *Proceedings of the 15th Conference on Uncertainty in Artificail Intelligence(UAI-99)*, pages 206–215, 1999. 50, 53, 104, 159

[50] Robert M. Fung and Kuo-Chu Chang. Weighing and Integrating Evidence for Stochastic Simulation in Bayesian Networks. In *Proceedings of the Fifth Annual Conference on Uncertainty in Artificial Intelligence*, UAI '89, pages 209–220, Amsterdam, The Netherlands, 1990. North-Holland Publishing Co. 42

[51] Mohamed Medhat Gaber, Shonali Krishnaswamy, and Arkady Zaslavsky. On-board mining of data streams in sensor networks. In *Advanced Methods for Knowledge Discovery from Complex Data*, Advanced Information and Knowledge Processing, pages 307–335. Springer London, 2005. 23, 140

[52] Mohamed Medhat Gaber, Shonali Krishnaswamy, and Arkady B. Zaslavsky. Resource-aware mining of data streams. *j-jucs*, 11(8):1440–1453, 2005. 20, 23

[53] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. Towards an adaptive approach for mining data streams in resource constrained environments. In Yahiko Kambayashi and Wolfram Mohania, Mukesh, editors, *Data Warehousing and Knowledge Discovery*, volume 3181 of *Lecture Notes in Computer Science*, pages 189–198. Springer Berlin Heidelberg, 2004. 23

[54] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. Mining data streams: a review. *SIGMOD Rec.*, 34(2):18–26, 2005. 16, 17, 19

[55] João Gama. *Knowledge Discovery from Data Streams*. CRC Press, May 2010. 5, 146

[56] João Gama and Mohamed Gaber. *Learning from Data Streams – Processing techniques in Sensor Networks*. Springer, 2007. 14, 15, 107, 160

[57] João Gama, Pedro Medas, Gladys Castillo, and Pedro Pereira Rodrigues. Learning with drift detection. In *SBIA*, pages 286–295, 2004. 16

[58] Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. Mining data streams under block evolution. *SIGKDD Explor. Newsl.*, 3(2):1–10, January 2002. 140

[59] Olga Georgieva and Dimitar Filev. Gustafson-kessel algorithm for evolving data stream clustering. In *CompSysTech '09: Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*, pages 1–6, New York, NY, USA, 2009. ACM. 25

[60] Lise Getoor and Mehran Sahami. Using Probabilistic Relational Models for Collaborative Filtering. In *Workshop on Web Usage Analysis and User Profiling (WEBKDD'99)*, pages 1–6, 1999. 137

[61] Chris Giannella, Jiawei Han, Jian Pei, Xifeng Yan, and Philip S. Yu. Mining frequent patterns in data streams at multiple time granularities. In *Data Mining: Next Generation Challenges and Future Directions, AAAI/MIT, H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.)*, 2002. 27, 143

[62] Anna C. Gilbert, Sudipto Guha, Piotr Indyk, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 389–398, New York, NY, USA, 2002. ACM. 20

[63] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 15:515–528, 2003. 25

[64] Sudipto Guha, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams. In *Proceedings of the Annual Symposium on Foundations of Computer Science.*, pages 359–366. IEEE, 2000. 19, 25

[65] Haipeng Guo and William Hsu. A survey of algorithms for real-time Bayesian network inference. In *AAAI/KDD/UAI02 Joint Workshop on Real-Time Deci-*

sion Support and Diagnosis Systems, pages 1–12. Edmonton, Canada, 2002. 41, 42

[66] Georges Hébrail. Statistical challenges in data stream applications. Technical report, 56th Session of the International Statistical Institute, Lisbonne, August 2007. 18

[67] Georges Hébrail. *Data stream management and mining*, chapter Data stream management and mining, pages 89–102. IOS Press, Amsterdam, The Netherlands, 2008. 19

[68] David Heckerman, Dan Geiger, and David M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995. 10.1007/BF00994016. 43

[69] Max Henrion. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. *Uncertainty in Artificial Intelligence*, 2:317–324, 1988. 42

[70] Avi Herscovici and Oliver Brock. Improving high-dimensional Bayesian network structure learning by exploiting search space information. Technical report, Department of Computer Science, University of Massachusetts Amherst, 2006. 53

[71] Stefan Hoeglinger, Russel Pears, and Yun Sing Koh. Cbdt: A concept based approach to data stream mining. In *PAKDD '09: Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pages 1006–1012, Berlin, Heidelberg, 2009. Springer-Verlag. 26

[72] Eric Horvitz, Jack Breese, David Heckerman, David Hovel, and Koos Rommelse. The lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 256–265. Morgan Kaufmann Publishers Inc., 1998. 5

[73] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *KDD '01: Proceedings of the seventh ACM SIGKDD interna-*

tional conference on Knowledge Discovery and Data mining, pages 97–106, New York, NY, USA, 2001. ACM. 16, 26, 142

[74] Kyu Baek Hwang, Jae Won Lee, Seung-Woo Chung, and Byoung-Tak Zhang. Construction of large-scale Bayesian networks by local to global search. In *Proceedings of the 7th Pacific Rim International Conference on Artificial Intelligence: Trends in Artificial Intelligence*, PRICAI '02, pages 375–384, London, UK, 2002. Springer-Verlag. 53

[75] Elena Ikonomovska and Sašo Džeroski. Regression on evolving multi-relational data streams. In *Proceedings of the 2011 Joint EDBT/ICDT Ph.D. Workshop*, PhD '11, pages 1–7, New York, NY, USA, 2011. ACM. 137

[76] Elena Ikonomovska, Suzana Loskovska, and Dejan Gjorgjevik. A survey of stream data mining. In *Proceedings of 8th National Conference with International Participation, ETAI*, pages 19–21, 2007. 14, 19

[77] iPolicy Networks. http://www.ipolicynetworks.com. 28

[78] Finn Verner Jensen, Bo Chamberlain, Torsten Nordahl, and Frank Jensen. Analysis in hugin of data conflict. In *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*, UAI '90, pages 519–528, New York, NY, USA, 1991. Elsevier Science Inc. 79

[79] Michael Irwin Jordan. *Learning in Graphical Models*. MIT Press, 1998. 4

[80] Mahnoosh Kholghi and Mohammadreza Keyvanpour. An analytical framework for data stream mining techniques based on challenges and requirements. *International Journal of Engineering Science and Technology (IJEST)*, Vol. 3 No. 3:1–7, 2011. 17, 141, 143

[81] Younghee Kim and Ungmo Kim. WSFI-mine: Mining frequent patterns in data streams. In *ISNN 2009: Proceedings of the 6th International Symposium on Neural Networks*, pages 845–852, Berlin, Heidelberg, 2009. Springer-Verlag. 28

[82] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009. 41

[83] Paul Komarek and Andrew W. Moore. A dynamic adaptation of AD-trees for efficient machine learning on large data sets. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 495–502, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. 106

[84] X.Z. Kong, Y.X. Bi, and D.H. Glass. A geometric moving average martingale method for detecting changes in data streams. In *Research and Development in Intelligent Systems XXIX*, pages 79–92. Springer, 2012. 16

[85] Miroslav Kubat and Ivana Krizakova. Forgetting and aging of knowledge in concept formation. *Applied Artificial Intelligence an International Journal*, 6(2):195–206, 1992. 21, 109, 162

[86] Solomon Kullback. *Information theory and statistics*. Courier Dover Publications, 1997. 49

[87] Venu Madhav Kuthadi, A. Govardhan, and P. Prem Chand. Incremental learning algorithm for dynamic data streams. In *IJCSNS International Journal of Computer Science and Network Security*, volume 8, pages 134–138, 2008. 26

[88] YongChul Kwon, Wing Yee Lee, Magdalena Balazinska, and Guiping Xu. Clustering events on streams using complex context information. In *Proceedings of the 2008 IEEE International Conference on Data Mining Workshops*, ICDMW '08, pages 238–247, Washington, DC, USA, 2008. IEEE Computer Society. 140

[89] Gerard Lacey and Shane MacNamara. Context-aware shared control of a robot mobility aid for the elderly blind. *I. J. Robotic Res.*, 19(11):1054–1065, 2000. 5

[90] Wai Lam. Bayesian network refinement via machine learning approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):240–251, 1998. 148

[91] Wai Lam and Fahiem Bacchus. Learning Bayesian Belief Networks: An Approach Based on the MDL Principle. *Computational Intelligence*, 10(3):269–293, 1994. 60, 61, 62

[92] Wai Lam and Fahiem Bacchus. Using new data to refine a Bayesian network. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 383–390, 1994. 60, 62, 106, 148, 160

[93] Mark Last. Online classification of nonstationary data streams. *Intelligent Data Analysis*, 6(2):129–147, 2002. 26, 140

[94] Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 157–224, 1988. 41

[95] Yan-Nei Law and Carlo Zaniolo. An adaptive nearest neighbor classification algorithm for data streams. In *Proceedings of the 9th European conference on Principles and Practice of Knowledge Discovery in Databases*, PKDD'05, pages 108–120, Berlin, Heidelberg, 2005. Springer-Verlag. 141

[96] Matthijs Leeuwen and Arno Siebes. Streamkrimp: Detecting change in data streams. In *Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases - Part I*, ECML PKDD '08, pages 672–687, Berlin, Heidelberg, 2008. Springer-Verlag. 16

[97] Philippe Leray and Olivier François. Bayesian network structural learning and incomplete data. In *Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR 2005)*, pages 33–40, Espoo, Finland, 2005. 43

[98] Hua-fu Li, Suh-yin Lee, and Man-kwan Shan. An efficient algorithm for mining frequent itemsets over the entire history of data streams. In *Proceedings*

of the First International Workshop on Knowledge Discovery in Data Streams, pages 1–10, 2004. 28

[99] Peipei Li, Xuegang Hu, and Xindong Wu. Mining concept-drifting data streams with multiple semi-random decision trees. In *ADMA '08: Proceedings of the 4th international conference on Advanced Data Mining and Applications*, pages 733–740, Berlin, Heidelberg, 2008. Springer-Verlag. 26

[100] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *VLDB '02: Proceedings of the 28th international conference on very large data bases*, pages 346–357, 2002. 27, 143

[101] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 251–262, New York, NY, USA, 1999. ACM. 20

[102] Alice Marascu and Florent Masseglia. Mining sequential patterns from data streams: a centroid approach. *Journal of Intelligent Information Systems*, 27(3):291–307, 2006. 27

[103] Alice Marascu and Florent Masseglia. Atypicity detection in data streams: A self-adjusting approach. *Intelligent Data Analysis*, 15(1):89–105, January 2011. 16

[104] Andrew Moore and Mary Soon Lee. Cached suficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research*, 8:67–91, 1998. 106, 160

[105] Subbaratnam Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers Inc, 2005. 19, 23

[106] Patrick Naïm, Pierre-Henri Wuillemin, Philippe Leray, Olivier Pourret, and Anna Becker. *Réseaux Bayésiens*. Eyrolles, Paris, 3 edition, 2007. 49

[107] Richard E. Neapolitan. *Learning Bayesian Networks*. Pearson Prentice Hall Upper Saddle River, 2004. 77

[108] Hoai-Tuong Nguyen. *Réseaux Bayésiens et apprentissage ensembliste pour l'étude différentielle de réseaux de régulation génétique*. PhD thesis, Université de Nantes, January 2012. 53

[109] Søren Holbech Nielsen and Thomas D. Nielsen. Adapting Bayes network structures to non-stationary domains. *Int. J. Approx. Reasoning*, 49(2):379–397, 2008. 79, 82, 109, 149, 161

[110] Liadan O'Callaghan, Nina Mishra, Adam Meyerson, Sudipto Guha, and Rajeev Motwani. Streaming-data algorithms for high-quality clustering. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE '02, pages 685–694, Washington, DC, USA, 2002. IEEE Computer Society. 142

[111] Kristian G. Olesen, Steffen L. Lauritzen, and Finn V. Jensen. aHUGIN: A system creating adaptive causal probabilistic networks. In *Proceedings of the Eighth international conference on Uncertainty in artificial intelligence*, pages 223–229. Morgan Kaufmann Publishers Inc., 1992. 56

[112] Carlos Ordonez. Clustering binary data streams with k-means. In *8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 12–19. ACM Press, 2003. 25

[113] Spiros Papadimitriou, Anthony Brockwell, and Christos Faloutsos. Adaptive, hands-off stream mining. In *Proceedings of the 29th international conference on very large data bases - Volume 29*, VLDB '03, pages 560–571. VLDB Endowment, 2003. 142

[114] Judea Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. In *Proceedings of the 7th Conference of the Cognitive Science Society, University of California, Irvine*, pages 329–334, August 1985. 41

[115] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988. 39

[116] Hanchuan Peng, Fuhui Long, and Chris Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1226–1238, 2005. 78

[117] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C++: The Art of Scientific Computing, Second Edition*. Cambridge University Press, 2nd edition, October 2002. 80, 81

[118] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978. 46, 60

[119] Robert W. Robinson. Counting unlabeled acyclic digraphs. In *Combinatorial mathematics V*, pages 28–43. Springer, 1977. 44

[120] Sergio Rodrigues De Morais and Alex Aussem. A novel scalable and data efficient feature subset selection algorithm. In *Proceedings of the European conference on Machine Learning and Knowledge Discovery in Databases - Part II*, ECML PKDD '08, pages 298–312, Berlin, Heidelberg, 2008. Springer-Verlag. 147

[121] Josep Roure Alcobé. Incremental hill-climbing search applied to Bayesian network structure learning. In *First International Workshop on Knowledge Discovery in Data Streams. (KDDS-ECML)*, pages 1–10, 2004. 57, 69, 100, 106, 123, 148, 150, 167

[122] Josep Roure Alcobé. *Incremental Methods for Bayesian Network Structure Learning*. PhD thesis, Universitat Politècnica de Catalunya, 2004. 44, 47

[123] Josep Roure Alcobé and Andrew W. Moore. Sequential update of ADtrees. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 769–776, New York, NY, USA, 2006. 75, 106

[124] Josep Roure Alcobé and Ramón Sangüesa. Incremental methods for Bayesian network structure learning. Technical report, Universitat Politècnica de Catalunya, 1999. 56, 148

[125] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A Bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers from the 1998 workshop*, volume 62, pages 98–105, 1998. 5

[126] Gideon Schwarz. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978. 46

[127] Ross D. Shachter and Mark Alan Peot. Simulation Approaches to General Probabilistic Inference on Belief Networks. In *Proceedings of the Fifth Annual Conference on Uncertainty in Artificial Intelligence*, UAI '89, pages 221–234, Amsterdam, The Netherlands, The Netherlands, 1990. North-Holland Publishing Co. 42

[128] Da Shi and Shaohua Tan. Hybrid incremental learning algorithms for Bayesian network structures. In *Proceedings of the 9th IEEE International Cognitive Informatics (ICCI) Conf*, pages 345–352, 2010. 76, 77

[129] Da Shi and Shaohua Tan. Incremental learning Bayesian network structures efficiently. In *Proceedings of the 11th Int Control Automation Robotics & Vision (ICARCV) Conf*, pages 1719–1724, 2010. 76, 77, 148

[130] Moninder Singh and Marco Valtorta. An algorithm for the construction of Bayesian network structures from data. In *Proceedings of the 9th Annual Conference on Uncertainty in AI*, pages 259–265. Morgan Kaufmann Publishers Inc., 1993. 50

[131] David J. Spiegelhalter and Steffen L. Lauritzen. Sequential updating of conditional probabilities on directed graphical structures. *Networks*, 20(5):579–605, 1990. 56

[132] Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction, and Search*. Adaptive Computation and Machine Learning Series. The MIT Press, 2nd edition, 2000. 49, 77, 78, 103, 146, 158

[133] Peter Spirtes and Christopher Meek. Learning Bayesian networks with discrete variables from data. In *Proceedings of the First International Conference on*

Knowledge Discovery and Data Mining, pages 294–299, San Francisco, 1995. Morgan Kaufmann. 50

[134] Maguelonne Teisseire, Pascal Poncelet, Parc Scientifique, Georges Besse, Florent Masseglia, Florent Masseglia, and Inria Sophia Antipolis. Sequential pattern mining: A survey on issues and approaches. In *Encyclopedia of Data Warehousing and Mining, nformation Science Publishing*, pages 3–29. Oxford University Press, 2005. 27

[135] Traderbot. http://www.traderbot.com. 28

[136] Ioannis Tsamardinos, Constantin F. Aliferis, and Alexander Statnikov. Time and sample efficient discovery of Markov blankets and direct causal relations. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge Discovery and Data mining*, KDD '03, pages 673–678, New York, NY, USA, 2003. ACM. 50, 147

[137] Ioannis Tsamardinos, Constantin F. Aliferis, Alexander Statnikov, and Laura E. Brown. Scaling-up Bayesian network learning to thousands of variables using local learning techniques. Technical report, Vanderbilt University, 2003. 53

[138] Ioannis Tsamardinos, Laura E. Brown, and Constantin F. Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. In *Machine Learning*, pages 31–78. Springer Netherlands, Inc., 2006. 7, 49, 50, 53, 90, 99, 103, 104, 133, 147, 149, 159

[139] Ioannis Tsamardinos, Alexander R. Statnikov, Laura E. Brown, and Constantin F. Aliferis. Generating Realistic Large Bayesian Networks by Tiling. In *Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference*, pages 592–597, California, USA, 2006. AAAI Press. 116, 163

[140] Robert Van Dam, Irene Langkilde-Geary, and Dan Ventura. Adapting adtrees for high arity features. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*, AAAI'08, pages 708–713. AAAI Press, 2008. 106

[141] Frank Van Harmelen, Vladimir Lifschitz, and Bruce Porter. *Handbook of knowledge representation*, volume 1. Elsevier, 2008. 42

[142] Thomas S. Verma and Judea Pearl. Equivalence and synthesis of causal models. In M. Henrion, R. Shachter, L. Kanal, and J. Lemmer, editors, *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 220–227, San Francisco, 1991. Morgan Kaufmann. 40

[143] João Vinagre and Alípio Mário Jorge. Forgetting mechanisms for incremental collaborative filtering. In *WTI 2010: Proceedings of the III International Workshop on Web and Text Intelligence*, pages 23–28, Brazil, October 2010. 21, 109, 162

[144] Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge Discovery and Data mining*, pages 226–235, New York, NY, USA, 2003. ACM. 26, 141

[145] Kaijun Wang, Junying Zhang, Fengshan Shen, and Lingfeng Shi. Adaptive learning of dynamic Bayesian networks with changing structures by detecting geometric structures of time series. *Knowledge and information systems*, 17(1):121–133, 2008. 137

[146] Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, pages 407–418, New York, NY, USA, 2006. ACM. 14

[147] Chongsheng Zhang, Yuan Hao, Mirjana Mazuran, Carlo Zaniolo, Hamid Mousavi, and Florent Masseglia. Mining frequent itemsets over tuple-evolving data streams. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 267–274. ACM, 2013. 16

[148] Chongsheng Zhang, Florent Masseglia, and Yves Lechevallier. ABS: The anti bouncing model for usage data streams. In *Proceedings of the 10th IEEE*

International Conference on Data Mining (ICDM), pages 1169–1174. IEEE, 2010. 25

[149] Nevin Zhang and David Poole. A simple approach to Bayesian network computations. In *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*, pages 171–178, 1994. 41

[150] Yunyue Zhu and Dennis Shasha. Statstream: statistical monitoring of thousands of data streams in real time. In *VLDB '02: Proceedings of the 28th international conference on very large data bases*, pages 358–369. VLDB Endowment, 2002. 21

l'université nantes angers le mans
PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Amanullah YASIN

Incremental Bayesian network structure learning from data streams

Résumé

Dans la dernière décennie, l'extraction du flux de données est devenue un domaine de recherche très actif. Les principaux défis pour les algorithmes d'analyse de flux sont de gérer leur infinité, de s'adapter au caractère non stationnaire des distributions de probabilités sous-jacentes, et de fonctionner sans relecture. Par conséquent, les techniques traditionnelles de fouille ne peuvent s'appliquer directement aux flux de données. Le problème s'intensifie pour les flux dont les domaines sont de grande dimension tels que ceux provenant des réseaux sociaux, avec plusieurs centaines voire milliers de variables. Pour rester a jour, les algorithmes d'apprentissage de réseaux Bayésiens doivent pouvoir intégrer des données nouvelles en ligne. L'état de l'art en la matiere implique seulement plusieurs dizaines de variables et ces algorithmes ne fonctionnent pas correctement pour des dimensions supérieures. Ce travail est une contribution au problème d'apprentissage de structure de réseau Bayésien en ligne pour des domaines de haute dimension, et a donné lieu à plusieurs propositions. D'abord, nous avons proposé une approche incrémentale de recherche locale, appelée iMMPC. Ensuite, nous avons proposé une version incrémentale de l'algorithme MMHC pour apprendre la structure du réseau. Nous avons également adapté cet algorithme avec des mécanismes de fenêtre glissante et une pondération privilégiant les données nouvelles. Enfin, nous avons démontré la faisabilité de notre approche par de nombreuses expériences sur des jeux de données synthétiques.

Mots clés

flux de données, réseaux Bayésiens, apprentissage de structure, apprentissage incrémental.

Abstract

In the last decade, data stream mining has become an active area of research, due to the importance of its applications and an increase in the generation of streaming data. The major challenges for data stream analysis are unboundedness, adaptiveness in nature and limitations over data access. Therefore, traditional data mining techniques cannot directly apply to the data stream. The problem aggravates for incoming data with high dimensional domains such as social networks, bioinformatics, telecommunication etc, having several hundreds and thousands of variables. It poses a serious challenge for existing Bayesian network structure learning algorithms. To keep abreast with the latest trends, learning algorithms need to incorporate novel data continuously. The existing state of the art in incremental structure learning involves only several tens of variables and they do not scale well beyond a few tens to hundreds of variables. This work investigates a Bayesian network structure learning problem in high dimensional domains. It makes a number of contributions in order to solve these problems. In the first step we proposed an incremental local search approach *iMMPC* to learn a local skeleton for each variable. Further, we proposed an incremental version of *Max-Min Hill-Climbing* (MMHC) algorithm to learn the whole structure of the network. We also proposed some guidelines to adapt it with sliding and damped window environments. Finally, experimental results and theoretical justifications that demonstrate the feasibility of our approach demonstrated through extensive experiments on synthetic datasets.

Key Words

Bayesian network, Data stream mining, Structure learning, Incremental learning.