

***Distributed and Online Maintenance of Graphical Models in  
Apache Flink***

*Copyright © 2022 Nikolaos Tzimos*

*All rights reserved.*



## **Abstract**

*With the growing need for large scale data analysis, distributed machine learning has grown importance in recent years. The raw data is described by a large number of interrelated variables and an important task is to describe the joint probability distribution over these variables, allowing simultaneous interferences and predictions to be made. Direct modeling of the joint probability distribution of all these variables may be infeasible, since the complexity of such a model grows exponentially with the number of variables. We focus on Bayesian Networks, the father of graphical models, and present a different communication-efficient approach using the well-known method of Functional Geometric Monitoring, for continuous learning and maintenance of Bayesian Networks in a distributed streaming environment. Finally, the experimental results confirmed the functionality of the proposed method.*



# Table of Contents

Introduction .....	9
<b>1.1 Objective</b> .....	9
<b>1.2 Thesis Outline</b> .....	10
Theoretical Background.....	11
<b>2.1 Notation</b> .....	11
<b>2.2 Bayesian Networks</b> .....	14
<b>2.2.1 Introduction to Bayesian Networks</b> .....	14
<b>2.2.2 Bayesian Network Semantics</b> .....	18
<b>2.2.3 Student Bayesian Network</b> .....	21
<b>2.3 Naïve Bayes Classifiers</b> .....	22
<b>2.3.1 Introduction to Naïve Bayes Classifiers</b> .....	22
<b>2.3.2 Naïve Bayes Classifiers Semantics</b> .....	22
<b>2.4 Forward Sampling</b> .....	26
<b>2.5 Learning parameters</b> .....	30
<b>2.5.1 Maximum Likelihood Estimation (MLE)</b> .....	30
<b>2.6 Laplace Smoothing</b> .....	33
<b>2.7 Distributed Continuous Model</b> .....	35
Problem statement.....	38
<b>3.1 Problem definition</b> .....	38
<b>3.2 The general approach</b> .....	40
Problem analysis.....	43
<b>4.1 A first approach</b> .....	43
<b>4.1.1 Approximated Distributed counters</b> .....	44
<b>4.1.1.1 Randomized Counters</b> .....	45
<b>4.1.1.2 Deterministic Counters</b> .....	48
<b>4.1.1.3 Comparison between RANDOMIZED and DETERMINISTIC</b> .....	49
<b>4.1.2 Analysis of BASELINE, UNIFORM and NON_UNIFORM</b> .....	51
<b>4.1.2.1 Dummy Father</b> .....	54
<b>4.1.2.2 Comparison among BASELINE, UNIFORM, NON_UNIFORM algorithms</b> .....	55
<b>4.1.3 Communication cost of Naïve Bayes Classifier</b> .....	55
<b>4.2 A second approach</b> .....	57
<b>4.2.1 Functional Geometric Monitoring (FGM)</b> .....	57
Design and Implementation of the system.....	60
<b>5.1 Apache Flink</b> .....	60

5.2 Apache Kafka .....	61
5.3 System Architecture .....	62
Experimental evaluation .....	65
6.1 Datasets .....	65
6.2 Performance metrics.....	66
6.3 Experimental Results .....	67
6.3.1 Communication cost related to the number of training instances .....	67
6.3.2 Communication cost related to the approximation factor $\epsilon$ .....	71
6.3.3 Communication cost related to the number of workers .....	73
6.3.4 Scalability.....	73
6.3.5 Dummy Father .....	79
Conclusions .....	80
7.1 Conclusions .....	80
7.2 Future Work.....	80
Appendix .....	81
Bibliography .....	93

## List of Figures

Figure 1: Directed Acyclic Graph (DAG) of the Student Bayesian network.....	12
Figure 2: Probability mass function(pmf) from a binary-valued random variable $X_i$ .....	16
Figure 3: Part of the graph from the Student Bayesian Network.....	17
Figure 4: Tabular CPD of Grade node .....	19
Figure 5: Equal width binning .....	20
Figure 6: Student Bayesian Network accompanied with CPDs .....	21
Figure 7: Naïve Bayes Classifier .....	23
Figure 8: Naïve Bayes Classifier for spam filtering .....	25
Figure 9: Topological ordering of Student Bayesian Network.....	27
Figure 10: Sampling of a multinomial distribution .....	28
Figure 11: Distributed Continuous Model .....	36
Figure 12: $nRC$ and $nDC$ varying the number of sites $k$ .....	50
Figure 13: Dummy Father .....	54
Figure 14: Apache Flink .....	60
Figure 15: Feedback loop .....	61
Figure 16: Apache Kafka .....	61
Figure 17: System Architecture .....	62
Figure 18: Hashing .....	64
Figure 19: Error to GT related to the training instances for both approaches, for the HEPAR II dataset.....	68
Figure 20: Error to EXACTMLE related to the number of training instances for both approaches for the HEPAR II dataset .....	68
Figure 21: Communication cost related to the number of training instances for both approaches from the HEPAR II dataset .....	69
Figure 22: Communication cost related to the number of training instances for both approaches from the HEPAR II dataset -Best results .....	70
Figure 23: Communication cost related to the approximation factor $\epsilon$ , for the HEPAR II dataset and UNIFORM algorithm .....	71
Figure 24: Communication cost related to the approximation factor $\epsilon$ for both approaches, for the HEPAR II dataset and UNIFORM algorithm .....	72
Figure 25: Error to GT related to the approximation factor $\epsilon$ for both approaches, for the HEPAR II dataset and UNIFORM algorithm .....	72
Figure 26: Communication cost related to the number of workers sites for both approaches, from the HEPAR II dataset and UNIFORM algorithm .....	73
Figure 27: Throughput(events/sec) related to the number of sites for both approaches, for the HEPAR II dataset.....	74
Figure 28: Throughput(events/sec) related to the number of sites for the first approach for the HEPAR II dataset HEPAR II, after the initial state .....	75
Figure 29: Throughput(events/sec) related to the number of sites for both approaches, for the HEPAR II dataset -Best results.....	76
Figure 30: Throughput(events/sec) related to the number of parallelism for both approaches for the HEPAR II dataset .....	76
Figure 31: Throughput(events/sec) related to the number of parallelism for the first approach for the HEPAR II dataset, after the initial state .....	77

Figure 32: Throughput(events/sec) related to the number of parallelism for both approaches for the HEPAR II dataset- Best results .....	78
Figure 33: Communication cost in conjunction with the Dummy Father (DF) method for HEPAR II, ALARM datasets.....	79
Figure 34: Mean error to GT related to the number of training instances for both approaches for the HEPAR II dataset .....	81
Figure 35: Communication cost related to the number of training instances for LINK, ALARM datasets .....	82
Figure 36: Communication cost related to the approximation factor $\epsilon$ for both approaches for the HEPAR II dataset .....	83
Figure 37: Communication cost related to the number of sites for both approaches for the HEPAR II dataset .....	84
Figure 38: Runtime(sec) related to the number of sites for both approaches for the HEPAR II dataset.....	84
Figure 39: Runtime(sec) related to the number of sites for the first approach for the HEPAR II dataset, after the initial state.....	85
Figure 40: Runtime(sec) related to the number of sites for both approaches for the HEPAR II dataset - Best results.....	85
Figure 41: Runtime(sec) related to the number of sites for both approaches for the HEPAR II dataset.....	86
Figure 42: Runtime(sec) related to the number of sites for the first approach for the HEPAR II dataset, after the initial state.....	86
Figure 43: Runtime(sec) related to the number of sites for both approaches for the HEPAR II dataset - Best results.....	87
Figure 44: Project Structure .....	88

## List of Tables

Table 1: Notation summary.....	45
Table 2: Space and communication complexity for each counter .....	49
Table 3: Approximation factor and Communication Cost for algorithms .....	54
Table 4: Bayesian Networks .....	66



# Chapter 1:

## Introduction

### *1.1 Objective*

The present thesis proposes a system capable of construction and maintenance of the well-known graphic model, namely Bayesian Networks, on data referring to large volumes of data, that are evolving, distributed, and multidimensional, while using minimal communication cost. The purpose of the system is the maintenance and continuous parameter learning using communication-efficient algorithms on distributed continuous model based on the Maximum Likelihood Estimation (MLE) algorithm.

The first approach is based on using approximate distributed counters with the combinations of the algorithms BASELINE, UNIFORM, and NON\_UNIFORM. These algorithms propose how we can define the approximation factor  $\epsilon$  on each counter, which aims to provide appropriate error guarantees of the joint probability distribution of MLE. The system supports two types of fundamental approximate distributed counters, the first refers to the RANDOMIZED counters and the second one refers to the DETERMINISTIC counters. This approach leads to an exponential reduction in communication cost compared to the conservative approach of EXACTMLE, which uses EXACT counters.

Moreover, the system supports the maintenance of a special-case of Bayesian Networks, the Naïve-Bayes Classifiers (NBC). The implementation of the system takes part in Apache Flink and Apache Kafka.

The present thesis except for the first approach, proposes an alternative approach to the maintenance and continuous parameter learning of Bayesian Networks. In particular, this approach uses the well-known method of Functional Geometric Monitoring in combination with the BASELINE and UNIFORM algorithms, which divide with the appropriate way the available error budget. The idea behind the use of the Functional Geometric Monitoring method is now each counter we do not need to treat each individual, but the approximate distributed counters can be treated as frequency vectors. This approach leads to the reduction of communication cost up to one order of magnitude to the previous approach and up to two orders of magnitude to EXACTMLE. Finally, the experimental evaluation confirms that the proposed approach can handle satisfyingly large data volumes by providing the appropriate scaling.

The present thesis integrates the widely known method of Laplace Smoothing for handling the zero parameters and proposes the method of Dummy Father (DF) in combination with the NON\_UNIFORM algorithm. The experimental evaluation confirms that this leads to a 50% reduction in communication cost.

## 1.2 Thesis Outline

The thesis is organized as follows: **Chapter 2** provides the necessary theoretical background, covering crucial concepts related to Bayesian Networks and Naïve Bayes Classifiers. Each concept is illustrated with a relevant example. Additionally, a detailed analysis is conducted for the Forward Sampling Algorithm and the method of Laplace Smoothing. Furthermore, the chapter delves into the Maximum Likelihood Estimate algorithms used for parameter learning and introduces the Distributed Continuous Model, the model employed in this thesis. The analysis within Chapter 2 offers a comprehensive exploration of these topics, providing valuable insights into their implementation and implications.

In **Chapter 3**, it contains the definition of the problem and the general approach that is used for both approaches. In **Chapter 4**, it contains the analysis of the problem, particularly the analysis of the first approach considering the approximate distributed counters, which in this case, we treat as individual entities. Finally, it contains the analysis of the second approach, which employs Functional Geometric Monitoring (FGM). In **Chapter 5**, it contains the design and implementation of the proposed system. In **Chapter 6**, it contains the experimental evaluation of each approach from different perspectives. Finally, in **Chapter 7**, it contains the conclusions, future work, and extensions of the system.

## Chapter 2:

# Theoretical Background

## 2.1 Notation

Throughout this thesis, the notations used are primarily based on [1].

Random variables are denoted using capital Latin letters  $X, Y, Z$  while their corresponding values are represented using the lowercase Latin letters  $x, y, z$ , respectively. Moreover, the notation  $Val(X)$  denotes the set of values of random variable  $X$ . When referring to categorical (discrete) random variables, we use the notation  $x^1, \dots, x^k$  to enumerate the values of random variable  $X$ , according to the fact that the continuous random variable  $X$  contains  $k$  different values. Additionally,  $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$  represent sets of random variables while  $\mathbf{x}, \mathbf{y}, \mathbf{z}$  contain the values of the set from random variables, accordingly, we can define  $Val(\mathbf{X})$  so in this case we refer to the set of values from the set of random variables  $\mathbf{X}$ .

Moreover, the function  $|\cdot|$  denotes the cardinality of the corresponding function-relation expect when defined differently. For instance,  $|Val(X)|$  corresponds to the number of values of the random variable  $X$  or equivalently the number of values contained in the set of values of the random variable  $X$ . In most cases, we use the  $P(x)$  for  $P(X = x)$ , given the fact that the  $x$  refers to the values of the random variable  $X$ . Finally, we consider  $P((X = x) \cap (Y = y))$  to be equivalent to  $P(X = x, Y = y)$  or  $P(x, y)$  or  $P(X, Y)$ .

Suppose we have a set of random variables  $\mathbf{X} = \{X_1, \dots, X_n\}$ . The joint probability distribution over the set  $\mathbf{X}$  is denoted as  $P(X_1, \dots, X_n)$  or  $P(\mathbf{X})$  while the distribution  $P(X_i)$  refers to the marginal distribution of the random variable  $X_i$ . When referring to a conditional probability distribution (CPDs), we use the notation  $P(X | Y)$  where  $X, Y$  represent any random variable. Finally, we use the  $\xi$  to denote a full-assignment of random variables from any set of random variables  $\mathbf{X}$ .

The two most fundamental rules that are used and are worth mentioning are the Chain Rule [1] and Bayes Rule [2]. Both rules are based on a widely known method of conditional probability. We define the conditional probability below.

Suppose  $\alpha, \beta$  refers to any two events. The conditional probability of  $\beta$  given  $\alpha$  is defined as:

$$\text{Conditional Probability : } P(\beta | \alpha) = \frac{P(\alpha \cap \beta)}{P(\alpha)}$$

The conditional probability is defined only when the denominator is greater than zero i.e.,  $P(\alpha) > 0$ .

The first rule, the Chain Rule, states that if  $\alpha_1, \dots, \alpha_k$  is a sequence of events, then:

$$\text{Chain Rule : } P(\alpha_1 \cap \dots \cap \alpha_k) = P(\alpha_1) \cdot P(\alpha_2 | \alpha_1) \cdots P(\alpha_k | \alpha_1 \cap \dots \cap \alpha_{k-1})$$

If the sequence of events consists of independent events, meaning  $P(a_i \cap a_j) = P(a_i) \cdot P(a_j)$  for each  $i, j \in \{1, \dots, \kappa\}$ , then:

$$P(a_1 \cap \dots \cap a_\kappa) = P(a_1) \cdot P(a_2) \cdots P(a_\kappa)$$

The second rule, *Bayes' Rule*, is an immediate consequence of the definition of conditional probability, if  $\alpha, \beta$  represents any two events, then:

$$\text{Bayes' Rule : } P(\alpha | \beta) = \frac{P(\beta | \alpha) \cdot P(\alpha)}{P(\beta)}$$

## Graphs

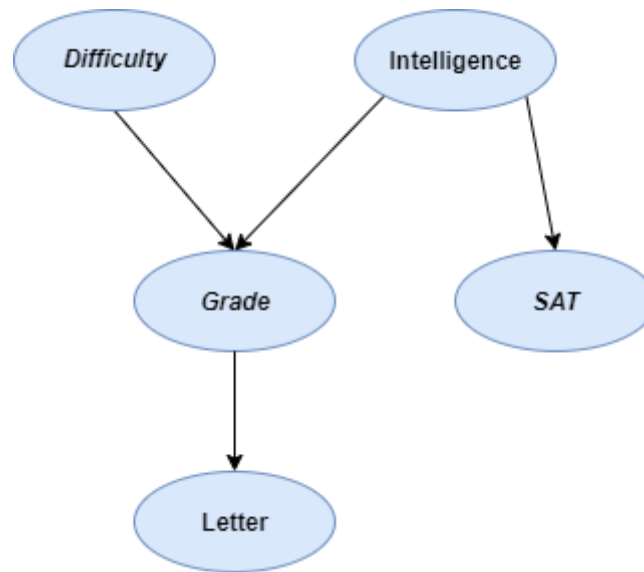


Figure 1: Directed Acyclic Graph (DAG) of the Student Bayesian network

This section contains the most fundamental notations used throughout this thesis regarding graphs. More details are represented in [1, Ch. 2].

The most primary *data structure* used in this thesis is the well-known structure of the *graph*. A graph consists of a set of nodes  $\mathcal{V}$  and a set of edges  $\mathcal{E}$ . Hereafter, we will use the notation  $\mathcal{G}(\mathcal{V}, \mathcal{E})$  or  $\mathcal{G}(\mathcal{X}, \mathcal{E})$  to denote a graph  $\mathcal{G}$ , where the  $\mathcal{V} = \{V_1, \dots, V_n\}$  represents the set of nodes and the  $\mathcal{E} = \{E_1, \dots, E_n\}$  denotes the set of edges, respectively.

In particular, we focus on *Directed Acyclic Graph (DAG)*. In this case each pair of nodes  $V_i, V_j \in \mathcal{V}$  is exclusively connected by a directed edge  $V_i \rightarrow V_j$ . The set of edges  $\mathcal{E}$  contains all the possible pairs of connected nodes. Additionally, the *acyclic* property of the graph ensures that there is no *directed path*  $V_1, \dots, V_k$  where  $V_1 = V_k$  which would represent a *loop*.

Given a *directed acyclic graph*  $\mathcal{G}(\mathcal{X}, \mathcal{E})$ , for any edge  $X_i \rightarrow X_j \in \mathcal{E}$  where  $X_i, X_j \in \mathcal{X}$  then the node  $X_j$  is the *child* node of  $X_i$ . We will use the notation  $Child(X_j)$  to express the set of nodes that are parents of  $X_j$ . Similarly, we will use the notation  $Par(X_i)$  to denote the set of nodes that are parents of  $X_i$ . Additionally, the notation  $NonDescendants(X_i)$  will be used to represent the set of nodes that are not *descendants* of  $X_i$ .

Figure 2 represents an instance of a *DAG* where the set of nodes  $\mathcal{V}$  is as follows:

$$\mathcal{V} = \{Difficulty, Intelligence, Grade, SAT, Letter\}$$

While the set of edges is as follows:

$$\mathcal{E} = \{Difficulty \rightarrow Grade, Intelligence \rightarrow Grade, Intelligence \rightarrow SAT, Grade \rightarrow Letter\}$$

Using the previous statements, we can make the following declarations for the node *Grade*:

$$Par(Grade) = \{Difficulty, Intelligence\}$$

$$Child(Grade) = \{Letter\}$$

$$NonDescendants(Grade) = \{SAT\}$$

## 2.2 Bayesian Networks

### 2.2.1 Introduction to Bayesian Networks

*Bayesian Networks* [3] represent an intersection between *graph theory* and *probability theory*, aiming to facilitate the modeling of probabilistic and causal inferences in *real-world* applications [4].

Bayesian Networks can be applied in various fields. For instance, they can be used as decision-support systems, specifically in sensor validation systems [5]. The basic idea is to determine whether a sensor is faulty by analyzing a set of sensor readings (represented through the Bayesian Network) that contribute to the overall system operation.

Another application of Bayesian Networks is in medical diagnosis systems [4, Ch.2]. In this case, the objective is to diagnose a disease based on patient findings. The basic idea can be expressed as follows:

$$diagnosis = \max_i P(D_i|E)$$

Generally,  $P(D_i|E)$  represents the probability of disease  $D_i$  given a set of evidence  $E$ , which includes observed patient symptoms and laboratory examination results. Examples of such networks include MUNIN [6], used for diagnosing neuromuscular disorders, and HEPAR2 [7], used for diagnosing liver disorders.

Bayesian Networks have various applications, including cybersecurity [8]. In this context, Bayesian Networks are utilized as real-time security analysis systems that determine whether the input is malicious or benign.

One specific type of Bayesian Network that falls under the category of Classifiers is Naïve Bayes Classifiers (See Chapter 2.3). Naïve Bayes Classifiers find applications in diverse fields. Some well-known examples include Document Classifiers [4, Ch.11] and Information Retrieval systems [4, Ch.12].

The graphical models we are referring to initially belong to the category of probabilistic graphical models, specifically Bayesian Networks. As previously mentioned, Bayesian Networks are graphical models with a fundamental characteristic represented by a graph (See Chapter 2.1). The use of a graph provides a compact representation of the joint probability distribution, typically defined in a high-dimensional probability space.

The representation of a graph has two perspectives. The first perspective pertains to representing the set of independencies among the nodes. The second aspect involves utilizing these independencies to decompose the initial distribution into smaller factors, typically defined in lower-dimensional probability spaces. This approach allows us to avoid working with the high-dimensional probability space of the joint probability distribution.

## Probability queries

Except for learning parameters (See Chapter 2.5), which define the joint probability distribution of a Bayesian Network, another objective is to estimate queries related to the distributions of random variables. This specific category of queries is referred to as probability queries. Probability queries consist of two parts:

- The first part pertains to the evidence, which comprises a subset  $\mathbf{E}$  of random variables accompanied by an instance  $\mathbf{e}$  of values from those random variables. We also use queries where the evidence is the empty set, i.e.,  $\mathbf{E} = \emptyset$ .
- The second part relates to the *query variables*. Generally, this set consists of a subset  $\mathbf{Y}$  of random variables. In most cases, the query variables pertain to the entire set of random variables in the network, rather than a subset. This implies querying the joint probability distribution of the network.

$$\text{Probability queries : } P(\mathbf{Y} | \mathbf{E} = \mathbf{e})$$

The main objective is to estimate the *probability queries* over the *joint probability distribution*, assuming that  $\mathbf{E} = \emptyset$ . We refer to these queries as follows:

$$P(\mathbf{Y}) = P(Y_1, \dots, Y_n)$$

Assuming a set of random variables  $\mathbf{X} = \{X_1, \dots, X_n\}$ , where each variable represents a binary value (e.g., the outcome of a coin toss), the objective is to estimate the *joint probability distribution*  $P(X_1, \dots, X_n)$  defined by the random variables in set  $\mathbf{X}$ . In the simplest case, this would require  $2^n - 1$  parameters, with one parameter for each possible assignment of values  $x_1, \dots, x_n$  from the random variables. However, managing the joint probability distribution becomes infeasible from various perspectives. This includes the challenge of allocating sufficient heap space to store all the different parameter values of the joint distribution, as well as the substantial volume of datasets required to estimate the parameters, which exponentially increases with the number of random variables.

Now, let's assume that the set  $\mathbf{X}$  consists of *marginally independent* random variables. This means that the distributions for any pair of random variables  $X_i, X_j$  are independent ( $X_i \perp X_j$ ). By applying the Chain Rule (See Chapter 2.1), we can express the joint probability distribution as follows:

$$P(X_1, \dots, X_n) = P(X_1) \cdot \dots \cdot P(X_n)$$

Using the *independence* property among the random variables, now we can define the *joint probability distribution* exclusively using the *marginal distribution*  $P(X_i)$  of the random variables  $\mathbf{X}$ . Exploiting the previous statement, if  $\theta_{X_i}$  denotes the parameter which defines the *marginal probability distribution*  $P(X_i)$  of the binary-valued variable.

By utilizing the independence property among the random variables, we can define the *joint probability distribution* exclusively using the *marginal distribution*  $P(X_i)$  of the random variables  $\mathbf{X}$ . Expanding on the previous statement, if  $\theta_{X_i}$  denotes the parameter that defines the *marginal probability distribution*  $P(X_i)$  of a binary-valued variable  $X_i$ :

Marginal probability distribution :  $P(X_i = x_i) = \theta_{x_i} = \begin{cases} \theta_i, & \text{where } x_i = x_i^1 \\ 1 - \theta_i, & \text{where } x_i = x_i^0 \end{cases}$

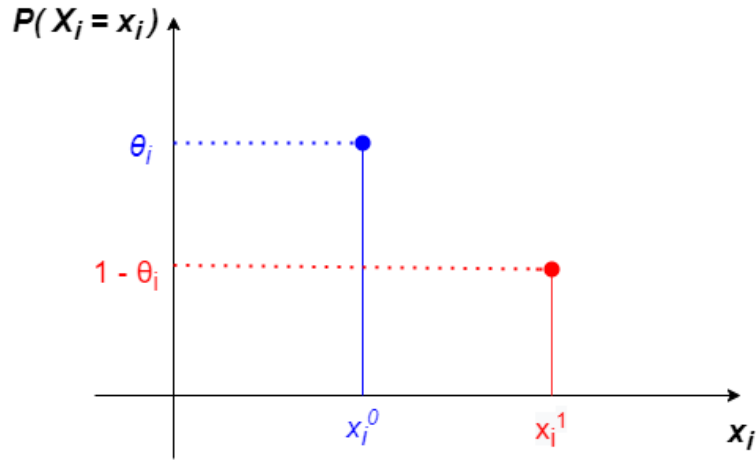


Figure 2: Probability mass function (pmf) from a binary-valued random variable  $X_i$

Now the only requirement to define the *joint probability distribution* is to determine the parameters  $\theta_{x_1}, \dots, \theta_{x_n}$  for distributions of random variables  $X_1, \dots, X_n$ . In conclusion, the joint probability distribution defines as follows:

$$P(x_1, \dots, x_n) = \prod_{i=1}^n \theta_{x_i}$$

In this way, we successfully reduce the initial space of *joint probability distribution* from a subspace of  $\mathbb{R}^{2^n}$  to an *n-dimensional manifold* of  $\mathbb{R}^{2^n}$ . Additionally, we achieve to obtain linear dependence on the number of random variables.

The previous example represents an ideal scenario that is difficult to achieve in *real-world* applications due to the presence of dependencies among different random variables. In most cases, it is more reasonable to consider *conditional independences* among different random variables as an alternative approach. By incorporating *conditional independences* and *Directed Acyclic Graph (DAG)*, we can effectively model and represent Bayesian Networks.



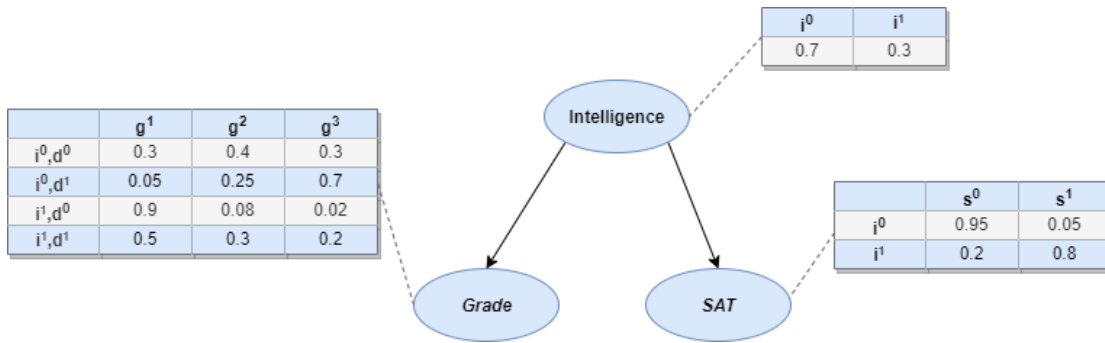


Figure 3: Part of the graph from the Student Bayesian Network

Let's assume the *Bayesian Network* from Figure 3. The network consists of the nodes *Intelligence(I)*, *Grade(G)*, and *SAT(S)*. The random variable  $I$  contains two values  $Val(I) = \{i^1, i^0\}$ , where  $i^1$  refers to *high-intelligence students* and  $i^0$  refers to *low-intelligence students*. Similarly, the random variable  $S$  contains two values  $Val(S) = \{s^1, s^0\}$ , where  $s^1$  denotes a *high score* and  $s^0$  denotes a *low score*. Finally, the random variable  $G$  has three values  $Val(G) = \{g^1, g^2, g^3\}$ , where  $g^1, g^2, g^3$  correspond to grades *A, B, C*, respectively. In this case, the direct approach to the *joint probability distribution* of the three variables requires  $12(2 \cdot 2 \cdot 3)$  parameters, with each parameter representing the probability of the combination of values from three variables.

The property of versatile independence among random variables can't be applied due to *dependencies* among them (for instance, the grade of a student affects the student's *intelligence* up to a certain point). We assume the *conditional independency*:  $(S \perp G | I)$  meaning that the random variable *Grade(G)* is independent of the random variable *SAT(S)* given the random variable *Intelligence(I)*.

Using the previous *conditional independency*, we can express the *joint probability distribution* using the following equation:

$$P(I, G, S) = P(S, G | I) \cdot P(I) = P(S | I) \cdot P(G | I) \cdot P(I)$$

Even in the simplest case, with the usage of one *conditional independency*, we can separate the *joint probability distribution* into smaller independent factors, which contributes to a more compact representation of the initial distribution. Now, the number of parameters required to define the *joint distribution* is 7 parameters (the difference between them is not significant, but it increases as the number of variables and *conditional independencies* also increases, the example is illustrative), one parameter for the distribution  $P(I)$ , two parameters for the distribution  $P(S)$  and four parameters for the distribution  $P(G)$ .

The separation of a *joint probability distribution* into smaller factors highlights the property of *modularity*. This means that the insertion or deletion of a random variable affects only a specific part of joint distribution rather than the entire as in direct approach of distribution. In the direct approach, we would need to renumerate all the possible combinations of values from the new set of random variables. Moreover, the factors of *joint probability distribution* compromise the main characteristic so the *Maximum Likelihood Estimation (MLE)* – See *Chapter 2.5*) can be applied in a distributed environment (See *Chapter 2.7*).

In conclusion, the representation of the *Bayesian Network* through a *directed acyclic graph (DAG)* offers two advantages:

- A compact representation of the set of *conditional independences* among the random variables that define the *joint probability distribution*.
- A mechanism that provides a compact representation of *joint probability distribution(factors)*. Through the process of *factorization*, we can avoid the initial *high-dimensional* space and instead work with a product of factors corresponding to *lower-dimensional* spaces.

## 2.2.2 Bayesian Network Semantics

This section presents the most essential definitions regarding the *Bayesian Networks*.

A *Bayesian Network structure*, denoted as  $\mathcal{G}(\mathcal{X}, \mathcal{E})$  is a *directed acyclic graph (DAG)* – See Chapter 2.1), where the nodes correspond to the set of random variables  $\mathcal{X} = \{X_1, \dots, X_n\}$  and the set of *directed edges*  $\mathcal{E}$  corresponds to the *conditional independences* among nodes. Each *directed edge* indicates a *direct dependence* among the involved random variables. We denote  $Par(X_i)$  as the set of parents nodes of  $X_i$  and  $NonDescendants(X_i)$  as the set of nodes that are not *descendants'* nodes of  $X_i$ . The graph  $\mathcal{G}$  can express the set of *conditional independences*, referred to as *local independences* and denoted as  $I_\ell(\mathcal{G})$ , with the following equation:

$$P(X_1, \dots, X_n) = \prod_{i=1}^n P(X_i | Par(X_i))$$

The latter equation is known as the *chain rule of Bayesian Networks*. Each factor of product  $P(X_i | Par(X_i))$  corresponds to well-known *local conditional probability distributions (CPDs)*.

The random variables used in this thesis belong to the category of *categorical-discrete* random variables. As a result, the *conditional probability distributions (CPDs)* of these variables fall into the well-known category of *Tabular CPDs*. Each *CPD*  $P(X_i | Par(X_i))$  can be represented as a table where each entry corresponds to the probability associated with a combination of values between  $X_i$  and  $Par(X_i)$ (See Figure 4).

Each row of *Tabular CPD*  $P(X_i | Par(X_i) = \mathbf{x}_i)$  represents a *multinomial distribution* with  $k = Val(X_i)$  states where  $\mathbf{x}_i \in Val(Par(X_i))$  and the corresponding probabilities(parameters)  $p_i \in [0,1]$  for each  $i \in Val(X_i)$  and  $\sum_{i=1}^{Val(X_i)} p_i = 1$ . Each row of *Tabular CPD* required a *k-dimensional parameter vector* to define the corresponding *multinomial distribution*. We denote the set of required parameters for all distributions(rows) of the CPD  $P(X_i | Par(X_i))$  of variable  $X_i$  as  $\theta_{X_i}$ . Additionally, we use  $\theta = \{\theta_{X_1}, \dots, \theta_{X_n}\}$  to represent all the parameters required for all *CPDs* in the *Bayesian network*  $\mathcal{G}(\mathcal{X}, \mathcal{E})$ , which is necessary to estimate the *joint probability distribution* of the network.

Tabular CPD	$g^1$	$g^2$	$g^3$
$i^0, d^0$	0.3	0.4	0.3
$i^0, d^1$	0.05	0.25	0.7
$i^1, d^0$	0.9	0.08	0.02
$i^1, d^1$	0.5	0.3	0.2

Multinomial Distribution  $P(\text{Grade} \mid i^0, d^0)$   
with  $\text{Val}(\text{Grade}) = \{g^1, g^2, g^3\}$  and  
 $\theta = \{0.3, 0.4, 0.3\}$

Figure 4: Tabular CPD of Grade node

We consider a *Bayesian Network*  $\mathcal{G}(\mathcal{X}, \mathcal{E})$ , where  $\mathcal{X}$  consists of  $n$  *binary* random variables. It can be proven that a direct approach to *joint probability distribution* requires  $2^n - 1$  parameters. However, by factorizing the *joint probability distribution* based on the graph  $\mathcal{G}$  and assuming that  $k$  is the maximum number of parents nodes for any variable, we can demonstrate that the number of parameters required for the *joint probability distribution* is less than  $n \cdot 2^k$ . This approach allows us to avoid *exponential independence* on the number of nodes  $n$ , which can be infeasible for large values of  $n$ . Instead, the *exponential independence* remains only on the number of nodes  $k$ , which is advantageous since a random variable usually depends on a small number of random variables in the network. In most cases, we observe that  $n \ll k$ . The latter property highlights one of the main benefits of the *Bayesian Network* because we achieve an exponentially smaller number of parameters for the joint probability distribution.

### Discretization

When a Bayesian Network involves *continuous* random variables, the initial step is to convert these variables into discrete ones. This conversion can be accomplished through a process known as *discretization*. This serves as a *pre-processing step* that utilizes the available information for each random variable, before proceeding to learn the parameters of *joint probability distribution*. Various methods can be employed for discretization, and detailed descriptions of these methods can be found in [9]. In this context, we specifically focus on and support a specific method, namely *equal width binning*, which is akin to constructing an *equal width histogram*.

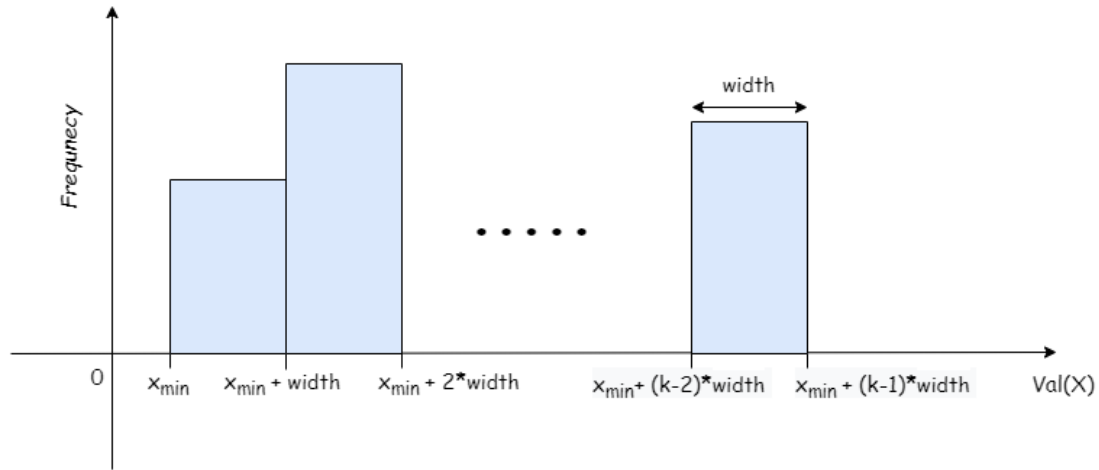


Figure 5: Equal width binning

The objective of the *equal width binning* is to divide the range of values of a continuous random variable into  $k$  equally-sized buckets, where  $k$  is a *user-defined* parameter. It is important to note that this method may not be suitable for *skewed* distributions, although there are more advanced techniques available to address this issue. However, such advanced methods are beyond the scope of this thesis.

Given a continuous random variable  $X$  with  $Val(X) \in [x_{min}, x_{max}]$  and assuming  $k$  represents the number of *buckets*, the width of each *bucket* can be calculated as follows:

$$width = \frac{x_{max} - x_{min}}{k}$$

While creating the *bucket boundaries*, they are defined in the following manner:

$$x_{min} + i \cdot width, \text{ where } i = 1, \dots, k - 1$$

The method falls under the *unsupervised setting* as it does not take into account any available *class labels*. Additionally, it belongs to the category of *global methods*, where the *partitions(buckets)* are created based on the entire *continuous space* defined by the random variable, independent of other variables. It is also considered as a *static* method, as it requires a *user-defined* parameter that determines the maximum number of *buckets-partitions* to be created.

It is important to note that the *discretization* method is not combined with other aspects, such as *Bayesian Structure Learning*. In this thesis, *discretization* is used exclusively as a *pre-processing step*.

### 2.2.3 Student Bayesian Network

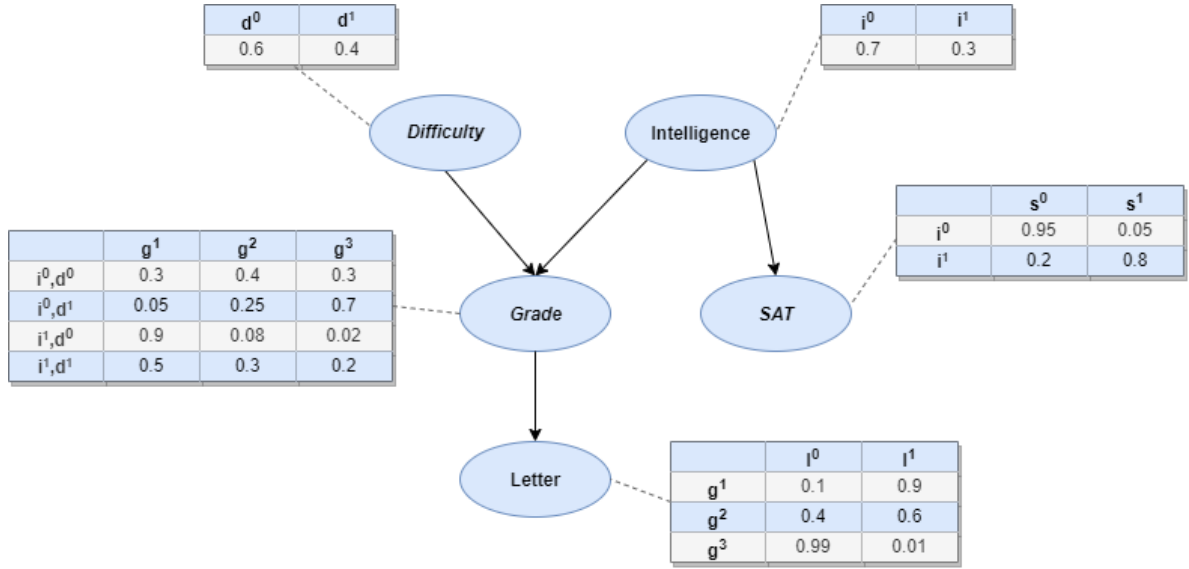


Figure 6: Student Bayesian Network accompanied with CPDs

Given the previous information, we can now examine a comprehensive example of a *Bayesian Network*. We will refer to it as the *Student Bayesian Network* (Figure 6). The *Student Network*  $\mathcal{G}(\mathcal{X}, \mathcal{E})$  consists of five nodes: the Intelligence ( $I$ ), the Difficulty( $D$ ), the Grade( $G$ ), the SAT grade( $S$ ) and the quality of recommendation letter( $L$ ):

$$\mathcal{X} = \{Difficulty(D), Intelligence(I), Grade(G), SAT(S), Letter(L)\}$$

All the random variables in the network are *binary-valued*, except for the *Grade*( $G$ ), which is a *ternary-valued* variable. The set of *local conditional independences*  $I_l(\mathcal{G})$  based on  $\mathcal{G}(\mathcal{X}, \mathcal{E})$ , can be defined as follows:

$$\text{For Intelligence, Difficulty nodes : } (I \perp D)$$

$$\text{For Letter node : } (L \perp I, D, S \mid G)$$

$$\text{For Grade node : } (G \perp S \mid I, D)$$

$$\text{For SAT node : } (S \perp D, G, L \mid I)$$

Given the  $\mathcal{G}(\mathcal{X}, \mathcal{E})$  and  $I_l(\mathcal{G})$ , we can express the *joint probability distribution* of  $\mathcal{G}$  in the following way (*chain rule of Bayesian Networks*):

$$P(I, D, S, G, L) = P(I) \cdot P(D) \cdot P(G \mid I, D) \cdot P(S \mid I) \cdot P(L \mid G)$$

In this case, the number of parameters required to define the *joint probability distribution* is 15. We need one parameter for each *binomial distribution* of *Intelligence*, *Difficulty* nodes, 2 parameters for each of the four *multinomial distributions* of *Grade* node, one parameter for each of the two *binomial distributions* of *SAT* node and one parameter for each of the three *binomial distributions* of *Letter* node. On the other hand, the direct approach to *joint probability distribution* requires 47 parameters ( $2^4 \cdot 3$ ).

### **Example of probability queries in the Student Bayesian Network**

The objective is to estimate *probability queries on joint probability distribution*. Based on the *Student Example*, an example of a probability query can be formulated in the following manner:

$$P(i^1, d^0, g^2, s^1, l^0) = P(i^1) \cdot P(d^0) \cdot P(g^2) \cdot P(s^1) \cdot P(l^0) = 0.004$$

The query expresses the probability of an intelligent student; the probability of a low-difficulty subject; the probability that an intelligent student takes *B* on a low-difficulty subject; the probability that an intelligent student has a high *SAT score* and the probability that a student has *B* grade and takes a weak recommendation letter.

## **2.3 Naïve Bayes Classifiers**

### **2.3.1 Introduction to Naïve Bayes Classifiers**

Initially, we focus on a special case of *Bayesian Networks*, namely the well-known *Naïve Bayes Classifiers* [10]. The *Naïve Bayes Classifier* is an intersection between the *graph and probabilistic theory*. In this case, the *classification* is based on the widely known rule of *Bayes' theorem*, which is described analytically below.

*Naïve Bayes Classifiers* can be applied to a variety of *real-word* applications. They are commonly used as *Documents Classifiers* and are particularly effective in *email spam filtering*. They are also utilized in *Medical Diagnosis Systems* and *Credit-Card Fraud Detection Systems*. In addition, they have proven to be valuable in the field of *Sentiment Analysis*. Furthermore, can be integrated into *recommender* systems [11], where they work in conjunction with *collaborative filtering* methods to create scalable hybrid *recommender systems* that achieve improved *accuracy* compared to the other existing systems. In this case, the *Naïve Bayes Classifier* functions similarly to *Documents Classifiers*, but the set of *class labels* represents a set of *recommendations*.

Considering the process of classification, the objective is to construct a function that assigns *class labels* to instances taking into account the available set of *attributes*. *Both the attribute and the class label set are known in advance, making it a supervised setting*.

### **2.3.2 Naïve Bayes Classifiers Semantics**

The *Naïve Bayes Classifier* operates by learning the *CPDs* of each *feature*  $X_i$  given the *class variable*  $C$ , denoted as  $P(X_i|C)$ , assuming that each *feature*  $X_i$  is independent of others given the *class variable*  $C$  (*naïve Bayes assumption*). *Classification* is performed using *Bayes' Rule*, where the *class label*  $c^i$  with the highest *posterior probability* given the *input feature vector*  $x$ ,  $P(C = c^i | x)$  is selected.

We provide the most fundamental definitions around *Naïve Bayes Classifiers*.

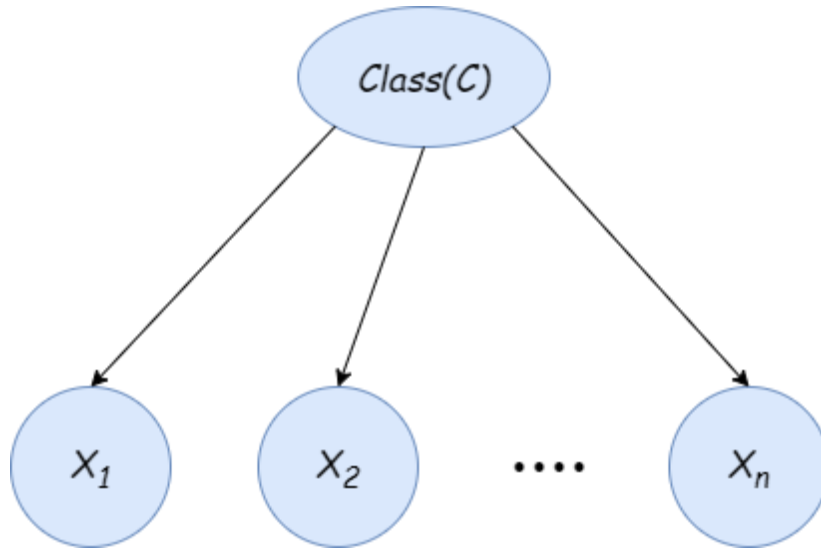


Figure 7: Naïve Bayes Classifier

A *Naïve Bayes Classifiers* structure  $\mathcal{G}(\mathcal{X}, \mathcal{E})$  is a *directed acyclic graph* (Figure 7), where the nodes correspond to the set of random variables  $\mathcal{X} = \{X_1, \dots, X_n, C\}$  and the set of *directed edges*  $\mathcal{E}$  represents the *conditional independences* among nodes. In this case, the set of variables  $\mathcal{X}$  can be divided into two disjoint subsets. The first subset refers to the set of features, denoted as  $\{X_1, \dots, X_n\} \subseteq \mathcal{X}$  assuming that the number of features is  $n$ . The second subset pertains to the *class variable*  $C \subseteq \mathcal{X}$ , with the set of *class labels* consisting of  $Val(C) = \{c^1, \dots, c^k\}$ , where  $k$  represents the number of class labels.

In the context of *Naïve Bayes Classifiers*, the so-called *naive Bayes assumption* it holds, which states that the *features* are *conditionally independent* given the *class label*. Furthermore, for each *feature* node  $X_i$  with  $i \in \{1, \dots, n\}$ , holds that  $Par(X_i) = C$ . As a result, the set of *conditional independences*  $I_\rho(\mathcal{G})$  can be expressed as follows:

$$\text{For each variable } X_i : (X_i \perp \mathbf{X}_{-i} \mid C)$$

$$\text{where the set } \mathbf{X}_{-i} = \{X_1, \dots, X_n\} - \{X_i\}.$$

Given the *structure* of *Naïve Bayes Classifiers*  $\mathcal{G}(\mathcal{X}, \mathcal{E})$  and the corresponding set of *local independencies*  $I_\rho(\mathcal{G})$ , we can express the *joint probability distribution*  $P(\mathcal{X})$  as follows:

$$P(C, X_1, \dots, X_n) = P(C) \cdot \prod_{i=1}^n P(X_i \mid C)$$

Each factor  $P(X_i \mid C)$  corresponds to the *local conditional probability distributions (CPDs)* of *feature*  $X_i$ . Similar to *Bayesian Networks*, we can express the *joint probability distribution* as a product of these factors (*factorization*). In this case, the *joint probability distribution* consists of the prior distribution  $P(C)$  and the set of *CPDs*  $P(X_i \mid C)$  where each of them represents a *multinomial* distribution. This approach allows us to significantly reduce the number of parameters required for the *joint probability distribution*.

The random variables used in this context belong to the category of *categorical-discrete* random variables. Therefore, the *conditional probability distributions (CPDs)* of these variables belong to the well-known category of *Tabular CPDs*. For *continuous* variables, we employ the method of *discretization* (See Chapter 2.2). In conclusion, the *Multinomial Naïve Bayes Classifier* is used throughout this thesis.

Each CPD  $P(X_i | C)$  can be represented as a *table* where each entry corresponds to the probability of a combination of values of  $X_i$  and the *class variable*  $C$ . Additionally, each CPD characterized by a set of parameters, denoted as  $\theta_{X_i}$ .

Given a *Naïve Bayes Classifier* consists of  $n$  *binary-valued features* and a *binary-valued class variable*, we can prove that the number of independent parameters in the *factorized joint probability distribution* is  $2n + 1$ . This means that the number of parameters is linearly dependent on the number of features  $n$ , in contrast to the direct approach of *joint probability distribution* where the dependence on the number of *features* is exponential ( $2^n - 1$  parameters). Despite the strong assumptions of *independence* among *features*, the simplicity and the linear dependence on a set of *features* result in a small number of required parameters. This property makes *Naïve Bayes Classifier* applicable in many cases involving *high-dimensional feature vectors*.

Despite its simplicity and the strong assumptions of independence, which may not hold in *realistic* scenarios, the *Naïve Bayes Classifier* performs quite well even when compared to a more sophisticated *classifier* like *C4.5*, as described in detail in [12]. The success of *Naïve Bayes Classifiers* as noted in [13], can be attributed to the fact that the *classification error* is not necessarily related to the quality of *fit* of the *joint probability distribution* (appropriateness of *conditional independencies*). In other words, the *estimation error* of *joint probability* does not necessarily result in *classification errors*. The *classification* of *class labels* is not highly *sensitive* to *errors* in estimating *instance probabilities*.

Regarding the process of *Classification*, we consider a *Naïve Bayes Classifiers structure*  $\mathcal{G}(\mathcal{X}, \mathcal{E})$  with the set of *features* corresponding to  $\{X_1, \dots, X_n\} \subseteq \mathcal{X}$  and a *class variable*  $C \subseteq \mathcal{X}$  where the values of the variable are represented as  $Val(C) = \{c^1, \dots, c^k\}$ . The objective, for any  $n$ -dimensional vector  $\mathbf{x} = [x^1, \dots, x^n]$  is to determine the *class label* with the highest *posterior probability* given an  $n$ -dimensional vector. To achieve this, we select the *class label* that maximizes the following expression:

$$class\ label = \arg\max_{c^i \in Val(C)} P(C = c^i | \mathbf{x})$$

Our goal is to maximize the probability  $P(C | \mathbf{x})$ . By applying *Bayes' rule*, the probability is defined as follows:

$$P(C = c^i | \mathbf{x}) = \frac{P(\mathbf{x} | C = c^i) \cdot P(C = c^i)}{P(\mathbf{x})} \text{ for each } c^i \in Val(C)$$



### 2.3.3 Spam classification problem

Considering the previous statements, we will now present a complete example of a *Naïve Bayes Classifier*. Our focus is on determining whether an *email* is a spam or nor, based on its context. This example falls within the broader field of *Document Classification*. We assume that the *class variable*  $C$  is a *binary-valued* variable, with values represented as  $Val(C) = \{spam, non\_spam\}$ .

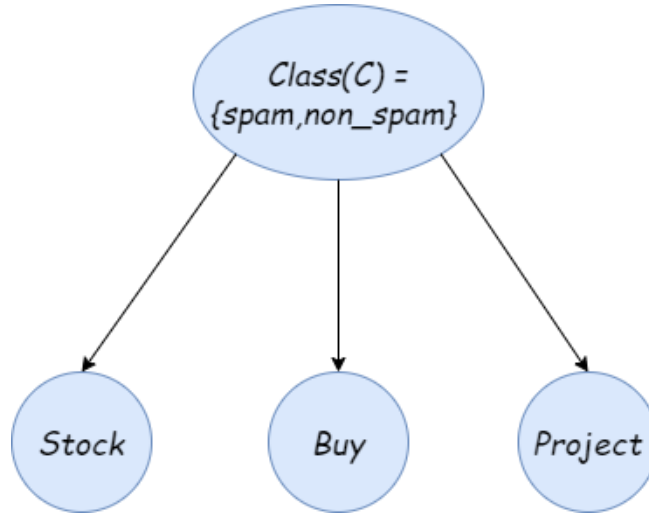


Figure 8: Naïve Bayes Classifier for spam filtering

The context of each email consists of a *bag-of-words*, where each *word* represents a *feature*. In this example, we assume three *features* corresponding to the words “Stock», «Buy” and “Project”. Each feature is accompanied by its *CPD*  $P(W_i|C)$ , where  $W_i \in \{Stock, Buy, Project\}$ . The network structure is illustrated in *Figure 8*. The set of features generally encompasses a large set of *keywords* used to define the *class label*.

Applying the *naïve Bayes assumption*, the *joint probability distribution* is defined as follows:

$$P(C, Stock, Buy, Project) = P(C) \cdot P(Stock | C) \cdot P(Buy | C) \cdot P(Project | C)$$

Our goal is to *classify* an email into one of the two *class labels*. Assuming that the email contains the words *stock*, and *buy* but does not contain the word *project*, the *input feature vector* can be represented as follows:  $x = [stock, buy, \neg project]$ . To determine the *class label*, we need to calculate the *posterior probabilities* given  $x$ , which can be defined as follows:

*For the class label spam:*

$$\begin{aligned} P(spam | x) &= P(x|spam) \cdot P(spam) \\ &= P(stock|spam) \cdot P(buy|spam) \cdot P(\neg project|spam) \cdot P(spam) \end{aligned}$$

For the class label *non\_spam*:

$$\begin{aligned} P(\text{non\_spam} | \mathbf{x}) &= P(\mathbf{x} | \text{non\_spam}) \cdot P(\text{non\_spam}) \\ &= P(\text{stock} | \text{non\_spam}) \cdot P(\text{buy} | \text{non\_spam}) \cdot P(\neg \text{project} | \text{non\_spam}) \cdot P(\text{non\_spam}) \end{aligned}$$

The *class label* is determined by comparing the probabilities  $P(\text{spam} | \mathbf{x})$  and  $P(\text{non\_spam} | \mathbf{x})$ , and selecting the one with the highest value.

## 2.4 Forward Sampling

In this section, we present the *Forward Sampling* algorithm [14] which is used to generate all the *training instances*. The *training instances* are used for the experimental evaluation of the proposed approaches.

For the generation of *training instances*, each instance consists of a *full particle* of all the variables in the *Bayesian network*  $\mathcal{G}(\mathcal{X}, \mathcal{E})$ , confirming that the setting is *supervised*.

We focus on estimating *probabilities queries*  $P(\mathbf{Y} = \mathbf{y})$  based on the *joint probability distribution*  $P(\mathcal{X})$  of the network  $\mathcal{G}$ , where  $\mathbf{Y} \subseteq \mathcal{X}$  and  $\mathbf{y} \in \text{Val}(\mathbf{Y})$ . However, we do not consider *conditional probability queries* of the form  $P(\mathbf{Y} = \mathbf{y} | \mathbf{E} = \mathbf{e})$ , where  $\mathbf{Y}, \mathbf{E} \subseteq \mathcal{X}$ ,  $\mathbf{y} \in \text{Val}(\mathbf{Y})$  and  $\mathbf{e} \in \text{Val}(\mathbf{E})$ .

We have chosen the *Forward Sampling* algorithm to generate *full particles* to estimate queries on the *joint probability distribution*. Unlike other algorithms like *Rejection Sampling* and *Likelihood Weighting* [15], which are typically used for generating *full particles* to estimate *conditional probability queries*.

In certain cases, when dealing with highly skewed distributions where only a small number of variables have negligible probability distributions, alternative algorithms like *Deterministic Search* may be necessary. However, since these cases are beyond the scope of this thesis, we have opted to utilize the *Forward Sampling* algorithm.

The algorithm is based on the concept of generating data for each variable according to its probability distribution. The first step in implementing the algorithm is to establish a *topological order* of the *DAG (Directed Acyclic Graph)* representing the Bayesian network  $\mathcal{G}$ . This ordering ensures that each node  $X_i$  is preceded before any node  $X_j$  in the sequence, meaning there are no directed paths from  $X_i$  to  $X_j$ . This step is crucial as it guarantees that data for each node's parent nodes is generated before the node itself.

Based on the ordering of nodes, the algorithm proceeds by sampling for each node according to its corresponding *CPD*  $P(X_i | \text{Par}(X_i))$ . The process is repeated for each node in the ordering. Once the process is completed for all nodes, a *particle* is generated. Below we describe the algorithm.

**Algorithm 1: Forward Sampling in BNs**

---

**Input:**

**Bayesian Network Structure**  $\mathcal{G}(\mathcal{X}, \mathcal{E})$  with  $\mathcal{X} = \{X_1, \dots, X_n\}$   
**T** number of full particles

**Output:** **T** full particles

1. Let  $X_1, \dots, X_n$  be a topological ordering of the Bayesian Network
  2. **For** 1 to **T**
  3.     Initialize an empty instantiation  $\mathbf{t}$
  4.     **Foreach** node **from** topological ordering  $X_1, \dots, X_n$
  5.         Get the values of parents of  $X_i$  from  $\mathbf{t}$
  6.         Sample  $x_i$  from  $P(X_i | \text{Par}(X_i))$
  7.         Add value  $x_i$  to  $\mathbf{t}$
  8.     **End**
  9.     **return**  $\mathbf{t} = (x_1, \dots, x_n)$
  10. **End**
- 

The process of generating a *particle* is described analytically below. We will use the *Student* (See Chapter 2.2.3) as an example. Assuming that the generated sequence of the topological order is as follows: *Difficulty(D), Intelligence(I), Grade(G), SAT(S), Letter(L)*, the algorithm operates as follows:

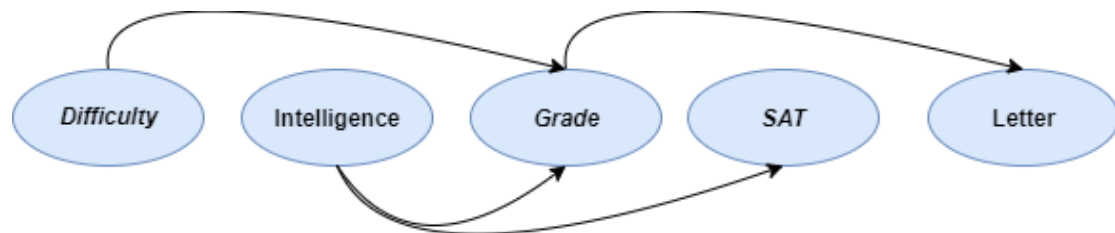


Figure 9: Topological ordering of Student Bayesian Network

1. Initially, we sample from the distribution  $P(D)$  for the *Difficulty* node, assuming  $Difficulty = d^0$
2. Next, we sample from the distribution  $P(I)$  for the *Intelligence* node, assuming  $Intelligence = i^0$ .
3. Then, we sample from the distribution  $P(G|I = i^0, D = d^0)$  for the *Grade* node, assuming  $Grade = g^2$ .
4. Subsequently, we sample from the distribution  $P(S|I = i^0)$  for the *SAT* node, assuming  $SAT = s^0$ .
5. Finally, we sample from the distribution  $P(L|G = g^2)$  for the *Letter* node, assuming  $Letter = l^0$ .
6. As a result, the particle corresponds to  $D = d^0, I = i^0, G = g^2, S = s^0, L = l^0$ .

### Analysis of the Forward Sampling algorithm

Given a dataset  $D = \{\xi[1], \dots, \xi[M]\}$  of full particles, which is generated by sampling, applying the *convergence bounds* [1, Ch. 17], we can estimate the expected value of any function using the following method:

$$E_D(f) = \frac{1}{M} \sum_{m=1}^M f(\xi[m]) \text{ where } M : \text{the size of particle set}$$

Our goal is to estimate *queries* of the form  $P(Y = \mathbf{y})$ , where the only requirement is to know the number of particles that contain the  $\mathbf{y}$ . The probability can be defined as follows:

$$P_D(\mathbf{y}) = \frac{1}{M} \sum_{m=1}^M I\{y[m] = \mathbf{y}\}$$

Where the function  $I$  refers to an indicator function, defined as:

$$\begin{cases} 1, & \text{if } y[m] \text{ contains the } \mathbf{y} \\ 0, & \text{otherwise} \end{cases}$$

Considering the complexity of the algorithm of *Forward Sampling*, assuming  $M$  corresponds to the total number of generated *full particles*,  $n = |\mathcal{X}|$  denotes to the number of nodes of the network,  $p = \max_i |Par(X_i)|$  denotes the maximum number of parents nodes for any variable, and  $d = \max_i |Val(X_i)|$  denotes the maximum number of values for any variable, then the complexity of the algorithm is  $O(M \cdot n \cdot p \cdot \log d)$ . This considers that the sampling for any variable is  $O(\log d)$  and the indexing time required to retrieve the values of parents nodes for any distribution using the appropriate data structure is  $O(p)$ .

### Sampling distribution

Regarding the scope of this thesis, our focus is on *categorical* random variables. We assume a *multinomial* distribution  $P(X)$  with values  $Val(X) = \{x^1, \dots, x^k\}$ , which is defined by the parameters  $\theta_1, \dots, \theta_k$ , respectively. The sampling process (*Figure 10*) can be summarized in the following three steps:

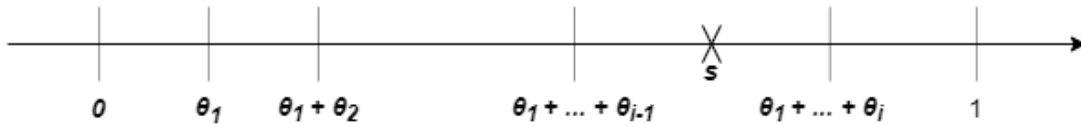


Figure 10: Sampling of a multinomial distribution

1. Initially, we generate a uniformly sampled value  $s$  from the interval  $[0,1]$ .
2. Then, we divide the interval into  $k$  sub-intervals:  $[0, \theta_1), [\theta_1, \theta_1 + \theta_2), \dots$  where  $i$  sub-interval is defined as  $[\sum_{j=1}^{i-1} \theta_j, \sum_{j=1}^i \theta_j)$
3. If the sampled value  $s$  falls within the  $i - th$  sub-interval, the generated value is  $x^i$  (the interval boundaries can be determined using the method of binary search in  $O(\log k)$  time).

## Estimation Quality Analysis

The quality of the estimation obtained from the sampling method, which measures the proximity to the *joint ground truth distribution* of *Bayesian Network*  $\mathcal{G}$ , is largely determined by the number of generated *particles*. In cases where we aim to quantify the *error guarantees* of the estimation, we can achieve using the following theorem.

We assume that  $P(Y = y)$  represents the *ground truth joint distribution*, derived from the *true values* of parameters,  $P_D(Y = y)$  denotes the distribution obtained from the set of generated *particles*, which is equivalent to the distribution obtained through the *MLE* algorithm.  $M$  represents the number of *particles*.

Theorem 1[1, Corollary 12.2]: Assuming  $M \geq \frac{3 \ln(\frac{2}{\delta})}{P(y)\epsilon^2}$ , we can provide an  $(\epsilon, \delta)$ -approximation of the *joint ground truth* distribution as follows:

$$(1 - \epsilon) \leq \frac{P_D(y)}{P(y)} \leq (1 + \epsilon) \text{ with probability } 1 - \delta$$

To guarantee an  $\epsilon$  *relative error* with an *error probability* up to  $\delta$  from the *joint ground truth* distribution, the size of the sample depends logarithmically on  $\frac{1}{\delta}$ , quadratically on  $\frac{1}{\epsilon}$  and linearly on  $1/P(y)$ . However, a problem with this approach arises when the distribution  $P(y)$  is unknown, making it difficult to determine the appropriate sample size for achieving a “good” estimation quality. While there are other *error guarantees* [1, Ch. 17] for the estimation, our main concern is the *relative error guarantees*.

An extension of the previous theorem involves assuming the prior learning of the parameters of *local CPDs* of *Bayesian Network*  $\mathcal{G}$ , which is defined as follows:

Theorem 2[1, Corollary 17.3]: Under the assumptions  $P(X_i | Par(X_i)) \geq \lambda \forall i, X_i, Par(X_i)$  and  $M \geq \frac{(1+\epsilon)^2}{2\lambda^2(d+1)\epsilon^2} \log \frac{nJ^{d+1}}{\delta}$ , the following equation holds:

$$e^{-n\epsilon} \leq \frac{P_D(y)}{P(y)} \leq e^{n\epsilon} \text{ with probability } 1 - \delta$$

where  $J = \max_{i=1}^n J_i$ ,  $\mu \epsilon J_i = |Val(X_i)|$ ,  $d = \max_{i=1}^n |Par(X_i)|$  και  $n = |\mathcal{X}|$ .

In this scenario, the required number of *particles* to ensure a “good” estimation depends not only on  $\epsilon, \delta$  but also on  $J, d, n$  which take into account the structure of *Bayesian Network*  $\mathcal{G}$ .

To ensure the correctness and quality of the estimation, it is necessary to bound, if feasible the values of *joint truth probability* of *queries* (e.g., quantifying  $P(Y) \geq 0.1$  for all the queries). This allows for determining an appropriate size of the query set to control the distribution.

## 2.5 Learning parameters

Our goal is to learn the *parameters* of the *joint probability distribution* of the *Bayesian Network Structure*  $\mathcal{G}(\mathcal{X}, \mathcal{E})$ . Given a known network structure known, we attempt to learn the parameters from the dataset  $D$ . To accomplish this, we utilize the *Maximum Likelihood Estimation (MLE)* algorithm, which will be discussed and analyzed below. The dataset consists of instantiations of *full assignments* confirming the *supervised* setting.

The objective is to construct a model  $M$ , which falls under the category of *parametric models* and is characterized by a parameter set  $\theta$ . Our primary concern is that the obtained *joint probability distribution* closely matches the distribution of the dataset. Therefore, the selection of parameters is defined in such a way that the available *training instances* achieve the highest probability, making them the *most probable*.

### 2.5.1 Maximum Likelihood Estimation (MLE)

Assuming a Bayesian Network Structure  $\mathcal{G}(\mathcal{X}, \mathcal{E})$  with the set of variables corresponds to  $\mathcal{X} = \{X_1, \dots, X_n\}$ .

Given a dataset  $D = \{\xi[1], \dots, \xi[M]\}$  (*training dataset*) consisting of *independent and identically distributed (i.i.d.)* samples from the *joint probability distribution*  $P(\mathcal{X})$  of  $\mathcal{G}$  (See Chapter 2.4). Each  $\xi[i]$  for  $i \in \{1, \dots, M\}$  represents an *instantiation of full assignments* of  $\mathcal{X}$ . Our objective is to determine the set of parameters  $\theta$  that define the required parameters for each *CPD* of each variable.

The first step in defining the *likelihood function* of parameters  $\theta$  given the dataset  $D$ , is as follows:

$$\text{likelihood} : L(\theta : D) = P(D|\theta)$$

Similarly, we can define the *log-likelihood* function which is commonly used (usually we calculate  $\log P(\xi|\theta)$  to avoid *overflow* problems), as follows:

$$\text{log-likelihood} : l(\theta : D) = \log P(D|\theta)$$

It is worth mentioning that the *negated form* of the *log-likelihood* is referred to as the *loss function*, which measures the *loss* given the learning parameters for any *instance*  $\xi$ .

$$\text{loss function} : \text{loss}(\xi|\theta) = -\log P(\xi|\theta)$$

Regarding the fact that the dataset  $D = \{\xi[1], \dots, \xi[M]\}$  consists of *i.i.d instances*, we can redefine the *likelihood function*, as follows:

$$L(\theta : D) = P(D|\theta) = \prod_{i=1}^M P(\xi[i] : \theta)$$

To maximize the *likelihood function*, we want to select the parameters  $\theta$  that maximize the *likelihood function*. Assuming  $\hat{\theta}$  represents the set that maximizes the *likelihood function*, the following equation holds:

$$\text{Maximum Likelihood Estimation: } L(\hat{\theta} : D) = \max_{\theta \in \Theta} L(\theta : D)$$

where  $\Theta$  denotes the hypothesis space

Exploiting the *Bayesian network structure*, we can define the *likelihood function*, as follows:

$$\begin{aligned}
L(\theta : D) &= P(D|\theta) = \prod_{i=1}^M P_G(\xi[i] : \theta) \\
&= \prod_{i=1}^M \prod_{j=1}^n P(x_i[m] | Par(X_i)[m] : \theta) \\
&= \prod_i \left[ \prod_m P(x_i[m] | Par(X_i)[m] : \theta) \right]
\end{aligned}$$

Each factor in brackets denotes the *condition likelihood function* of the random variable  $X_i$  given  $Par(X_i)$ . Assuming  $\theta_{X_i}$  denotes the set of parameters that defines the *CPD*  $P(X_i|Par(X_i))$ , the following equation holds:

$$L(\theta : D) = \prod_i L_i(\theta_{X_i} : D)$$

Each factor denotes the *local likelihood* of the random variable  $X_i$  and is defined as follows:

$$L_i(\theta_{X_i} : D) = \prod_m P(x_i[m] | Par(X_i)[m] : \theta_{X_i})$$

The latter property is known as the *global decomposability* of the *likelihood function*. Assuming that each  $\theta_{X_i}$  is independent of  $\theta_{X_j}$  for each  $i \neq j$  with  $i, j \in \{1, \dots, n\}$ , we can maximize each *local likelihood function* independently from the others and combine them to obtain the *MLE*. This property, along with the property of *local decomposability* constitutes the ideal scenario to apply in a *distributed* environment.

Given a dataset  $D = \{\xi[1], \dots, \xi[M]\}$  and the corresponding *Bayesian Network Structure*  $\mathcal{G}(\mathcal{X}, \mathcal{E})$ , assuming  $\theta_{X_i}$  is independent of  $\theta_{X_j}$  for each  $i \neq j$  with  $i, j \in \{1, \dots, n\}$  and  $\hat{\theta}_{X_i}$  maximizes the *local likelihood function*  $L_i(\theta_{X_i} : D)$  of random variable  $X_i$  then the set  $\hat{\theta} = \{\hat{\theta}_{X_1}, \dots, \hat{\theta}_{X_n}\}$  maximizes the *likelihood function*  $L(\theta : D)$ .

Beyond the property of *global decomposability* of the *likelihood function*, the property of *local decomposability* also holds. Given the *tabular CPD*  $P(X_i | Par(X_i))$  of the random variable  $X_i$  with  $Par(X_i) = \mathbf{U}$ , which is our primary concern, we can redefine the *local likelihood function* in the following manner:

$$\begin{aligned}
L_i(\theta_{X_i} : D) &= \prod_m P(x_i[m] | Par(X_i)[m] : \theta_{X_i}) \\
&= \prod_m \theta_{x_i[m] | Par(X_i)[m]} \\
&= \prod_{\mathbf{u} \in Val(\mathbf{U})} \left[ \prod_{x \in Val(X_i)} \theta_{x_i | \mathbf{u}}^{M[u, x_i]} \right]
\end{aligned}$$

Where  $M[\mathbf{u}, x_i]$  represents the counter of the combination of values  $X_i = x_i$  and  $Par(X_i) = \mathbf{u}$  in  $D$ .

Considering the latter equation, we achieve to concentrate all the references of the parameter  $\theta_{x_i|\mathbf{u}}$  for each  $x_i \in Val(X_i)$  together. We attempt to maximize the parameters  $\theta_{x_i|\mathbf{u}}$  taking into account the restriction  $\sum \theta_{x_i|\mathbf{u}} = 1$  for each  $\mathbf{u} \in Val(\mathbf{U})$ . The latter property demonstrates the property of *local decomposability*, now we can maximize the parameters  $\theta_{x_i|\mathbf{u}}$  for each  $\mathbf{u} \in Val(\mathbf{U})$  independently from the others.

### Maximum Likelihood Estimate

We can prove that the *maximum likelihood estimate*  $\hat{\theta}_{X_i}$  of  $\theta_{X_i}$  of *tabular CPD*  $P(X_i | Par(X_i))$ , which predisposes that the *likelihood function* is derivable, defined as follows:

$$\text{Maximum likelihood estimate : } \hat{\theta}_{X_i} = \frac{C_i(x_i, x_i^{par})}{C_i(x_i^{par})}$$

for each  $x_i \in Val(X_i), x_i^{par} \in Par(X_i)$

Where  $C_i(x_i, x_i^{par})$  denotes the number of instances  $(X_i = x_i, Par(X_i) = x_i^{par})$  in  $D$  and  $C_i(x_i^{par})$  denotes the number of instances  $(Par(X_i) = x_i^{par})$  in  $D$ . To obtain the maximum likelihood estimate the only requirement is to maintenance the counters  $C_i(x_i, x_i^{par})$  and  $C_i(x_i^{par})$  for each random variable  $X_i$  of the network and for each  $x_i \in Val(X_i)$  and  $x_i^{par} \in Val(Par(X_i))$ . *Algorithm 2* summarizes the *MLE* algorithm.

---

#### Algorithm 2: Maximum likelihood Estimation (MLE)

---

**Input:**

Bayesian Network Structure  $\mathcal{G}(\mathcal{X}, \mathcal{E})$

$D = \{ \xi[1], \dots, \xi[M] \}$  of full assignments

**Output:**

$\hat{\theta} = \{ \hat{\theta}_{x_1}, \dots, \hat{\theta}_{x_n} \}$

1. // Count
  2. **Foreach**  $\xi[1]$  **from**  $D$
  3.     **Foreach**  $X_i$  **from**  $\mathcal{X}$
  4.         Increment count( $x_i, x_i^{par}$ )
  5.         Increment count( $x_i^{par}$ )
  6.     **End**
  7. **End**
  - 8.
  9. // Estimation
  10. **Foreach**  $X_i$  **from**  $\mathcal{X}$
  11.     **Return**  $\hat{\theta}_{X_i} = F_i(x_i, x_i^{par}) / F_i(x_i^{par})$
  12. **End**
-



Given a *Bayesian Network Structure*  $\mathcal{G}(\mathcal{X}, \mathcal{E})$ , the *MLE* of the *joint probability distribution* can be defined as follows:

$$P(\mathcal{X}) = \prod_{i=1}^n \theta_{x_i} = \prod_{i=1}^n \frac{C_i(x_i, x_i^{par})}{C_i(x_i^{par})}$$

### **MLE consistency**

The algorithm *MLE* also holds the property of *consistency* [1, Theorem 16.1]. Given a sequence of *i.i.d* samples  $D_M = \{\xi[1], \dots, \xi[M]\}$  from the distribution  $P(\mathcal{X})$ , then the property of *consistency* demonstrates as follows:

$$\lim_{M \rightarrow \infty} P_{D_M}(A) = P(A)$$

For large datasets, the *empirical distribution* approaches the obtained distribution of the *MLE* algorithm that is close enough to the initial distribution  $P(\mathcal{X})$  with high probability, specifically the probability converges to 1 as  $M \rightarrow \infty$ .

### **Data fragmentation and overfitting**

One of the restrictions of the *learning parameters* of a *Bayesian Network* given *training instances*, constitutes the fact that an increase in the number of parents nodes for any node leads to an exponential increase in *parents assignments* of the node and simultaneously leads to an exponential decrease to the number of data for each *parent assignment*. This phenomenon is denoted as *data fragmentation*, where the dataset is divided into a large number of small subsets. In case, the size of the dataset is limited then someone subset is also limited and therefore this scenario can be led to *overfitting* (a large number of zeros in distribution).

## 2.6 Laplace Smoothing

An issue during the process of the *MLE* algorithm in both *Bayesian Network* and *Naïve Bayes Classifier* arises when the size of *training instances* is very limited (of course, this can also occur when the size of the dataset is large but combined usually with *highly skewed distributions*), is that the dataset may not be representative of the *joint probability distribution*, as it may lack data for certain variables in the network, particularly those with small parameter values  $\theta_{x_i}$ .

When we encounter situations where there is no data available for specific values of network nodes, estimating queries on the joint probability distribution becomes challenging. Treating these missing parameters as zero does not accurately reflect the truth, as it does not necessarily mean that these values do not exist. The problem becomes apparent when we have no data for a *non-zero parameter*  $\theta$ .

Attempting to answer *user-posed queries* involving values or a combination of values that have small probabilities and small  $\theta$  values (usually occurring when the queries refer to *highly unlikely queries, queries quite small joint probability distribution* that is unlikely to happen) leads to a zero estimation. Consequently, this results in a non-pragmatic scenario where the estimation of *joint probability distribution* does not reflect the truth due to the absence of data for certain parameters.

To address this problem, one effective approach is the widely known method of *Laplace smoothing*. *Laplace smoothing* can be applied in both *zero* and *non-zero* parameters although our primary focus is on the *zero* parameters. The method works as follows: for each *zero parameter*, we add  $\lambda$  (*pseudocount*) occurrences of the parameter value regardless of whether the value has been observed in the dataset. Since the parameter  $\theta_{X_i}$  represents a probability, it is necessary to normalize it by the number of values of variable  $X_i$  and multiplying it by the value of parameter  $\lambda$ . Assuming  $\theta_{X_i}$  corresponds to the *CPD*  $P(X = x_i | Par(X_i))$  of the random variable  $X_i$ , the following equation holds when *Laplace Smoothing* is applied:

*Maximum likelihood estimate with Laplace Smoothing*

$$P(X = x_i | Par(X_i)) = \frac{\text{count}(X = x_i \text{ and } Par(X_i)) + \lambda}{\text{count}(Par(X_i)) + \lambda \cdot \text{Val}(X_i)} \text{ where } \lambda > 0$$

We can demonstrate the application of *Laplace Smoothing* using the following examples. Assuming a *biased coin* with two faces  $\{Head(H), Tail(T)\}$  and one toss which resulted in *Head*. Estimating the parameters given the result of the previous toss and the *MLE* algorithm, then:

*Maximum Likelihood estimate*

$$p(H) = \frac{1}{1} = 1, \quad p(T) = \frac{0}{1} = 0$$

Regarding both of estimations, none of them are consistent. This example demonstrates the problem of *overfitting* of the *MLE* algorithm which confirms the fact when the size of the dataset is small and the number of parameters is quite large then the *MLE* leads to *overfitting* (most of the parameters are considered as zero which does not reflect the reality).

The following equation holds when *Laplace Smoothing* is applied and assuming  $\lambda = 1$ :

*Maximum Likelihood estimate with Laplace Smoothing*

$$p(H) = \frac{1 + 1}{1 + 2} = \frac{2}{3}, \quad p(T) = \frac{0 + 1}{1 + 2} = \frac{1}{3}$$

The application of *Laplace Smoothing* leads to a more pragmatic scenario and simultaneously results in preventing the problem of *overfitting*, even when the dataset is quite small.

By varying the value of  $\lambda$  parameter, we can define the added *smoothing-regularization* on parameter. The larger the value of  $\lambda$ , the more the distribution approaches the uniform distribution. Regarding *zero parameters*, both the numerator and denominator of the previous equation are zero regardless of the value of  $\lambda$ . In this case, the distribution accurately approaches the uniform distribution.

Although, in most cases, the numerator is zero. Considering the experimental evaluation, the value of  $\lambda$  where the quality of the obtained estimation is quite “good” is achieved when  $\lambda = 1$ .

Another advantage of this method is the effect of the parameter based on the size of the dataset. As the dataset size increases, the influence of  $\lambda$  diminishes, which is desirable since a larger dataset is more reliable. When considering a zero parameter, as soon as data about it is detected, it is converted into a *non-zero* parameter. This marks the point at which the usage of the *Laplace Smoothing* method is discontinued. In the context of this thesis, the *Laplace Smoothing* method is exclusively applied to zero parameters in both *Bayesian Networks* and *Naïve Bayes Classifier*. Otherwise, the parameter estimation relies solely on the dataset.

## 2.7 Distributed Continuous Model

This section describes the mode used throughout this thesis, namely the *Continuous Distributed Monitoring Model* [16]. The model represents an intersection between the *communication model*, where  $k \geq 2$  sites exist and our objective is the *one-shot computation* of a specific function, and the *data stream model* where  $k = 1$  (only one site) and our objective is the *continuous monitoring* of a specific function.

This model simulates most of the *learning tasks over big data streaming* systems, which involve enormous volumes of datasets (on the order of *Petabytes*). These datasets are streaming and distributed and we aim to ensure *real-time* responses.

The *Continuous Distributed Monitoring Model* consists of a set of *observers*, each of which tracks only one part of the observations. The objective is to estimate a function over the union of the observation for each time.

For instance, the function can be the counting of the observation set (*count-tracking problem* – which is our primary concern), where the function represents a *linear* and *monotonic increasing function*. Of course, there are more complex functions, such as the *second frequency moment*  $F_2$  which can be applied to various situations, such as *estimating join sizes*.

Our goal is to minimize the *communication cost* among observes (*communication-efficient*). To achieve this, it is usually necessary to provide an approximate estimation of the corresponding function (*approximate answers*).

We assume that the observation set for each observer represents a *high-speed, high-volume, and high-throughput data stream* so approaches such as *periodic polling or centralization of streams* may be infeasible.

Specifically, we assume  $k$  sites (or workers) and the set of sites denoted as  $S = \{S_1, \dots, S_k\}$ . Each site  $S_i$  receives a stream of observations with varied rates probably. Also, we assume that the  $A_i(t)$  corresponds to a *multiset of elements (bag of elements)* that have been received from the site  $S_i$  up to the time  $t$  and  $A(t) = \uplus_i A_i(t)$  with  $i \in \{1, \dots, k\}$  which corresponds to the union of stream of all sites, with  $\uplus$  denotes *multiset addition*.

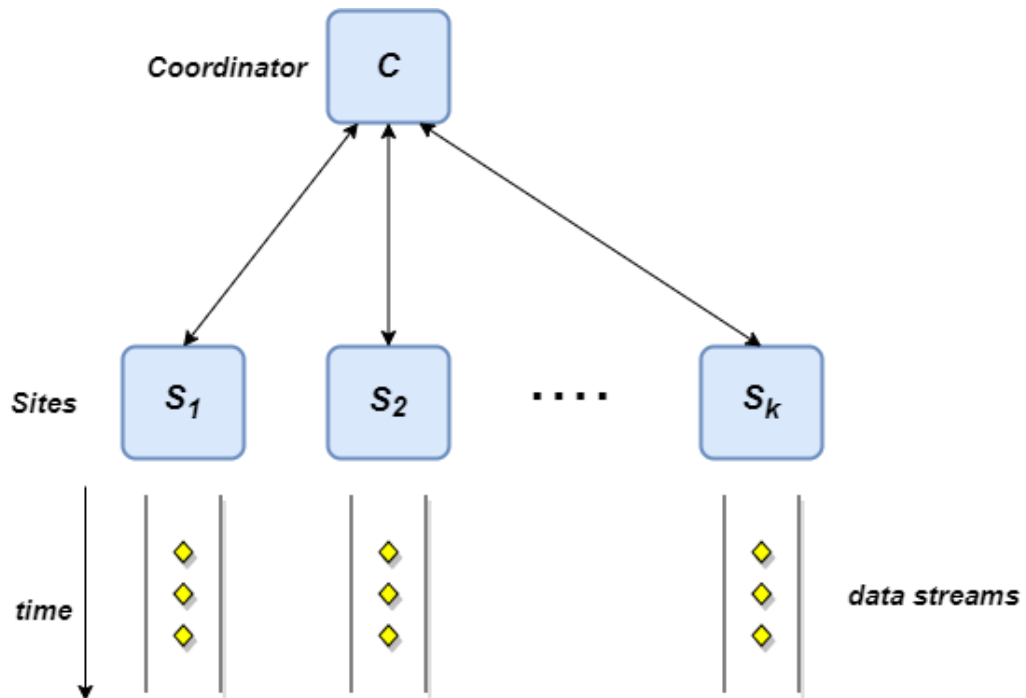


Figure 11: Distributed Continuous Model

Beyond the individual sites, there exists a designated site known as the *Coordinator*. The *Coordinator* communicates directly with each site  $S_i$  and therefore, there is a *two-way communication channel* between the *Coordinator* and each one of the  $k$  sites (Figure 11). Although there is no direct communication channel among the sites themselves, this does not pose a limitation. By utilizing the *Coordinator* as an intermediate node, sites can communicate through the *Coordinator*. The *downstream communication cost* refers to the number of messages sent from the sites to the *Coordinator*, while the *upstream communication cost* refers to the number of messages sent from the *Coordinator* to the sites.

Our goal is to monitor the function of the union of the data streams. We expect the *Coordinator* to continuously maintain the value  $f(A(t))$  for any given time  $t$ . For instance, in the case of the *count-tracking problem*, the function can be defined as follows:  $f(A(t)) = |A(t)|$ .

Our primary focus is on *value monitoring*, where at any time  $t$  we aim to provide an estimation  $\hat{f}(A(t))$  of the function  $f(A(t))$  while also ensuring *error guarantees*. We seek an  $\varepsilon, \delta$ -*approximation* of the function  $f(A(t))$  for any given time, which can be defined as follows:

$$(1 - \varepsilon) \cdot f(A(t)) \leq \hat{f}(A(t)) \leq (1 + \varepsilon) \cdot f(A(t)) \text{ with probability } 1 - \delta, \forall t$$

The analysis of the *MLE* algorithm reveals a crucial component: the maintenance of the necessary counters for each *CPD* of the network, considering the *Continuous Distributed Model*. We refer to these counters as *distributed counters* (See Chapter 4.1.1). Additionally, the maintenance of *distributed counters* is accompanied by minimal communication cost (*communication efficient*). This problem is widely known as the *count-tracking* problem and has been discussed analytically in [17] and [20]. The *count-tracking* problem can be viewed as the maintenance of  $F_1$  *frequency moment*.

## Chapter 3:

### Problem statement

#### 3.1 Problem definition

The general idea of the problem is to design a system that is capable of maintaining the necessary *distributed counters* with minimal communication cost (*communication-efficient*) while providing *error guarantees* for the maintained model. These systems consist of *massive, dynamic, high-throughput, and distributed data sources*, where centralizing datasets may be *inefficient and infeasible* due to the *enormous communication cost*.

We aim to design a system capable of effectively maintaining a graphical model over distributed and streaming datasets, with a specific emphasis on providing an accurate approximation of the Maximum Likelihood Estimate (MLE) at any given time.

The only requirement to obtain the *maximum likelihood estimate* is to have the *counters* for the parameters defined for each *CPD* in the network. We aim to maintain a collection of *distributed counters* necessary for estimating the MLE. However, in a distributed environment, utilizing *EXACT counters* (See Chapter 4.1.1), where each site sends an *update* message to the Coordinator whenever receives an event, communication cost becomes the bottleneck of the entire system.

To address this problem, a *tradeoff* between *efficiency* and estimation *quality* is needed. Instead of using *Exact MLE*, which incurs substantial *communication cost*, it is preferable to utilize an *accurate approximate* of *MLE* with lower *communication cost*. By leveraging the property of independence among the parameters and appropriately allocating the available error budget to different and independent parameters (known as *slack allocation problem*), that firstly provides the error of *joint probability distribution* remains within acceptable bounds and secondly, the *communication cost* is as little as possible, then we can provide an *accurate approximate* of *MLE* with minimal communication cost.

Using both the first approach which involves a collection of *approximate distributed counters* treating each counter individually, and the proposed approach utilizing the method of *Functional Geometric Protocol (FGM)*, we can achieve our objective. This involves the utilization of the *BASELINE, UNIFORM, and NON\_UNIFORM* [18] algorithms, which define the appropriate way to divide the available *error budget* and are discussed and analyzed below.

Given a *Bayesian Network structure*  $\mathcal{G}(\mathcal{X}, \mathcal{E})$ , the probability  $\hat{P}(\mathcal{X})$  represents the *EXACTMLE* of the *joint probability distribution* of the network  $\mathcal{G}$ . Furthermore, given an *approximation factor*  $\varepsilon$ , where  $\varepsilon \in [0, 1]$ , an  $\varepsilon$ -*approximation* of *MLE* is the *joint probability distribution*  $\tilde{P}(\mathcal{X})$  for each  $x \in Val(\mathcal{X})$ , if and only if the following equation holds:

$\varepsilon$  – *approximation of MLE*

$$e^{-\varepsilon} \leq \frac{\tilde{P}(x)}{\hat{P}(x)} \leq e^{\varepsilon}$$

Similarly, given an additional parameter  $\delta$ , where  $\delta \in [0,1]$ , then  $\tilde{P}(\mathcal{X})$  refers to an  $(\varepsilon, \delta)$ -approximation, if and only if the following equation holds:

$$(\varepsilon, \delta) - \text{approximation of MLE}$$

$$e^{-\varepsilon} \leq \frac{\tilde{P}(\mathbf{x})}{\hat{P}(\mathbf{x})} \leq e^{\varepsilon} \text{ with probability } 1 - \delta$$

Our objective is to maintain an  $(\varepsilon, \delta)$ -approximation of MLE, at any given time. Given a Bayesian Network structure  $\mathcal{G}(\mathcal{X}, \mathcal{E})$ , the EXACTMLE of the joint probability distribution  $\tilde{P}(\mathcal{X})$  (See Chapter 2.5) can be defined as follows:

$$\tilde{P}(\mathcal{X}) = \prod_{i=1}^n \theta_{X_i} = \prod_{i=1}^n \frac{C_i(x_i, x_i^{par})}{C_i(x_i^{par})}$$

Where the  $C_i(x_i, x_i^{par})$  represents the number of events ( $X_i = x_i, Par(X_i) = x_i^{par}$ ), and  $C_i(x_i^{par})$  represents the number of events ( $Par(X_i) = x_i^{par}$ ).

Assuming  $A_i(x_i, x_i^{par})$ ,  $A_i(x_i^{par})$  represents the approximate version of the distributed counters  $C_i(x_i, x_i^{par})$ ,  $C_i(x_i^{par})$ , respectively, for any input vector  $\mathbf{x} = [x_1, \dots, x_n]$ , and given an approximation factor  $\varepsilon$ , we want the following statement to hold at any time:

$$e^{-\varepsilon} \leq \frac{\tilde{P}(\mathbf{x})}{\hat{P}(\mathbf{x})} = \prod_{i=1}^n \left( \frac{A_i(x_i, x_i^{par})}{C_i(x_i, x_i^{par})} \cdot \frac{C_i(x_i^{par})}{A_i(x_i^{par})} \right) \leq e^{\varepsilon}$$

### Definition of the Classification Problem

Beyond the scope of the Bayesian Networks, we focus on a special case of them, namely the Naïve Bayes Classifier. Given the collection of the approximate distributed counters, we need to redefine the Classification problem. Assuming a Naïve Bayes Classifier  $\mathcal{G}(\mathcal{X}, \mathcal{E})$  and an approximation factor  $\varepsilon$ , for any input vector  $\mathbf{x}$  (or known as evidence), the class label  $c^i$  resolve the Bayesian classification problem with  $\varepsilon$  error, if:

$$\hat{P}(C = c^i | \mathbf{x}) \geq (1 - \varepsilon) \cdot \max_{c^i \in Val(C)} \hat{P}(C = c^i | \mathbf{x})$$

We choose the class label  $c^i$  that the conditional probability is close to the highest probability. Given an  $(\varepsilon, \delta)$ -approximation of the MLE of the joint probability distribution of a Naïve Bayes Classifier, the class label  $c^i$  resolves the Bayesian Classification problem with  $\varepsilon$  error [18, Lemma 13].

Consequently, at any time we want an  $(\varepsilon, \delta)$ -approximation of the MLE of the joint probability distribution of the Bayesian Network over the distributed continuous model while using the minimal communication cost (communication efficient).

## 3.2 The general approach

This section describes the general approach used in both the first approach (See Chapter 4.1) and the second approach (See Chapter 4.2). The general approach can be summarized in the following four algorithms.

The first algorithm is the *INITIALIZATION* (Algorithm 1) algorithm. This algorithm is used by the sites, including the Coordinator. It is executed first and is responsible for *initializing* all the *approximated distributed counters* required from each *CPD* of the *Bayesian Network*, which is necessary for estimating the *MLE* of the *joint probability distribution* of the network. The first parameter refers to the *Bayesian Network* to be used, the second parameter refers to the type of the *approximate distributed counter* (*type\_counter*) that will be used regarding the first approach, and refers to the type of the *frequency vector* that will be used considering the second approach. The last parameter, *schema\_error*, specifies the algorithm to be used and defines the appropriate value of the *approximation factor*  $\epsilon$  for each of the available *counters*, choosing from the *BASELINE*, *UNIFORM*, and *NON\_UNIFORM* algorithms.

---

### Algorithm 1: *INITIALIZATION*

---

#### Input:

*type\_counter*: corresponds to one of the four available algorithms (*RANDOMIZED*, *DETERMINISTIC*, *EXACT*, *CONTINUOUS*)

*schema\_error*: corresponds to one of the three available algorithms (*BASELINE*, *UNIFORM*, *NON\_UNIFORM*)

*Bayesian Network Structure*  $\mathcal{G}(\mathcal{X}, \mathcal{E})$ , where  $\mathcal{X} = \{X_1, \dots, X_n\}$  are the nodes of the *Bayesian Network*

1. **Foreach**  $X_i$  **from**  $\mathcal{X}$
  2.     // Generate all the distributed counters from the node  $X_i$  based on the type of counter
  3.     *GENERATE\_DIST\_COUNTERS* (*type\_counter*,  $X_i$ )
  - 4.
  5.     // Initialize the distributed counters based on the error schema
  6.     **Foreach**  $A_i(x_i, x_i^{par})$  **from**  $X_i \cap Par(X_i)$
  7.         *INITIALIZE\_DIST\_COUNTER* ( $A_i(x_i, x_i^{par})$ , *schema\_error*)
  8.     **Foreach**  $x_i^{par}$  **from**  $Par(X_i)$
  9.         *INITIALIZE\_DIST\_COUNTER* ( $A_i(x_i^{par})$ , *schema\_error*)
  10. **End**
- 

The second algorithm is responsible for managing the events and is called the *UPDATE* algorithm (*Algorithm 2*). It is exclusively used by the sites to handle the received *data stream* of *training instances*. Its objective is to manage the corresponding *input vector* while providing the necessary increments to the *distributed counters*.



### Algorithm 2: UPDATE

---

**Input:**  $\mathbf{x} = \langle x_1, \dots, x_n \rangle$  is a full particle of nodes from the Bayesian Network

```
1. Foreach  $x_i$  from  $\mathbf{x}$ 
2.
3.   If ( $X_i$  has parents) then
4.     // Get the counters to correspond to the parents' nodes of  $X_i$ 
5.     GET_COUNTER ( $x_i^{par}$ )
6.     INCREMENT_DIST_COUNTER ( $x_i^{par}$ )
7.   End
8.
9.   // Get the counters to correspond to the  $X_i$  node
10.  GET_COUNTER ( $x_i$ )
11.
12.  // Increment the distributed counters for the  $X_i$  node
13.  INCREMENT_DIST_COUNTER ( $x_i$ )
```

---

The algorithm responsible for handling the messages between the *sites* and the *Coordinator* and vice versa is called the *HANDLE\_MESSAGE algorithm (Algorithm 3)*. It is used by both sides, the *sites* while receiving messages from the *Coordinator* and the *Coordinator* while receiving messages from the *sites* (*two-way communication channel*). Depending on the type of message (*type\_message*), it performs the corresponding action. The action usually involves either sending a message (such as *broadcasting* a value from the *Coordinator* to *sites*) or *updating* the value of a variable. Both the type of message and the subsequent sequence of actions are defined by the corresponding approach.

### Algorithm 3: HANDLE\_MESSAGE

---

**Input:**

*type\_message*: corresponds to one of the available types of messages  
*message*: corresponds to a message

```
1. // Handle the message based on the type of it
2. HANDLE_MESSAGE (type_message)
3. UPDATE_COUNTERS ()
```

---

Finally, the algorithm responsible for estimating *probability queries* is the *ESTIMATE algorithm (Algorithm 4)*, which is used exclusively by the *Coordinator*. When receiving *queries*, the *Coordinator* only needs to estimate the value of the *joint probability distribution* of the query.

**Algorithm 4: ESTIMATE**

---

**Input:**  $\mathbf{x} = \langle x_1, \dots, x_n \rangle$  is a full particle of nodes from the Bayesian Network

**Output:** Return the estimated probability from the event

1. **Foreach**  $x_i$  **from**  $\mathbf{x}$
  2.     // Get the counters for the parameter  $\theta_{x_i}$
  3.     GET\_COUNTER( $x_i, x_i^{par}$ )
  4.     GET\_COUNTER( $x_i^{par}$ )
  - 5.
  6.     // Estimate the parameter for the specific  $X_i$
  7.     ESTIMATE\_PAR ( $\theta_{x_i}$ )
  8.     UPDATE\_EST\_PROBABILITY ( $\theta_{x_i}$ )
  9. **End**
  10. **Return** EST\_PROBABILITY
-

## Chapter 4:

### Problem analysis

#### 4.1 A first approach

The first approach to the problem is the maintenance of a collection of *approximate distributed counters*, treating each counter *individually*. We need to define the appropriate value of the *approximation factor*  $\epsilon$  among the given *approximate counters*, aiming to achieve an  $\epsilon$ -*approximation* of the *MLE of joint probability distribution*, while using the minimum communication cost.

This can be achieved using the three algorithms *BASELINE*, *UNIFORM*, and *NON\_UNIFORM* [18], which define the way to divide the *approximation factor*  $\epsilon$  among the *approximate distributed counters*. The implemented type of *approximate distributed counter* is the *RANDOMIZED counter* and the *DETERMINISTIC counter*, which are discussed and analyzed below.

#### **EXACT counters**

The first solution to provide *continuous maintenance* of the *MLE*, which also requires the *continuous maintenance* of corresponding *counters*, is the *EXACT counters* (*Algorithm 1,2*). This way, the Coordinator maintains the *EXACTMLE* of the *joint probability distribution* at any time, but the communication cost quickly becomes the *bottleneck* of the entire system.

---

#### **Algorithm 1:** *Exact Counter (EXACT) - Worker*

**Input:**  $\mathbf{x} = \langle x_1, \dots, x_n \rangle$  is a full particle of nodes from the Bayesian Network

---

1. **If** (*received* = ``INPUT``)
  2.     **Foreach**  $x_i$  **from**  $\mathbf{x}$
  3.         **If** ( $X_i$  has parents) **then**
  4.             // Get the counters to correspond to the parents' nodes of  $X_i$
  5.             `GET_COUNTER` ( $x_i^{par}$ )
  6.             `INCREMENT_COUNTER` ( $x_i^{par}$ )
  7.             `SEND_MESSAGE` ( $x_i^{par}$ , ``INCREMENT``)
  8.         **End**
  - 9.
  10.         `GET_COUNTER` ( $x_i$ )
  11.         `INCREMENT_COUNTER` ( $x_i$ )
  12.         `SEND_MESSAGE` ( $x_i$ , ``INCREMENT``)
  13.     **End**
  14. **End**
- 

The idea behind the *EXACT counter* is that for each increment on the counter, the site also sends a message to the Coordinator (*Algorithm 1*) to update the value of the counter

(Algorithm 2). However, this approach leads to a loss of the benefit of the distributed environment, as the communication cost quickly becomes substantial.

Given a Bayesian Network Structure  $\mathcal{G}(\mathcal{X}, \mathcal{E})$ , which consists of  $n$  nodes, using *EXACT counters*, we can prove that the total communication cost required to continuously maintain the MLE of the  $\text{ou } \mathcal{G}$  is equal to  $O(m \cdot n)$ , where  $m$  is the number of *observations*. This corresponds to sending  $m$  messages of size  $n$ . The *communication cost* increases linearly with the number of the *observations* in *data stream*. Therefore, it is necessary to find a way to avoid this linear dependence on the number of *observations*.

---

**Algorithm 2:** Exact Counter (*EXACT*) - Coordinator

---

**Input:**

*type\_message*: corresponds to one of the available types of messages  
*message*: corresponds to a message

1. **If** (*type\_message* = "INCREMENT")
  2.     *GET\_COUNTER\_MSG* (*message*)
  3.     *UPDATE\_COUNTER*
- 

#### 4.1.1 Approximated Distributed counters

To mitigate the linear dependency on the number of *training instances*, it is necessary to maintain an  $(\epsilon, \delta)$ -approximation of the MLE of the Bayesian Network  $\mathcal{G}(\mathcal{X}, \mathcal{E})$ . To accomplish this, appropriate *approximate distributed counters* are employed, which are accompanied by the minimal *communication cost*. The implemented types of counters are the *RANDOMIZED* and the *DETERMINISTIC* counters, which are discussed in detail. *Table 2* provides the space and communication complexity of each counter, while *Table 1* presents the notations used for both on *RANDOMIZED* and *DETERMINISTIC counters*.

#### **Count-Tracking problem**

Each *site*  $S_i$ , where  $i \in \{1, \dots, k\}$ , maintains a *local counter*  $n_i$ , initially set to zero. As data is received, each *site*  $S_i$  increments its *local counter*  $n_i(t)$ , where  $n_i(t)$  denotes the value of the  $n_i$  at the time  $t$ . Additionally, assuming each *update* for each site follows the format  $\langle i, c \rangle$ , each *site*  $S_i$  updates the *local counter*  $n_i(t)$  by  $c$  (*Cash register model*), where  $c > 0$  and specifically  $c = 1$ , allowing only *insertions*. As a result, each *local counter*  $n_i$  represents a *linear, monotonic increasing function* and accordingly the total counter.

Our objective is for the Coordinator to maintain an  $(\epsilon, \delta)$ -approximation of the *total counter*  $n(t) = \sum_{i=1}^k n_i(t)$  at any given time, where  $0 \leq \epsilon, \delta \leq 1$ . This can be defined as follows:

$$(1 - \epsilon) \cdot n(t) \leq \hat{n}(t) \leq (1 + \epsilon) \cdot n(t) \text{ with probability } 1 - \delta$$

While also ensuring the minimal *communication cost*, it has been mentioned in [19], the minimum required *communication cost* to maintain such a counter is equal to  $O(k/\epsilon \cdot \log N)$ , where  $N$  denotes the final value of the total counter  $n$  and  $k$  denotes the number of *sites*.

<i>Notations</i>	<i>Description</i>
$n_i$	<i>Local counter of <math>S_i</math></i>
$n$	<i>Total counter</i>
$\bar{n}_i$	<i>Last sent value of <math>S_i</math> to Coordinator</i>
$p$	<i>Probability <math>p \in [0,1)</math>. Initially, <math>p = 1</math></i>
$\hat{n}_i$	<i>The estimator of <math>n_i</math></i>
$\hat{n}$	<i>The estimation over the sites (sum of <math>\hat{n}_i</math>)</i>
$\bar{n}$	<i>The last broadcast value from the Coordinator</i>
$\lfloor x \rfloor_2$	<i>The next power of 2 less than <math>x</math></i>
$n'_i$	<i>The last update of the <math>n_i</math> on Coordinator</i>
$n'$	<i>The sum of <math>n'_i</math></i>
$N$	<i>The final value of the counter</i>
$\varepsilon$	<i>Approximation factor</i>
$\delta$	<i>Delta refers to randomized counters</i>
$k$	<i>The number of sites</i>

Table 1: Notation summary

#### 4.1.1.1 Randomized Counters

The *RANDOMIZED counter* is the first implemented and discussed in detail below. It has been proven to have the *lowest communication cost* in the *count-tracking problem*, as stated in [19]. The communication cost is  $O(\sqrt{k}/\varepsilon \cdot \log N)$ , and as the name suggests, it utilizes a *randomized algorithm*. By employing *randomization*, the communication cost is reduced by  $\sqrt{k}$  compared to any *deterministic method*.

By utilizing the *randomization method*, we can continuously maintain an *unbiased estimator* for  $n_i$ , such that  $E[\hat{n}_i] = n_i$ , while the *variance* of the *estimator* is  $Var[\hat{n}_i] = (\varepsilon n)^2/k$ .

Given the *variance* of each *estimator* is  $(\varepsilon n)^2/k$ , we can prove the total variance is  $(\varepsilon n)^2$ .

By applying the property of the *independence* of  $n_i$ 's, the following holds:

$$Var[\hat{n}] = Var\left[\sum_{i=1}^k \hat{n}_i\right] = \sum_{i=1}^k Var[\hat{n}_i] = k \cdot Var[\hat{n}_i] = (\varepsilon n)^2$$

The proceeding statement is sufficient to produce an *estimator*  $\hat{n}$  with an error  $\varepsilon n$  and a probability  $1 - \delta$ , by utilizing *Chebyshev's inequality*.

Therefore, given the *variance* of each *unbiased estimator*  $\hat{n}_i$  is  $(\varepsilon n)^2/k$ , it is sufficient to support an *error* equal to  $\varepsilon n/\sqrt{k}$ , compared to  $\varepsilon n/k$  which is required for any *deterministic method*. As a result, we achieve an improvement of  $\sqrt{k}$ . This is demonstrated by utilizing *Chebyshev's inequality*, where the following statement must hold:

$$\frac{Var[\hat{n}_i]}{(error)^2} \leq 1 \Rightarrow error \geq \varepsilon n/\sqrt{k}$$

Each site  $S_i$  sends the last value of the local counter  $n_i$  to the Coordinator with a probability  $p$  when receives an event that increments the *local counter*  $n_i$ . Regarding the estimation of the *local counter*  $n_i$  on Coordinator, the following applies:

$$\hat{n}_i = \begin{cases} \bar{n}_i - 1 - 1/p, & \text{if } \bar{n}_i \text{ exists} \\ 0, & \text{otherwise} \end{cases}$$

Where  $\bar{n}_i$  represents the *last update* on Coordinator for the *local counter*  $n_i$ , while the total estimation of the total count  $n$  is the sum of the corresponding  $\hat{n}_i$  i.e.,  $\hat{n} = \sum_{i=1}^k \hat{n}_i$ .

We can prove that the  $\hat{n}_i$  is an *unbiased estimator* with variance  $Var[\hat{n}_i] \leq 1/p^2$ , because the *total estimation*  $\hat{n}$  is an *unbiased estimator* with variance  $Var[\hat{n}] \leq k/p^2$ . Setting the probability appropriately  $p = \theta(\sqrt{k}/\epsilon n)$ , is sufficient, then the variance of the total estimator is  $Var[\hat{n}] = (\epsilon n)^2$ , which is sufficient to maintain an *estimator*  $\hat{n}$  with *error*  $\epsilon n$  with probability  $1 - \delta$  at any time. The proof is discussed in detail in [19].

Considering that the  $n$  is not accessible from all sites at any time and  $p = \theta(\sqrt{k}/\epsilon n)$ , it is sufficient for each site  $S_i$  has a *constant-factor approximation* of the  $n$  in each *round*. To do so, not only each site  $S_i$  sends an *increment* message to the Coordination with probability  $p$  (INCREMENT), but also each site  $S_i$  sends an additional message whenever the *local counter*  $n_i$  doubles (DOUBLES). When the Coordinator receives the DOUBLES message, it then updates the variable  $n' = \sum_{i=1}^k n'_i$ , where  $n'_i$  denotes the *last update* of the *local counter*  $n_i$  on Coordinator. When  $n'$  doubles (more precisely, when changes by a factor between 2 and 4), the Coordinator broadcasts the  $n'$  at all sites and then a new round begins. This way ensures that each site always has a *constant-factor approximation* of  $n$  in each *round*, which is used from sites to define the probability  $p$ . We assume  $\bar{n}$  represents the last broadcast value of  $n'$ .

While  $\bar{n} \leq \sqrt{k}/\epsilon$  then we set  $p = 1$ . As a result, the first  $O(\sqrt{k}/\epsilon)$  elements will be sent to the Coordinator due to the probability  $p = 1$ . When  $\bar{n} > \sqrt{k}/\epsilon$ , we set  $p = 1/\lfloor \epsilon \bar{n} / \sqrt{k} \rfloor_2$ . While the  $\bar{n}$  increased in each round, accordingly the probability decreased (more precisely, the probability sub-halved in each round).

At the beginning of each round, the probability decreased, so we need to adapt the values of  $\bar{n}_i$  on the Coordinator to ensure that the whole system runs with the new probability  $p$ . This can be achieved in the following manner: each site  $S_i$  decides with probability  $p = 1/2$ , if the  $\bar{n}_i$  remains the same. If decide to change it, then each site  $S_i$  flips a *coin* with probability  $1/p$  (with the new  $p$ ) repeatedly. For each failed *coin flip* decreases, the  $\bar{n}_i$  by 1 until a *successful coin flip* or  $\bar{n}_i = 0$ . In the end, each site  $S_i$  sends a message to the Coordinator with the new value of  $\bar{n}_i$ . The last step is necessary because of ensuring to prevent the *bias* will be generated on the estimation *estimator*  $\hat{n}_i$  with the change of  $p$ . The whole process is summarized in Algorithms 3,4.

---

**Algorithm 3: Randomized Counter (RC) - Worker**

---

**Input:**  $\epsilon$ ,  $\delta$  $type\_message$ : corresponds to one of the available types of messages

1. **If** ( $n_i$  increment by one)
  2.      $SEND\_MESSAGE(n_i, p, 'INCREMENT')$
  3.      $UPDATE(\bar{n}_i)$
  4. **If** ( $n_i$  doubles)
  5.      $SEND\_MESSAGE(n'_i, p=1, 'DOUBLES')$
  6.      $UPDATE(\bar{n}_i)$
  7. **If** ( $type\_message = 'UPDATE'$ )
  8.     // New round begins
  9.      $UPDATE(\bar{n})$
  10.    **If** ( $\bar{n} < \sqrt{k}/\epsilon$ ) Set  $p=1$
  11.    **Else**
  12.     Set  $p = 1/\lceil \epsilon \bar{n} / \sqrt{k} \rceil_2$
  13.     **If** ( $p$  halved)
  14.     // Adjust the  $n_i$
  15.      $SEND\_MESSAGE(n_i, p = \frac{1}{2}, 'INCREMENT')$
  16.     **If** (loose flip)
  17.     While (Flip until succeed)
  18.      $SEND\_MESSAGE(n_i - num\_of\_failures, p = 1, 'INCREMENT')$
  19.      $UPDATE(\bar{n}_i)$
- 

Regarding the communication cost of the *RANDOMIZED counter*, the cost can be analyzed as follows. In each round, the communication cost consists of the communication cost of the broadcast of the value  $\bar{n}$  from the Coordinator (*upstream communication cost*), resulting in a cost of  $O(k)$ . Additionally, the *communication cost* required to send the sites to the Coordinator (*downstream communication cost*), equals  $O(np)$ . Combining these costs, the communication cost per round is  $O(k + np) = O(k + \sqrt{k}/\epsilon) = O(\sqrt{k}/\epsilon)$ . Considering that there are  $\log N$  rounds, the total communication cost is  $O(\sqrt{k}/\epsilon \cdot \log N)$ .

---

**Algorithm 4: Randomized Counter (RC) - Coordinator**

---

**Input:**  $type\_message$  corresponds to one of the available types of messages

1. **If** ( $type\_message = 'INCREMENT'$ )
  2.      $UPDATE(\bar{n}_i)$
  3.      $CALCULATE(\hat{n}_i)$
  4. **If** ( $type\_message = 'DOUBLES'$ )
  5.      $UPDATE(n'_i)$
  6.      $UPDATE(n')$
  7.     **If** ( $n'$  doubles) // Change by a factor between 2 and 4
  8.     // New round begins
  9.     Set  $\bar{n} = n'$
  10.     $BROADCAST(\bar{n}, 'UPDATE')$
-

#### 4.1.1.2 Deterministic Counters

The second counter implemented and discussed in detail below is the *DETERMINISTIC counter*. Similar approaches have been used in [17] and [20]. As the name suggests, we focus on *deterministic* methods while ensuring an  $\varepsilon$ -*approximation* of the *counter*. At any time  $t$  the following statement holds:

$$(1 - \varepsilon) \cdot n(t) \leq \hat{n}(t) \leq (1 + \varepsilon) \cdot n(t)$$

The algorithm operates as follows. Given an  $\varepsilon$ -*approximation* of the *counter*, the *maximum error* that can be supported by Coordinator at any time is  $\varepsilon n$ . Considering the independent of the  $k$  sites, we can easily prove that the *maximum error* at each site, while ensuring an  $\varepsilon$ -*approximation*, equals  $\varepsilon n/k$ .

The whole process is summarized in *Algorithms 4,5*. Since each site cannot know the exact value of  $n$  at any time, although it is sufficient for each site to maintain a constant *factor approximation* of the  $n$  through  $\bar{n}$  at each *round*, where  $\bar{n}$  denotes the last broadcast value of the Coordinator. Initially, we set  $\bar{n} = 0$ . Now, each *site*  $S_i$  sends a message to the Coordinator whenever the  $n_i > \varepsilon \bar{n}/k$  (*INCREMENT*). When the Coordinator receives  $k$  such messages, indicating that the maximum error has been reached, it broadcasts a message to all sites to receive the *local counters* from each site (*DRIFT*). Upon receiving a *DRIFT* message, the Coordinator updates the value of  $\bar{n}$ . When  $k$  messages have been received (one from each site), the Coordinator broadcasts an *UPDATE* message to all *sites* with the new value of the  $\bar{n}$ , marking the beginning of a new round. Finally, it can be observed that the first  $O(k/\varepsilon)$  elements will be sent to the Coordinator, as  $\varepsilon \bar{n}/k \leq 1$ .

---

#### **Algorithm 4:** Deterministic Counter (DC) - Worker

---

**Input:** *epsilon*

*type\_message* corresponds to one of the available types of messages

1. **If** (*type\_message* = `INPUT`)
  2.     *Increment*  $n_i$  // *local drift*
  3. **If** ( $n_i > \varepsilon \bar{n}/k$ )
  4.     *SEND\_MESSAGE* ( $n_i, p = 1, \text{`INCREMENT`}$ )
  5.     *Reset the local counter*  $n_i$  // *local drift*
  - 6.
  7. **If** (*type\_message* = `DRIFT`)
  8.     *SEND\_MESSAGE* ( $n_i, p = 1, \text{`DRIFT`}$ )
  9.     *Reset the local counter*  $n_i$  // *local drift*
  - 10.
  11. **If** (*type\_message* = `UPDATE`)
  12.     // *New round begins*
  13.     *UPDATE* ( $\bar{n}$ )
- 

The communication cost of the *DETERMINISTIC counter* can be analyzed as follows. In each round, the communication cost consists of the communication cost of the broadcast of both the value  $\bar{n}$  and a *DRIFT* message from the Coordinator (*upstream communication cost*), resulting in a cost of  $O(k)$ . Additionally, the communication cost required from the sites to the Coordinator (*downstream communication cost*) resulting in a cost of  $O(k + k/\varepsilon)$ . Combining these costs, the communication cost per round is  $O(k + k/\varepsilon) = O(k/\varepsilon)$ . The



Coordinator broadcasts the updated value of  $\bar{n}$  each time it increases by a factor of  $1 + \sum_{i=1}^k \varepsilon/k = 1 + \varepsilon$ . This implies that the number of rounds is bounded by  $O(\log_{1+\varepsilon} N)$ . Therefore, the total communication cost is  $O(k/\varepsilon \cdot \log N)$ .

---

**Algorithm 5:** *Deterministic Counter (DC) – Coordinator*

---

**Input:** *type\_message corresponds to one of the available types of messages*

1. **If** (*type\_message* = `INCREMENT`)
  2.     UPDATE ( $\bar{n}_i$ )
  3.     CALCULATE ( $\hat{n}_i$ )
  4.     **If** (*num\_messages\_inc* =  $k$ )
  5.         BROADCAST(`DRIFT`)
  - 6.
  7.     **If** (*type\_message* = `DRIFT`)
  8.         UPDATE ( $\bar{n}$ )
  9.         **If** (*num\_messages\_drift* =  $k$ )
  10.         // New round begins
  11.         BROADCAST( $\bar{n}$ , `UPDATE`)
- 

<i>Count-Tracking problem</i>	<i>Space (per site)</i>	<i>Communication cost</i>
<i>EXACT</i>	$O(1)$	$O(N)$
<i>RANDOMIZED</i>	$O(\log N)$	$O(\sqrt{k}/\varepsilon \cdot \log N)$
<i>DETERMINISTIC</i>	$O(\log N)$	$O(k/\varepsilon \cdot \log N)$

Table 2: Space and communication complexity for each counter

#### 4.1.1.3 Comparison between RANDOMIZED and DETERMINISTIC

The only difference between the counters is the dependence on the number of sites  $k$ , specifically, the *RANDOMIZED counter* has an improvement by a factor of  $\sqrt{k}$ . The latter was confirmed through experimental evaluation, where the change of the communication cost of the *RANDOMIZED counter* was smaller compared to the *DETERMINISTIC counter* when varying the number of sites  $k$ .

The reasons for implementing the *DETERMINISTIC counter* are twofold. Firstly, it relates to the completeness of the problem. We want to address the problem by considering how each *approximate counter* can be adapted. Secondly, we observe that when the number of sites is small, the *DETERMINISTIC counter* exhibits a *lower communication cost*.

In the *DETERMINISTIC counter*, the quantity  $\varepsilon \cdot \bar{n}_{DC}/k$  defines when a message is sent, whereas, for the *RANDOMIZED counter*, it is the probability  $p = 1/\lceil \varepsilon \cdot \bar{n}_{RC}/\sqrt{k} \rceil_2$ . The values of  $\bar{n}_{DC}$  and  $\bar{n}_{RC}$  determine the number of messages that will be sent, considering the values of the  $\varepsilon, k$  constant.

In most cases, we observe that  $\bar{n}_{RC} < \bar{n}_{DC}$ , indicating that the number of messages required for the *DETERMINISTIC counter* to reach the value  $\bar{n}_{DC}$  is bigger. Assuming the *DETERMINISTIC counter* sends a message with a probability  $p'$ , which is determined by the quantity  $\varepsilon \cdot \bar{n}_{DC}/k$ . Despite  $\bar{n}_{RC} < \bar{n}_{DC}$  resulting in a larger number of messages to achieve the probability  $p'$ , henceforth the  $p' > p$ . Although the *DETERMINISTIC counter* sends more messages to reach the value of  $\bar{n}_{DC}$ , it gains an advantage due to  $p' > p$ .

When the difference between  $\bar{n}_{RC}$  and  $\bar{n}_{DC}$  remains small, which is primarily defined by the number of *sites*  $k$ , the advantage gained from the *DETERMINISTIC counter* is larger. With a smaller number of *sites*, the difference between them becomes less significant, and the advantage of the *DETERMINISTIC counter* is amplified. However, when the difference between  $\bar{n}_{RC}$  and  $\bar{n}_{DC}$  is substantial, the advantage of the *DETERMINISTIC counter* diminishes. This is because the number of messages required to reach the value of  $\bar{n}_{DC}$  is significantly larger than the number of messages to reach the value  $\bar{n}_{RC}$ , reducing the impact of the probability  $p'$ . This occurs when  $\bar{n}_{DC} \gg \bar{n}_{RC}$ .

Based on the experimental evaluation, it is preferable to use the *DETERMINISTIC counter* when the number of *sites* is small, while the *RANDOMIZED counter* is more suitable when the number of *sites* is large.

As shown in *Figure 12*, we measure the values of  $\bar{n}_{RC}$  and  $\bar{n}_{DC}$  ( $y$  – axis) while varying the number of *sites*  $k$ , assuming  $\varepsilon = 4 \times 10^{-4}$ ,  $\delta = 0.25$  and  $k \in [2,30]$ . As the number of *sites* increases, the difference in values  $\bar{n}_{RC}$  and  $\bar{n}_{DC}$  also becomes significantly larger.

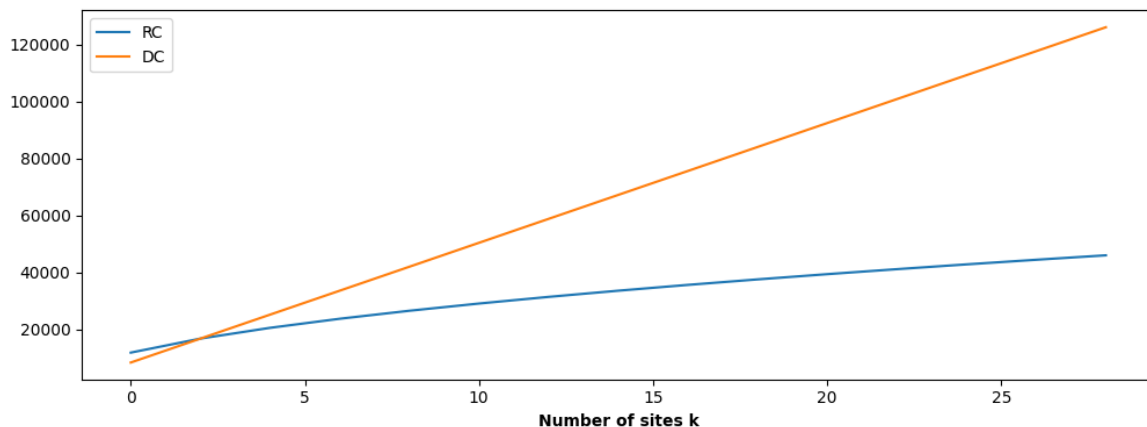


Figure 12:  $\bar{n}_{RC}$  and  $\bar{n}_{DC}$  varying the number of sites  $k$

#### 4.1.2 Analysis of BASELINE, UNIFORM and NON\_UNIFORM

Apart from the general approach and the previously discussed *approximated distributed counters*, the remaining analysis focuses on how to allocate the available error budget among the approximate *distributed counters*  $A_i(x_i, x_i^{par})$ ,  $A_i(x_i^{par})$ , while ensuring an  $(\epsilon, \delta)$ -*approximation of the MLE* at any time.

This section introduces and examines the *BASELINE*, *UNIFORM*, and *NON\_UNIFORM* algorithms, which provide a way to divide the available error budget.

The analysis includes the *RANDOMIZED counter*. Table 3 presents the results of each one of the algorithms. In the case of the *DETERMINISTIC counter*, the difference lies in the parameters  $\delta$  and  $k$ . Specifically, the *DETERMINISTIC counter* exhibits a linear dependency on the number of sites  $k$ , while it does not depend on  $\delta$ .

Given a Bayesian Network Structure  $\mathcal{G}(\mathcal{X}, \mathcal{E})$ , where the number of nodes denoted as  $\mathcal{X} = \{X_1, \dots, X_n\}$ , which constitutes a *Bayesian Network* with  $n$  nodes. Additionally,  $J_i$  represents the domain cardinality of the  $X_i$  node, denoted as  $J_i = |\text{Val}(X_i)|$  for each  $i \in \{1, \dots, n\}$ , while the  $K_i$  represents the domain cardinality of *parents* nodes of  $X_i$ , denoted as  $K_i = |\text{Val}(\text{Par}(X_i))|$  for each  $i \in \{1, \dots, n\}$ . Finally, the  $k$  denotes the number of *sites*.

Additionally, we assume  $J = \max_{i=1}^n J_i$  denotes the node with the largest set of values in the network,  $d = \max_{i=1}^n |\text{Par}(X_i)|$  denotes the maximum number of parents node for any variable in the network and the parameters  $\epsilon, \delta$  within the range of  $0 \leq \epsilon, \delta \leq 1$ .

#### **BASELINE**

The *BASELINE* algorithm is the first algorithm introduced. The *approximation factor* for each *approximate distributed counter*  $A_i(x_i, x_i^{par})$ ,  $A_i(x_i^{par})$ , is assigned to  $\frac{\epsilon}{3n}$ . The *BASELINE* algorithm ensures an  $(\epsilon, \delta)$ -*approximation of MLE* of the *joint probability distribution* of the network  $\mathcal{G}$  at any time. The proof is discussed in detail in [18, Section IV-C]. The error budget is divided uniformly among the counters.

The total communication cost given  $m$  *training instances* is:

$$O\left(\frac{n^2 \cdot J^{d+1} \sqrt{k}}{\epsilon} \cdot \log \frac{1}{\delta} \cdot \log m\right)$$

In this case, by replacing the value of the approximation factor  $\epsilon$ , the *communication cost* of each *approximate distributed counter* is  $O\left(\frac{n \cdot \sqrt{k}}{\epsilon} \cdot \log \frac{1}{\delta} \cdot \log m\right)$  messages. Additionally, for each  $i \in \{1, \dots, n\}$ , there are at most  $J^{d+1}$  *counters*  $A_i(x_i, x_i^{par})$  and  $J^d$  *counters*  $A_i(x_i^{par})$ , for each  $x_i \in \text{Val}(X_i)$  and  $x_i^{par} \in \text{Val}(\text{Par}(X_i))$ . Combing them, the total communication cost is  $O\left(\frac{n^2 \cdot J^{d+1} \sqrt{k}}{\epsilon} \cdot \log \frac{1}{\delta} \cdot \log m\right)$  messages. By using the *BASELINE* algorithm, we manage to avoid the linear dependency on the number of *training instances*  $m$ , as the dependency is logarithmic.

## UNIFORM

A different approach to the problem leads to the *UNIFORM* algorithm. The *approximation factor* for each *approximate distributed counter*  $A_i(x_i, x_i^{par})$ ,  $A_i(x_i^{par})$ , is  $\frac{\varepsilon}{16\sqrt{n}}$ . By employing the *UNIFORM* algorithm, we again ensure an  $(\varepsilon, \delta)$ -*approximation* of the *MLE* of the *joint probability distribution* of the network  $\mathcal{G}$ . The proof is described in detail in [18, Section IV-C]. The error budget is uniformly divided among the counters.

The total communication cost given  $m$  *training instances* is:

$$O\left(\frac{n^{3/2} \cdot J^{d+1} \sqrt{k}}{\varepsilon} \cdot \log \frac{1}{\delta} \cdot \log m\right)$$

The *communication cost* for each *approximate distributed counter* is  $O\left(\frac{\sqrt{nk}}{\varepsilon} \cdot \log \frac{1}{\delta} \cdot \log m\right)$  messages, given the *approximation factor*, is  $\frac{\varepsilon}{16\sqrt{n}}$ . For each  $i \in \{1, \dots, n\}$ , there are at most  $J^{d+1}$  counters  $A_i(x_i, x_i^{par})$  and  $J^d$  counters  $A_i(x_i^{par})$ , for each  $x_i \in Val(X_i)$  and  $x_i^{par} \in Val(Par(X_i))$ . Combing them, the total communication cost is  $O\left(\frac{n^{3/2} \cdot J^{d+1} \sqrt{k}}{\varepsilon} \cdot \log \frac{1}{\delta} \cdot \log m\right)$  messages. The dependency on the number of *training instances* is logarithmic and we achieve an improvement of  $\sqrt{n}$  on the communication cost compared to the *BASELINE* algorithm.

## NON\_UNIFORM

The two previous algorithms uniformly divide the available among the *counters*. None of them consider elements related to the structure of a node, such as the *domain cardinality* and the number of parents nodes. Now, the difference is the *NON\_UNIFORM* includes such elements. The available *error budget* is no longer uniformly divided among the counters, but the *approximation factor* for each *approximate distributed counter*  $A_i(x_i, x_i^{par})$ ,  $A_i(x_i^{par})$  is related to both  $J_i = |Val(X_i)|$  and  $K_i = |Val(Par(X_i))|$ .

In this case, there is a distinction between the counters. For each *approximate distributed counter*  $A_i(x_i, x_i^{par})$ , the *approximation factor* is defined in the following manner:

$$v_i = \frac{(J_i K_i)^{\frac{1}{3}} \cdot \varepsilon}{16\alpha}, \text{ where } \alpha = \left(\sum_{i=1}^n (J_i K_i)^{\frac{2}{3}}\right)^{1/2}$$

While for each *approximate distributed counter*  $A_i(x_i^{par})$ , the *approximation factor* is defined in the following manner:

$$\mu_i = \frac{(K_i)^{\frac{1}{3}} \cdot \varepsilon}{16\beta}, \text{ where } \beta = \left(\sum_{i=1}^n (K_i)^{\frac{2}{3}}\right)^{\frac{1}{2}}$$

By applying the values of  $v_i, \mu_i$ , we ensure an  $(\varepsilon, \delta)$ -*approximation* of the *MLE* of the *joint probability distribution* of the network  $\mathcal{G}$ . The proof is discussed in detail in [18, Section IV-C].

In this case, the *approximation factor* of each *approximate distributed counter*  $A_i(x_i, x_i^{par})$  depends on both  $J_i$  and  $K_i$ . This means that the error is proportional to both the set of values of  $X_i$  and the domain cardinality of parents nodes of  $X_i$ . On the other hand, the *approximation*

factor of each *approximate distributed counter*  $A_i(x_i^{par})$  solely depends on  $K_i$ , meaning that the error is only proportional to the domain cardinality of parents nodes of  $X_i$ .

The total communication cost given  $m$  training instances is:

$$O\left(\Gamma \cdot \frac{\sqrt{k}}{\varepsilon} \cdot \log \frac{1}{\delta} \cdot \log m\right), \text{ όπου } \Gamma = \left(\sum_{i=1}^n (J_i K_i)^{\frac{2}{3}}\right)^{3/2} + \left(\sum_{i=1}^n (K_i)^{\frac{2}{3}}\right)^{3/2}$$

The *communication cost* of each *approximate distributed counter*  $A_i(x_i, x_i^{par})$  is  $O\left(\frac{\sqrt{k}}{v_i} \cdot \log \frac{1}{\delta} \cdot \log m\right)$  messages. For each node  $X_i$ , the counters  $A_i(x_i, x_i^{par})$  are defined by both  $J_i$  and  $K_i$ , so there are at most  $J_i \cdot K_i$  counters. Combining them, the total communication cost for all counters  $A_i(x_i, x_i^{par})$  for each node  $X_i$  is:

$$M_1 = \sum_{i=1}^n \frac{J_i \cdot K_i \cdot \sqrt{k}}{v_i} \cdot \log \frac{1}{\delta} \cdot \log m$$

Replacing the  $v_i$ , we can redefine the  $M_1$  with the following manner:

$$M_1 = \left(\sum_{i=1}^n (J_i \cdot K_i)^{2/3}\right)^{3/2} \cdot \frac{\sqrt{k}}{\varepsilon} \cdot \log \frac{1}{\delta} \cdot \log m$$

Accordingly, for each node  $X_i$  there are at most  $K_i$  counters  $A_i(x_i^{par})$ , for which the communication cost is  $O\left(\frac{\sqrt{k}}{\mu_i} \cdot \log \frac{1}{\delta} \cdot \log m\right)$  messages. Combining them, the total communication cost for all the nodes is (replacing also the  $\mu_i$ ):

$$M_2 = \left(\sum_{i=1}^n K_i^{2/3}\right)^{3/2} \cdot \frac{\sqrt{k}}{\varepsilon} \cdot \log \frac{1}{\delta} \cdot \log m$$

The total communication cost is  $M_1 + M_2 = O\left(\Gamma \cdot \frac{\sqrt{k}}{\varepsilon} \cdot \log \frac{1}{\delta} \cdot \log m\right)$  messages. The linear dependency on the number of training instances also remains.

	<i>Approximation Factor</i>	<i>Communication Cost</i>
<i>BASELINE</i> $A_i(x_i, x_i^{par}), A_i(x_i^{par})$	$\frac{\varepsilon}{3n}$	$O\left(\frac{n^2 \cdot J^{d+1} \sqrt{k}}{\varepsilon} \cdot \log \frac{1}{\delta} \cdot \log m\right)$
<i>UNIFORM</i> $A_i(x_i, x_i^{par}), A_i(x_i^{par})$	$\frac{\varepsilon}{16\sqrt{n}}$	$O\left(\frac{n^{3/2} \cdot J^{d+1} \sqrt{k}}{\varepsilon} \cdot \log \frac{1}{\delta} \cdot \log m\right)$

<i>NON_UNIFORM</i> $A_i(x_i, x_i^{par})$	$v_i = \frac{(J_i K_i)^{\frac{1}{3}} \cdot \varepsilon}{16a}$ $\alpha = \left( \sum_{i=1}^n (J_i K_i)^{\frac{2}{3}} \right)^{1/2}$	$O\left(\Gamma \cdot \frac{\sqrt{k}}{\varepsilon} \cdot \log \frac{1}{\delta} \cdot \log m\right)$ $\Gamma = \left( \sum_{i=1}^n (J_i K_i)^{\frac{2}{3}} \right)^{3/2} + \left( \sum_{i=1}^n (K_i)^{\frac{2}{3}} \right)^{3/2}$
$A_i(x_i^{par})$	$\mu_i = \frac{(K_i)^{\frac{1}{3}} \cdot \varepsilon}{16\beta}$ $\beta = \left( \sum_{i=1}^n (K_i)^{\frac{2}{3}} \right)^{1/2}$	

Table 3: Approximation factor and Communication Cost for algorithms

#### 4.1.2.1 Dummy Father

During the evaluation of the *NON\_UNIFORM* algorithm, where the *approximation factor*  $\varepsilon$  is proportional to both  $J_i$  and  $K_i$ , we observed a problematic situation related to the *orphan nodes*, that is  $Par(X_i) = \emptyset$ . By applying the *NON\_UNIFORM* algorithm, the *approximation factor*  $\varepsilon$  for each node is zero due to the  $K_i$ 's node is also zero. As a result, we need to maintain the *EXACT counter* for each  $x_i \in Val(X_i)$  of a such node. Consequently, when the number of *orphan nodes* is significant incurring quite high *communication cost* for maintaining them.

To prevent this problem, we suggest for each *orphan node*  $X_i$ , the insertion of a *dummy parent node* (*Dummy father* -- Figure 13). Specifically, for each such node, we set  $K_i = 1$ . With the insertion of the node managing to avoid the maintenance of the *EXACT counter* for each counter  $A_i(x_i, x_i^{par})$  relating to the *orphan node*  $X_i$ .

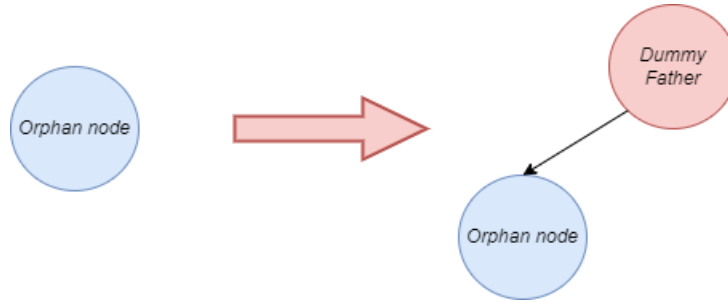


Figure 13: Dummy Father

Moreover, we achieve to increase in the value of  $\alpha = \left( \sum_{i=1}^n (J_i K_i)^{\frac{2}{3}} \right)^{1/2}$ , which leads to a slightly larger *approximation factor* for the rest of the counters. This also leads to slightly *lower communication cost*. The primary advantage is the avoidance of maintaining the *EXACT counters*, which mainly contributes to the reduction of the communication cost achieved.

The *NON\_UNIFORM* algorithm is combined with the *Dummy Father* method in experimental evaluation. We achieve to reduce the *communication cost* by a percentage is ranged from 0 – 50% related to the *NON\_UNIFORM* (See. Chapter 6.3.5). To mention that the reduction percentage is achieved, mainly depends on the topology of the *Bayesian Network*. The reduction percentage varied among the different *Bayesian Networks* because the reduction depends on whether the network contains orphan nodes or not.

#### 4.1.2.2 Comparison among BASELINE, UNIFORM, NON\_UNIFORM algorithms

Generally, comparing the previous algorithms, the following statement holds  $BASELINE < UNIFORM < NON\_UNIFORM$ .

The difference between *BASELINE* and *UNIFORM* is directly reflected in the communication cost. In most cases, *BASELINE* is better than *UNIFORM* as  $\frac{\epsilon}{16\sqrt{n}} > \frac{\epsilon}{3n}$ , so the *communication cost* for each *approximate distributed counter* of the *UNIFORM* algorithm is lower. However, when the number of nodes  $n \leq 29$ , the *BASELINE* appears better than the *UNIFORM* as  $\frac{\epsilon}{16\sqrt{n}} < \frac{\epsilon}{3n}$  and so the *communication cost* for each *approximate distributed counter* of the *BASELINE* algorithm is lower. Both *BASELINE* and *UNIFORM* have the same dependency on the  $k, \epsilon, \delta$  and  $m$  parameters, although the only difference is on the  $n$  parameter, specifically the *UNIFORM* algorithm is better by a factor of  $\sqrt{n}$ .

Regarding the *UNIFORM* and *NON\_UNIFORM* algorithms, the *NON\_UNIFORM* is at least as good as the *UNIFORM* algorithm, given the *NON\_UNIFORM* algorithm exploits more information about the network. Both *UNIFORM* and *NON\_UNIFORM* have the same dependency on  $k, \epsilon, \delta$  and  $m$  parameters, the point which differs is on the parameters  $J_i, K_i$ , respectively. When the differences of  $J_i, K_i$  among nodes are significant, then the *NON\_UNIFORM* prevails. However, when the values of  $J_i$  and  $K_i$  are quite close, the *UNIFORM* algorithm appears slightly better, which is confirmed on the experimental evaluation because the *Bayesian Networks* are used, and do not appear significant differences in parameters  $J_i, K_i$  among the available nodes.

#### 4.1.3 Communication cost of Naïve Bayes Classifier

This section presents the *communication cost* for each of the three algorithms for the *Naïve Bayes Classifiers*.

Given a *Naïve Bayes Classifier*  $\mathcal{G}(\mathcal{X}, \mathcal{E})$ , where the set of nodes is denoted as  $\mathcal{X} = \{X_1, \dots, X_n, C\}$ . In this case,  $K_i$  for each node  $X_i$  represents the domain cardinality of the *class variable*  $C$ , where  $K_i = J_c$  for each  $X_i \in \mathcal{X} - \{C\}$ . Additionally, the maximum number of parents nodes that exist in the graph is 1, so  $d = 1$ .

Based on *Table 3*, the total communication cost of the *BASELINE* algorithm is:

$$\text{BASELINE with Naïve Bayes Classifier: } O\left(\frac{n^2 \cdot J^2 \sqrt{k}}{\varepsilon} \cdot \log \frac{1}{\delta} \cdot \log m\right)$$

For the *UNIFORM* algorithm, the total communication cost is:

$$\text{UNIFORM with Naïve Bayes Classifier: } O\left(\frac{n^{3/2} \cdot J^2 \sqrt{k}}{\varepsilon} \cdot \log \frac{1}{\delta} \cdot \log m\right)$$

Regarding the *NON\_UNIFORM* algorithm, for each *approximate distributed counter*  $A_i(x_i, x_i^{par})$ , the *approximation factor* is defined as follows:

$$v_i = \frac{(J_i)^{\frac{1}{3}} \cdot \varepsilon}{16\alpha}, \text{ where } \alpha = \left(\sum_{i=1}^n (J_i)^{\frac{2}{3}}\right)^{1/2}$$

While for each *approximate distributed counter*  $A_i(x_i^{par})$ , the *approximation factor* is defined as follows:

$$\mu_i = \frac{(K_i)^{\frac{1}{3}} \cdot \varepsilon}{16\beta} = \frac{\varepsilon}{16\sqrt{n}}, \text{ όπου } \beta = \sqrt{n \cdot (J_c)^{2/3}}$$

The total communication cost required of the *NON\_UNIFORM* algorithm for the *Naïve Bayes Classifier*, given  $m$  training instances, is:

$$\text{NON_UNIFORM with Naïve Bayes Classifier} \\ O\left(J_c \cdot \left(\sum_{i=1}^n (J_i)^{\frac{2}{3}}\right)^{3/2} \cdot \frac{\sqrt{k}}{\varepsilon} \cdot \log \frac{1}{\delta} \cdot \log m\right)$$

In all cases for all algorithms, we achieve logarithm dependency on the number of *training instances*.



## 4.2 A second approach

The second approach proposed for solving the problem while ensuring an  $(\epsilon, \delta)$ -approximation of the MLE of the joint probability distribution of the Bayesian Network Structure  $\mathcal{G}(\mathcal{X}, \mathcal{E})$  over the Distributed Continuous Model. Instead of treating each counter individually, the approach views the counters collectively as vectors and specifically as frequency vectors. This shift allows for a more general and effective approach to be applied to solving the problem, leading to improved results, as discussed in detail below. To achieve this, we incorporate the method of Functional Geometric Monitoring [21] in combination with BASELINE and UNIFORM algorithms. These algorithms are solely used to define the appropriate value of the approximation factor  $\epsilon$  among the counters.

The NON\_UNIFORM algorithm is not used in this approach. As mentioned before, the approximation factor  $\epsilon$  defined by the NON\_UNIFORM is specialized for each approximate counter, while in this case, we utilize frequency vectors. The goal is to have algorithms that can universally be applied to all the counters, and therefore we have selected the BASELINE and UNIFORM algorithms for this purpose. By leveraging the concept of frequency vectors and combining it with the BASELINE and UNIFORM algorithms, we can effectively address the challenges posed by the MLE tracking problem.

The Functional Geometric Monitoring (FGM) method has been chosen due to its significant benefits in terms of communication cost and scalability of the system compared to other methods, particularly its predecessor, Geometric Monitoring (GM) [22]. FGM represents the successor of the GM method, offering improved performance and capabilities

The FGM constitutes a universally applicable method that can be employed for various monitoring problems. It is independent of the monitoring problem, as it employs a problem-specific family of functions, known as safe function which will be discussed in detail later. The method of the FGM can be integrated with ease into any stream processing framework like Apache Flink [23] and Apache Spark [24], providing a general and powerful tool for monitoring a wide range of problems.

Furthermore, the FGM method can be provably adapted under adverse conditions of the monitoring problem, regarding tight monitoring bounds (making it a suitable choice for our purposes). Additionally, the FGM method can be easily adapted in the presence of skew distribution among sites.

### 4.2.1 Functional Geometric Monitoring (FGM)

Given the number of sites  $k$ , where each site  $S_i$  with  $i \in \{1, \dots, k\}$  maintains a frequency vector of counters  $\mathbf{S}_i(t)$  (local state vector), which is updated properly (See Chapter 3.2), while receiving data. In addition, the global state vector  $\mathbf{S}(t)$  refers to the sum of the corresponding local states  $\mathbf{S}_i(t)$ , denoted as  $\mathbf{S}(t) = \sum_{i=1}^k \mathbf{S}_i(t)$ . Our primary concern is the region of the continuous query ( $Q$ ), aiming for the Coordinator to continuously maintains an  $\epsilon$ -approximation of the query value of the global state vector  $\mathbf{S}(t)$ , defined as follows:

$$Q(\mathbf{S}(t)) \in (1 \pm \epsilon)Q(\mathbf{E}(t))$$

With  $\mathbf{E}(t) = \sum_{i=1}^k \mathbf{E}_i(t)$ , where  $\mathbf{E}_i$  denotes the estimated vector on the Coordinator for each site  $S_i$ . To maintain the previous equation, each site  $S_i$  needs to periodically send a message to the Coordinator, particularly at each round, informing how the local state vector  $\mathbf{S}_i(t)$  has

varied from the received *local data stream*. The Coordinator maintains an *estimator vector*  $\mathbf{E}_i$  for each *site*  $S_i$ . Therefore, each time the site sends such a message to the Coordinator, it sends its *drift vector*  $\mathbf{X}_i(t) = \mathbf{S}_i(t) - \mathbf{E}_i$ . The Coordinator updates the  $\mathbf{E}_i$  by adding the  $\mathbf{X}_i$  (vector addition). More details are discussed in [21].

*Functional Geometric Monitoring* (FGM) ensures the *safety* of the entire system, and the value of the *monitoring query* falls within the desired bounds, utilizing a *safe function*  $\varphi$ . The *safe function* denotes a *real function* that is defined by the monitoring problem. Particularly, the safe function employed in each *site*  $S_i$ , where each site  $S_i$  needs to monitor its  $\varphi(\mathbf{X}_i)$  as its *drift vector*  $\mathbf{X}_i$  varies. It is sufficient to prove that if the sum  $\psi = \sum_{i=1}^k \varphi(\mathbf{X}_i) > 0$ , then the *safety* of the entire system is ensured. The *safe zone composition* and quality criteria of *safe zones* are discussed in detail in [25].

The analysis of the MLE algorithm primarily centers around the *monitoring of frequency moments*  $F_p$ . The objective is to effectively maintain the *counters*  $A_i(x_i, x_i^{par})$ ,  $A_i(x_i^{par})$  within a given *Bayesian Network Structure*  $\mathcal{G}(\mathcal{X}, \mathcal{E})$ , where the set of nodes is denoted as  $\mathcal{X} = \{X_1, \dots, X_n\}$ . Therefore, the problem can be framed as the continuous maintenance of the  $F_1$  *frequency moment* of a vector  $\mathbf{x} = [x_1, \dots, x_n]$ . The  $F_1$  *moment*, also known as the  $l^1$  *norm*, is defined as follows:

$$\|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|$$

Regarding our case, each  $x_i$  of the *frequency vector*  $\mathbf{x}$  corresponds to the counters  $A_i(x_i, x_i^{par})$ ,  $A_i(x_i^{par})$ . Therefore, the  $\mathbf{x}$  contains all the counters of each node  $X_i$  in the network  $\mathcal{G}$ . The *frequency* of each counter  $A_i(x_i, x_i^{par})$  and  $A_i(x_i^{par})$  is maintained, representing the number of times appears in the data stream.

Furthermore, when monitoring the  $F_1$  *frequency moment*, we employ *safe functions* that are defined as follows:

$$\|\mathbf{x} + \mathbf{E}\|_1 - T$$

In the context of continuous querying, the threshold for each round will vary and is equivalent to  $(1 + \varepsilon)\|\mathbf{E}\|_1$ . Therefore, the *safe function* can be defined as follows:

$$\|\mathbf{x} + \mathbf{E}\|_1 - (1 + \varepsilon)\|\mathbf{E}\|_1$$

Our objective is to continuously maintain an  $\varepsilon$ -*approximation* of the *MLE* of a *Bayesian Network*  $\mathcal{G}(\mathcal{X}, \mathcal{E})$ . To achieve this, we need to address two prerequisites. The first prerequisite involves dividing the available error budget among the counters in a way that provides an  $\varepsilon$ -*approximation* of the *MLE*. Since the *counters* are framed as *frequency vectors* and the *safe function* follows the previous format, it is evident that the *approximation factor*  $\varepsilon$  must be the same for all *counters*. This immediately implies that the *BASELINE* and *UNIFORM* algorithms are the only ones that can be applied. The second prerequisite is to ensure an  $\varepsilon$ -*approximation* for each counter  $A_i(x_i, x_i^{par})$  and  $A_i(x_i^{par})$  for each  $X_i$ , given that the *error budget* has been appropriately allocated among the counters.

By applying the previous formulation of the *safe function*, it becomes evident that the maximum error at each round is proportional to  $\varepsilon\|\mathbf{E}\|_1$ . However, it is insufficient for ensuring

an  $\varepsilon$ -approximation for each counter  $A_i(x_i, x_i^{par}), A_i(x_i^{par})$ . The reason is that the error is determined by the sum of the values of all the counters, which it not accurately represent the error for each counter. To illustrate this point, we consider a *frequency vector* that contains one small counter and several high counters. The error at each round will be affected by the values of the high counters. As a result, the changes of the small counter may not be detected because its changes will be smaller than the overall error influenced by the high counters. Consequently, the error calculation is based on the sum of all counter values, which does not accurately reflect the error for each specific counter.

We propose a *safe function* in the format  $\|x + E\|_1 - T$ , but the *threshold*  $T$  is defined as follows:

$$T = \|E\|_1 + \text{minViolatedValue}$$

where  $\text{minViolatedValue} = \varepsilon \cdot \min_i E$

The *minViolatedValue* is defined based on the value of the approximation factor  $\varepsilon$  and the minimum value among counters on  $E$ . This approach ensures an  $\varepsilon$ -approximation for each counter  $A_i(x_i, x_i^{par})$  and  $A_i(x_i^{par})$  for each  $X_i$ . By calculating the error at each round using the minimum value among counters, we obtain the least possible error. By ensuring that there is no violation for the minimum counter, we infer that there are no violations for the other counters because they must change by an amount greater than what is defined. Although this approach may result in more rounds, treating the counters collectively and each message on the Coordinator contains information for all the available counters, significantly reduces the communication cost.

Despite the error defined by the minimum counter value, which is significantly small, the FGM method offers the advantage of treating the counters collectively. This means that for each update message from each site  $S_i$ , the Coordinator receives information for all counters. As a result, the communication cost is significantly reduced, as confirmed by the experimental evaluation. We achieve an improvement of 10x in communication cost over the first approach and an improvement of 100x in communication cost compared to EXACTMLE. Moreover, the FGM provides a scalable and high-throughput solution for the problem, as demonstrated in the experimental evaluation.

### **Rebalancing**

Our objective is for the round last longer, which is desirable as it allows for better summarization of the local streams on local state vectors. Our goal is to minimize the upstream communication cost associated with the estimated vector  $E$  at the beginning of each round, whenever possible. While the condition  $\psi = \sum_{i=1}^k \varphi(X_i) > 0$  does not necessarily imply a significant movement of the global state  $S(t)$  relative to  $E$ , the safe function may be quite useful. Instead of broadcasting a new safe function to the sites (incurring upstream communication cost), the Coordinator broadcasts a scaling factor  $\lambda$ . This scaling factor is then used by the sites to appropriately adjust their drift vectors. By doing so, we reduce the communication cost associated with broadcasting an estimated vector. In experimental evaluation, the FGM method is used in conjunction with the rebalancing method.

## Chapter 5:

# Design and Implementation of the system

### 5.1 Apache Flink

The *Apache Flink* [23] is an *open-source* framework used for *distributed stream data processing*. The core of the Flink constitutes the streaming engine, which provides the ability to perform stateful operators over *unbounded data streams* while guaranteeing *in-memory* speed for computations. The Flink is designed to integrate easily into various *cluster environments* such as Hadoop, Yarn, and Kubernetes.

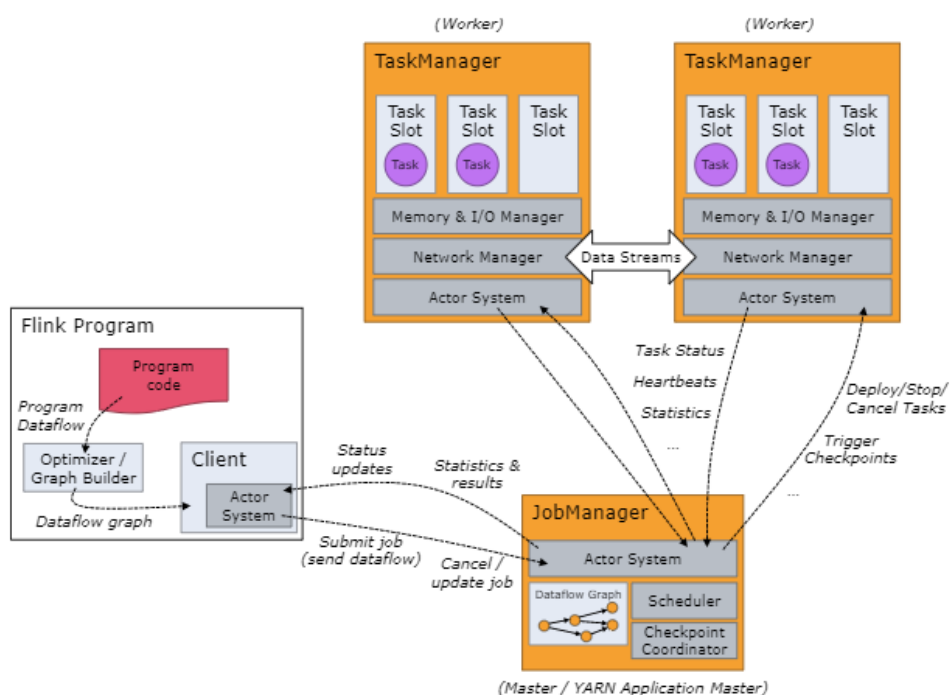


Figure 14: Apache Flink

The Apache Flink consists of two *processes*: the *Job Manager* and the *TaskMangers* (Figure 14). The *Job Manger* is responsible for coordinating the entire execution process of the application, while each *TaskManager* is responsible for the execution *tasks(operators)* of the *dataflow*, as well as buffering and the *exchange of data streams*.

Furthermore, Flink provides a wide range of *APIs* that can be used for developing our application. One of the most significant *APIs* is the one that contains the *stateful operators*. These operators allow us to save *states* while processing the dataset. Each *state* can be framed as a *snapshot* of the operator at a given time, which keeps track of information related to the operator. In our approach, both each site and Coordinator represent a *stateful operator*.

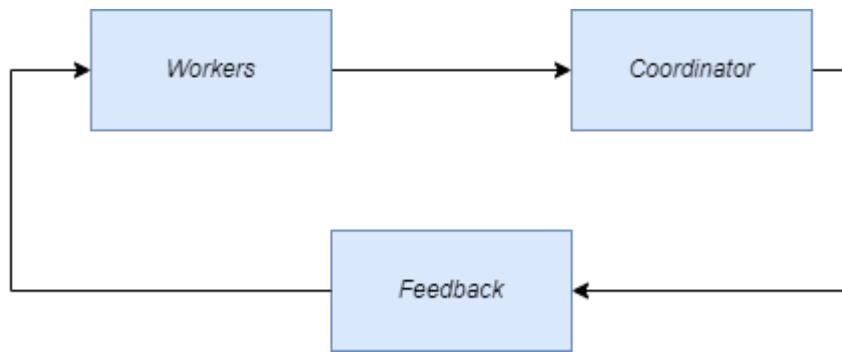


Figure 15: Feedback loop

Between the sites and the Coordinator, there was a *feedback loop*. It can be represented as the *redirecting* of one operator output to the previous one (Figure 15). The feedback loop can be met in cases related to the *continuous maintenance* of someone model, which is equivalent to our problem. The *Apache Flink* provides the *Iterative Stream* operator for implementing the *feedback loop*. Given that the version of the Flink used for executing all the experiments was in the early stage (See Chapter 6), we decided to use a *Kafka topic* as a feedback loop.

## 5.2 Apache Kafka

*Apache Kafka* [26] is an open-source platform that provides the capability of *distributed event streaming*. *Kafka* has emerged as the successor to *traditional messaging systems*, offering essential characteristics such as *high-throughput*, *high-reliability*, *scalability*, and *durability*, which are crucial for processing *distributed streaming data sources*. In *Apache Kafka*, the datasets are stored in *topics*, which are further divided into *partitions* to ensure the *scalability* and *reliability* of the system. Typically, there is more than one *Kafka broker* which constitutes a *Kafka cluster*. *Kafka* offers two primary services: *Kafka consumer* and *Kafka producer*. The *Kafka consumer* is used to read data from a topic while the *Kafka producer* is responsible for publishing data to a topic.

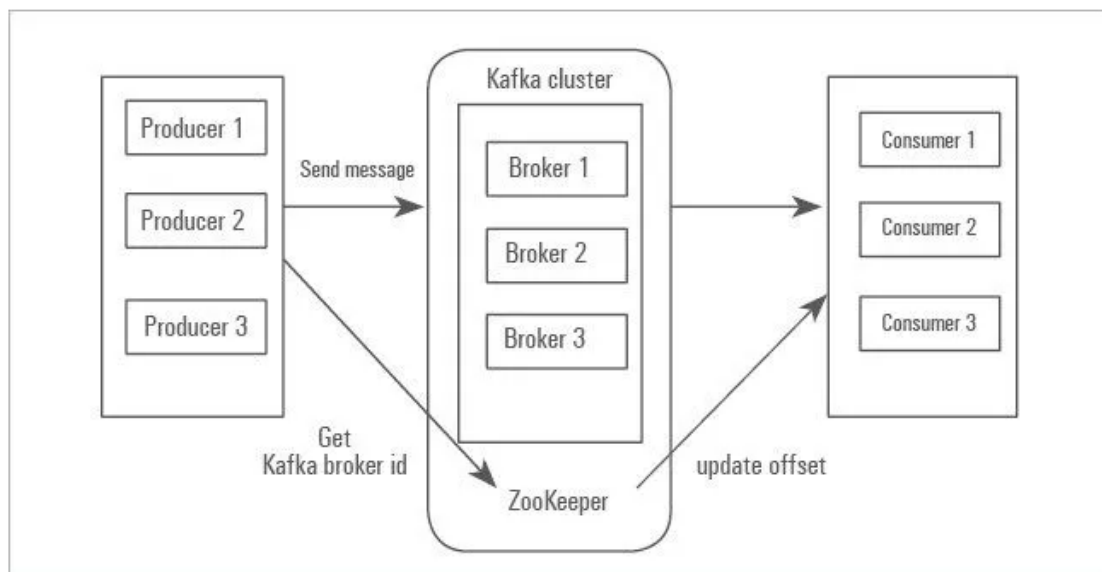


Figure 16: Apache Kafka

In our system, we utilize *Kafka* used for reading *data streams* that are directed to the sites and for implementing the *feedback loop*. For the *feedback loop*, a *Kafka topic* is used as a *buffer* between the producer and the consumer. In this context, the *producer* represents the *Coordinator* which produces all the necessary control messages while the *consumer* represents the *sites*, which need to consume and process the messages sent by *Coordinator*.

### 5.3 System Architecture

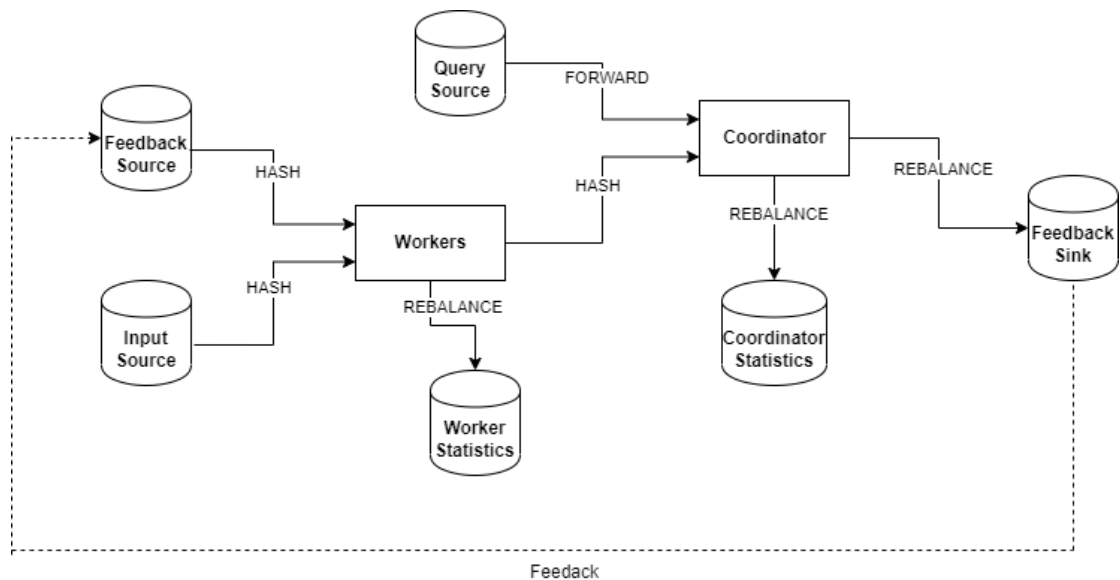


Figure 17: System Architecture

Both *Apache Flink* and *Apache Kafka* are utilized in the implementation of our system. Figure 17 provides an abstract view of our system architecture.

The *Input Source operator* is responsible for reading the incoming dataset from a *Kafka topic* which serves as the *Input Source*. It is responsible for defining the dataset in the appropriate format. Each input is represented as a tuple, following a specific format:

$$\langle value, worker_{id} \rangle$$

The *value* denotes the *input feature vector* of a *Bayesian Network* (or a *Naïve Bayes Classifier*), while the *worker<sub>id</sub>* serves as a *pseudo-key* used for distributing the dataset among *Workers*. It ensures that the dataset is evenly distributed among the workers, following a horizontal partitioning strategy.

Each *Worker* represents a two-input operator, therefore can be framed as a *keyed co-process operator*. The first input refers to the incoming dataset which needs to be processed, while the second input refers to the *Feedback Source*. In this context, the *Feedback Source* contains the control messages that are sent by the *Coordinator*. Additionally, the *Feedback Source* and *Feedback Sink* denote the same *topic (feedback topic)*, which acts as a *buffer* between the *Coordinator* and the *Workers*. Specifically, the *Coordinator* acts as *Kafka Producer* in *Feedback*

*Sink*, while the *Workers* act as *Kafka Consumers* in *Feedback Source*. This achieves a *two-way communication channel* between *Coordinator* and *Worker* (*feedback loop*).

Accordingly, the *Coordinator* is treated as a two-input operator and can be also implemented as a *keyed co-process operator*. The first input corresponds to the messages received from *Workers*, while the second input comes from the *Query Source*. In this context, the *Query Source* can treat either as the *testing source* which includes *queries* used to evaluate the quality of *joint probability distribution* estimation or as a *query source* that contains *user-posed queries* related to the *joint probability distribution* of a *Bayesian Network*.

The exchanged messages between the *Coordinator* and *Worker* are represented as a triplet, following a specific format:

$$\langle value, worker_{id}, type\_message \rangle$$

The *value* field represents the content of the message, which could be a value of an *approximate distributed counter* or a *frequency vector* in the case of the *FGM* method. The *worker<sub>id</sub>* field is used differently depending on the direction of the message flow. For messages from *Workers* to *Coordinator*, it denotes the *cord<sub>id</sub>*, while for messages from the *Coordinator* to *Workers*, it represents the *worker<sub>id</sub>* indicating the intended *Worker* to receive the message. Finally, the *type\_message* field defines the message type based on the specific action performed.

Each message, in addition to the three fields, in case of the *approximate distributed counters* and are treated as individual entities, the message needs to include another field related to the counter that the message is referring to. The following mechanism is also used to retrieval of the counters during the update process. Specifically, each counter in *Bayesian Network* is unique and characterized by the following fields:

$$\langle counterName, counterValue, counterParentsName, counterParentsValue \rangle$$

While in *Naïve Bayes Classifier*, each counter is uniquely characterized by the following fields:

$$\langle index, counterValue, classValue \rangle$$

Where the *index* field denotes the position of the counter value in the *input feature vector*.

In case each message contains all the previous fields, the size of the message will be significantly large. Instead of each counter characterized by a *unique key* corresponded to a long number. This number is generated by hashing all the fields of each counter, using an appropriate hash function (*Figure 18*).

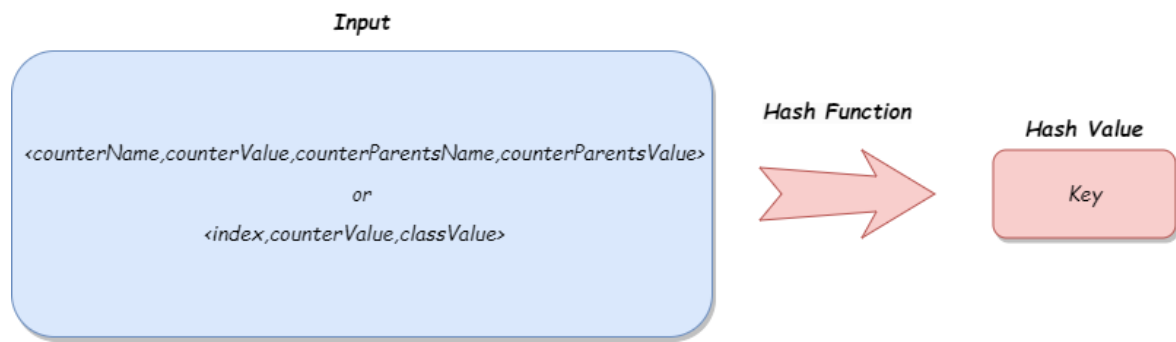


Figure 18: Hashing

The hashing process is not performed each time a message is sent, instead, it is carried out during the process of the *Initialization*. Each counter is converted into a *unique key* that will be used throughout the entire process. This allows us to reduce the size of each message and create a fast retrieval mechanism for the counters, utilizing an appropriate structure (*Hash Map*).

Both the *Worker* and *Coordinator* components include a *side-output*, which serves as a *logging* mechanism. This mechanism enables us to log various important metrics related to our system, including *throughput*, *communication cost* at a given time, and more.

Further details on the *project structure* and *setup*, can be found in Appendix. Additionally, the code for the project is available at [27].



## Chapter 6:

# Experimental evaluation

The experimental evaluation comprises a wide range of experiments covering both the first approach (See *Chapter 4.1*) and the proposed approach (See *Chapter 4.2*). The first section validates the functionality of the proposed approach, while the second section refers to the system's capability to handle large volumes of datasets. The third section provides a comparison between the two approaches. All the experiments are conducted using *Bayesian Networks*, with similar results observed for *Naïve Bayes Classifiers*.

In the first approach, we conducted experiments using two types of counters: *RANDOMIZED* and *DETERMINISTIC* counters, in conjunction with the *BASELINE*, *UNIFORM*, and *NON\_UNIFORM* algorithms. For the second approach, which utilizes the FGM method, we conducted various experiments using the *BASELINE* and *UNIFORM* algorithms. It should be noted that the Bayesian Network structure is predetermined, and our objective is to learn the parameters.

### **Testing Setup**

All the experiments described in this thesis were conducted on the *SoftNet Cluster* of the *SoftNet lab* [28]. The *SoftNet Cluster* is composed of *12 Quad Core Xeon X3323 2.5GHz, 8 GB DDR3 RAM*. The version of *Apache Flink version* is *1.10.0* and the version of the *Apache Kafka Connector version* is *1.9.3*.

## 6.1 Datasets

All the *Bayesian Networks* used in our experiments are *real-world Bayesian Networks* and are available in the repository [29]. Additionally, the *Bayesian Networks* had been extensively studied in prior studies. We selected *Bayesian Networks* from different size categories. Specifically, we chose the *ALARM* [30] as a *small-size network* ( $\leq 40$  nodes), the *HEPAR2* [7] as a *medium-size network* ( $\leq 100$  nodes), and the *LINK* as a *large-size network* ( $\leq 1000$  nodes). Finally, all the *Bayesian Networks* in our experiments refer to *Discrete Bayesian Networks*.

Table 4 provides an overview of the *Bayesian Networks* used in the experiments. It includes the number of nodes in each Bayesian Network, which corresponds to the size of the *input feature vector*. The second column indicates the number of edges, representing the number of *conditional dependencies* among the nodes in each *Bayesian Network*. Additionally, the table displays the number of parameters required to define the *joint probability distribution* of each *Bayesian Network*, while the last column indicates the number of counters required to maintenance, for estimating the parameters of each *Bayesian Network*.

<i>Dataset</i>	<i>Number of nodes</i>	<i>Number of edges</i>	<i>Number of parameters</i>	<i>Number of counters</i>
<i>ALARM</i>	<i>37</i>	<i>46</i>	<i>509</i>	<i>983</i>
<i>HEPAR II</i>	<i>70</i>	<i>123</i>	<i>1453</i>	<i>2816</i>
<i>LINK</i>	<i>724</i>	<i>1125</i>	<i>14211</i>	<i>26609</i>

Table 4: Bayesian Networks

### **Training Data**

Regarding the *training dataset*, we generated datasets for each *network* from the *joint probability distribution* using the *Forward Sampling method* (See Chapter 2.4). The size of datasets for each network varied from 5K to 50M. These generated datasets were used exclusively for learning the parameters of each Bayesian Network.

### **Testing Data**

The *testing dataset* contains *queries* about specific *events*, intending to evaluate the *accuracy* of the system. Particularly, we measure the system’s ability to accurately estimate the probability of various events. To accomplish this, we generate *1000 events* based on the *joint probability distribution* of the *Bayesian Network*. The probability of these events is estimated using the learning parameters of the trained network. All events in the testing dataset pertain exclusively to probability queries regarding the joint probability distribution. It is important to note that there are no limitations on the values of the probability events. The testing *dataset* encompasses both *highly unlikely* and *highly likely* queries, with the only requirement being that the assigned probabilities must be non-negative, in adherence to the basic probability axiom.

## **6.2 Performance metrics**

### *Communication cost*

The illustrated *communication cost* represents the total number of bytes of messages. Encompassing both the *upstream* and *downstream communication cost*.

### *Error to Ground Truth (GT)*

For each *testing event*, we calculate the probability using the *learning parameters* of the *approximate model*, which corresponds to the *trained network*. We then compare this probability to the *ground truth* probability event obtained using the values of the *ground truth* parameters of the network. Consequently, the metric measures the *error* of the *ground truth* (GT) probability, denoted as *Error to GT*.

### *Error to EXACTMLE*

For the *BASELINE*, *UNIFORM*, and *NONUNIFORM* algorithms, we measure the *error* to EXACTMLE and denote it as *Error to EXACTMLE*.

## 6.3 Experimental Results

For all the following experiments are reported the mean value of five independent *runs*. Furthermore, the *Bayesian Networks* do not appear significant differences between  $J_i, K_i$  among the nodes, resulting in the differences among *BASELINE*, *UNIFORM*, and *NONUNIFORM* algorithms both approaches are negligible. We focus on the differences between of two approaches. We set  $k = 16$ ,  $\varepsilon = 0.1$  and  $\delta = 0.1$ , unless otherwise specified.

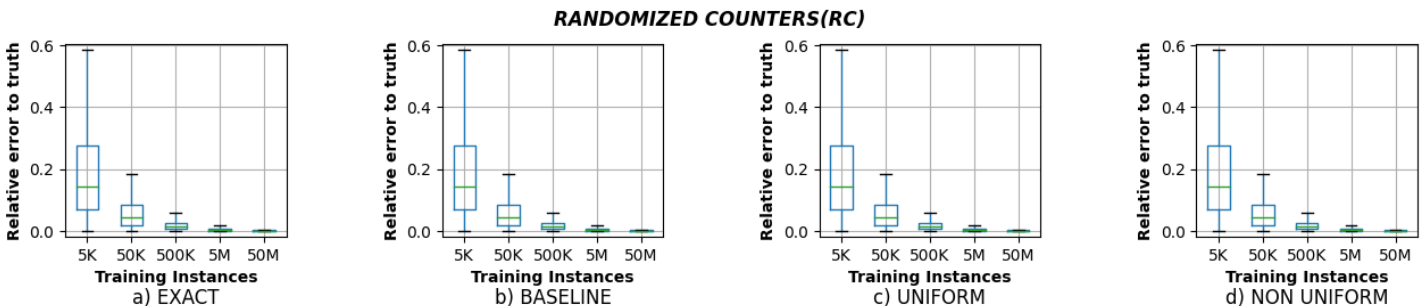
We present the results of four categories of experiments. The first category measures the *communication cost* while varying the number of *training instances*, and the second category examines the *communication cost* while varying the *approximation factor  $\varepsilon$ . Last but not least, the third category measures the *communication cost* while varying the number of *workers*. Lastly, in the fourth category, we evaluate the scalability of the system. In all cases, we compare the two approaches themselves, and each approach with *EXACT counters*, that is the *EXACTMLE*.*

### 6.3.1 Communication cost related to the number of training instances

The first category focuses on the communication cost while varying the number of *training instances*. The size of datasets ranges from 5K to 50M and the dataset used is the *HEPAR II*.

In this category, we compare the *Error to GT* between the two approaches. For the first approach, we measure the error for both *RANDOMIZED* and *DETERMINISTIC* counters, while for the second approach, we use the *FGM* method. The *Error to GT* represents the *absolute error* of the *probability queries*.

As the number of training instances increases for both approaches, the error decreases. This is expected because, with more training instances, more information about the parameters is obtained, resulting in more accurate parameter estimation. Furthermore, the *interquartile range* shrinks with the increase in training instances, indicating a reduction in the variance of the *probability query* errors. Both approaches achieve good accuracy, for instance, after 5M instances, the median error is less than 1% for each approach. The diagrams illustrating the *error to GT* of the *HEPAR II* network in both approaches can be found in the Appendix.



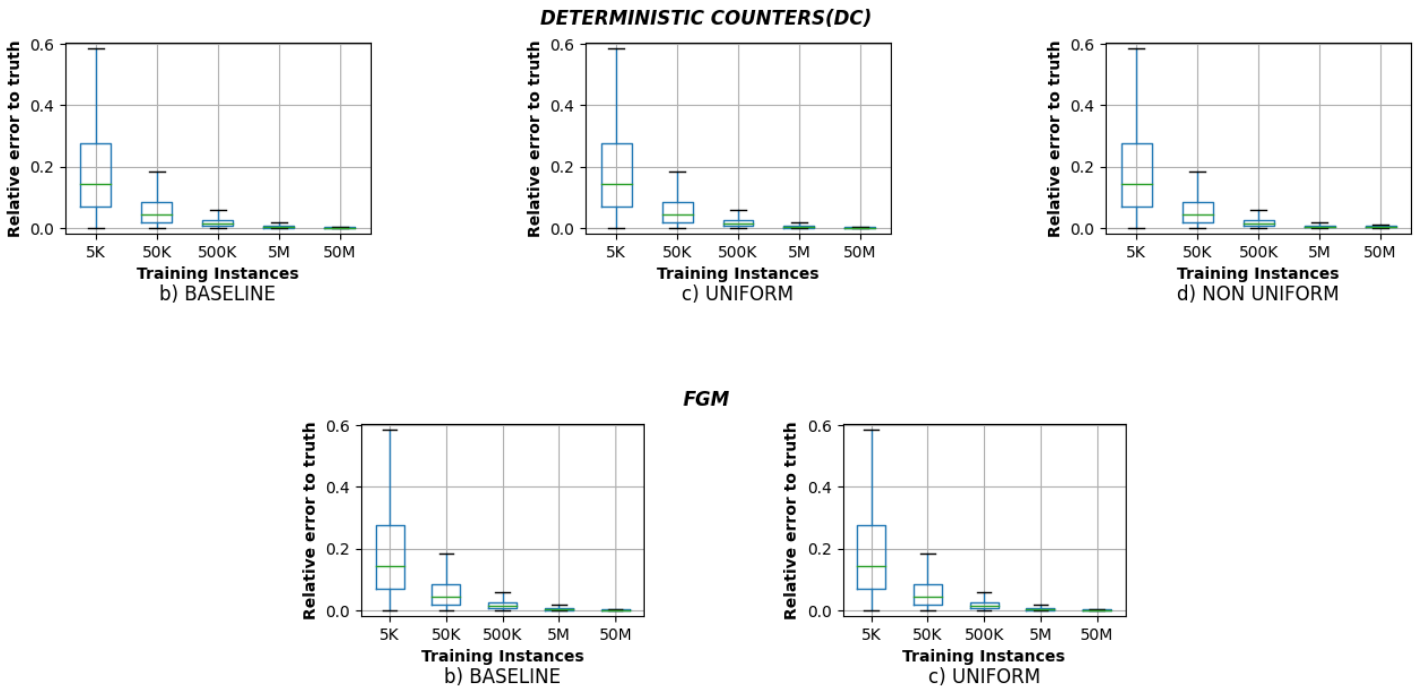


Figure 19: Error to GT related to the training instances for both approaches, for the HEPAR II dataset

We compare the *Error to EXACTMLE* to the number of *training instances* for both approaches. In this case, the *Error to EXACTMLE* represents the *mean absolute error* of the probability queries. This error can be attributed to two sources: *statistical error* and *approximation error*. The *statistical error* arises due to the number of training instances and depicts the error relating to the *ground truth values* of the parameters. The *approximation error*, on the other hand, captures the error between the learnt model and the model obtained using *EXACT counters*, which is influenced by the *approximation factor*  $\epsilon$ .

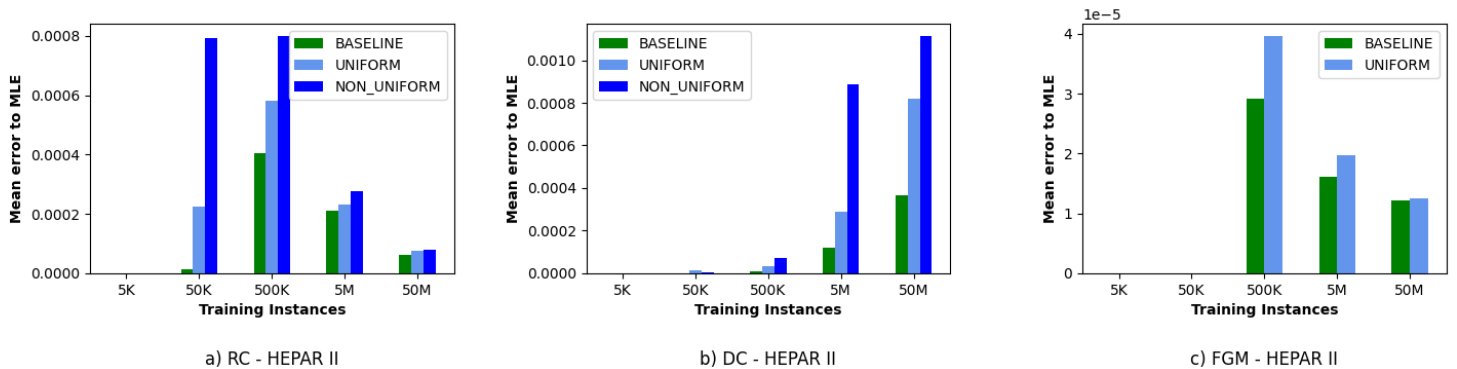


Figure 20: Error to EXACTMLE related to the number of training instances for both approaches for the HEPAR II dataset

Our objective is to control the *approximation error* to the *EXACTMLE*. We observe that for both approaches the error approximately remains bounded as the number of training instances is varied. The latter confirms the functionality of each approach. We guarantee for each approach combined with the *BASELINE*, *UNIFORM*, and *NON\_UNIFORM* algorithms that the error is always bounded, where the *error guarantees* are always specified by the *approximation factors*  $\epsilon$ ,  $\delta$ . For a small number of training instances ( $\leq 50K$ ), the error is negligible due to the *tight bounds* for each counter. The latter holds since the values of the counters are quite small in this case. For example, when  $\epsilon = 0.1$  the approximation factor for each counter is roughly in the order of  $10^{-4}$ , therefore the *bounds* are extremely tight for small values of the counters.

The *BASELINE* obtains the smallest *error*, which corresponds to the best *accuracy*, closely followed by the *UNIFORM* and *NON\_UNIFORM* algorithms. The latter holds because the *BASELINE* algorithm is accompanied by *tight bounds* for each *counter* compared to the others. The purpose of this experiment is to prove the functionality both of approaches and not to infer results among the *BASELINE*, *UNIFORM*, and *NON\_UNIFORM* algorithms.

Additionally, we measure the *communication cost* related to the number of *training instances* for both approaches. Note that the y-axis is in logarithmic scale.

Firstly, there are no significant differences among the *BASELINE*, *UNIFORM*, and *NON\_UNIFORM* algorithms. As mentioned earlier, there are no significant differences between  $J_i$  and  $K_i$  among the nodes of the *HEPAR II* network. Generally speaking, the *UNIFORM* and *NON\_UNIFORM* algorithm is quite close while both algorithms appear to be better than the *BASELINE* algorithm as the number of training instances increases.

As the number of training instances increases, the difference to *EXACTMLE* also increases for both approaches. This is desirable as it proves that both approaches can handle large volumes of datasets sufficiently while using as little communication cost as possible. In the case of the first approach, both *RANDOMIZED* and *DETERMINISTIC* counters achieve up to an order of magnitude less communication cost of *EXACTMLE*. This practically means that they send 10 times fewer messages than *EXACTMLE*. On the other hand, the second approach achieves up to two orders of magnitude less communication cost than *EXACTMLE*, which means that it sends 100 times fewer messages than *EXACTMLE*.

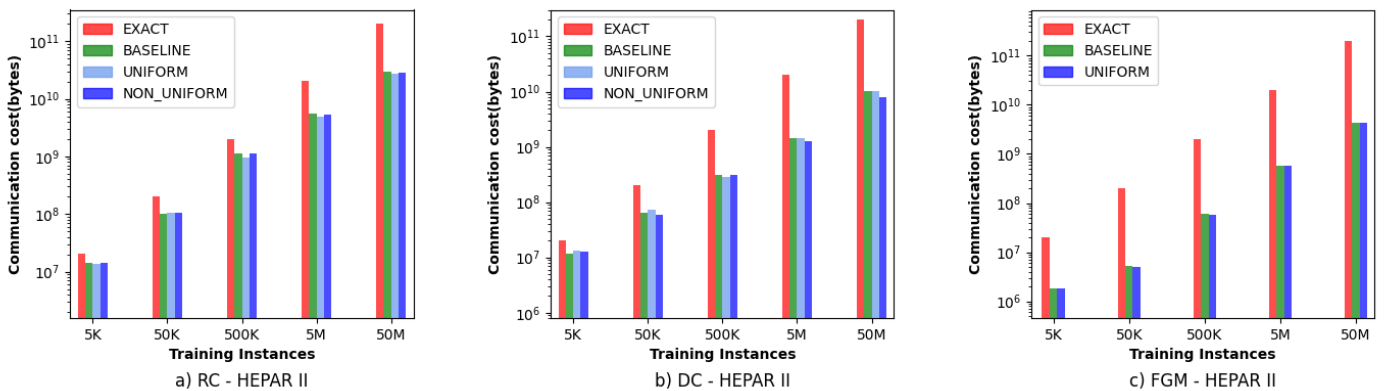
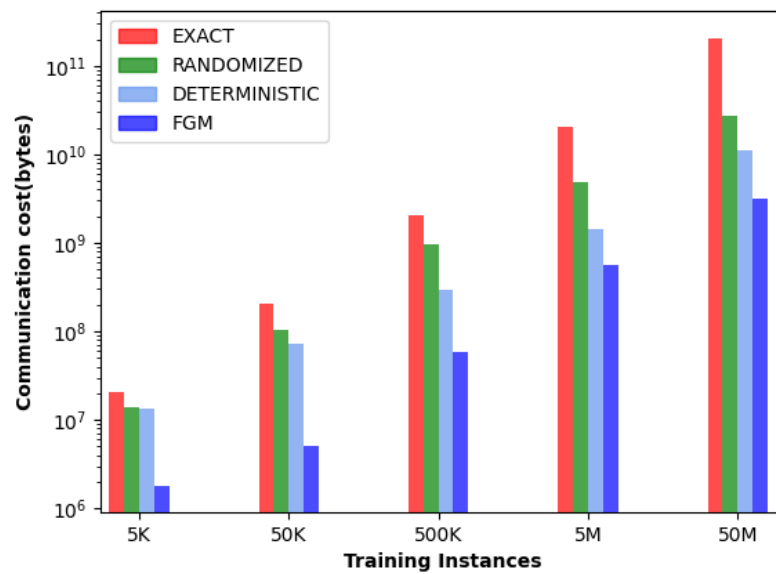


Figure 21: Communication cost related to the number of training instances for both approaches from the HEPAR II dataset

The communication cost tends to increase logarithmically with the number of training instances. This trend is particularly evident when examining the LINK dataset. The results for both the LINK and ALARM datasets can be found in the Appendix.

In Figure 22, for each approach among the *BASELINE*, *UNIFORM*, and *NON\_UNIFORM* algorithms are used the algorithm which resulted in the lowest communication cost. We observe that the proposed approach outperforms the first approach, which utilizes *individually approximate distributed counters* (either *RANDOMIZED* or *DETERMINISTIC*), by up to an order of magnitude in terms of communication cost. When comparing the *RANDOMIZED* and *DETERMINISTIC* counters, we observe that *DETERMINISTIC* achieves a 50% improvement in communication cost (See. Chapter 4.1.1.3).



HEPAR II

Figure 22: Communication cost related to the number of training instances for both approaches from the HEPAR II dataset -Best results

### 6.3.2 Communication cost related to the approximation factor $\epsilon$

In this section, we compare the *communication cost* in relation to the *approximation factor*  $\epsilon$ . We vary the *approximation factor*  $\epsilon$  in the range of  $[0.02, 0.12]$  with an increment of 0.02 and measure the *communication cost* for both approaches. The size of datasets is 100K, 500K, 1M, 5M respectively, using the *HEPAR II dataset*. The results for both approaches combined with the *UNIFORM* algorithm are presented here, while the results of the other algorithms can be found in Appendix.

For both approaches and all algorithms, we observe that the *communication cost* does not closely follow the changes in the approximation factor  $\epsilon$ . This implies that the *communication cost* is not sensitive to variations in the *approximation factor*  $\epsilon$  for both approaches. In the first approach, with both types of counters, we initially observe a reduction in the communication cost by around 15% – 20% when the approximation factor  $\epsilon$  changes from 0.02 to 0.04. However, this reduction gradually diminishes as the *approximation factor*  $\epsilon$  increases. On the other hand, the communication cost of the second approach remains stable across different values of the approximation factor  $\epsilon$ . In other words, the approximation factor  $\epsilon$  has little impact on the communication cost of the FGM method. This can be attributed to how we define safe functions to achieve our objective.

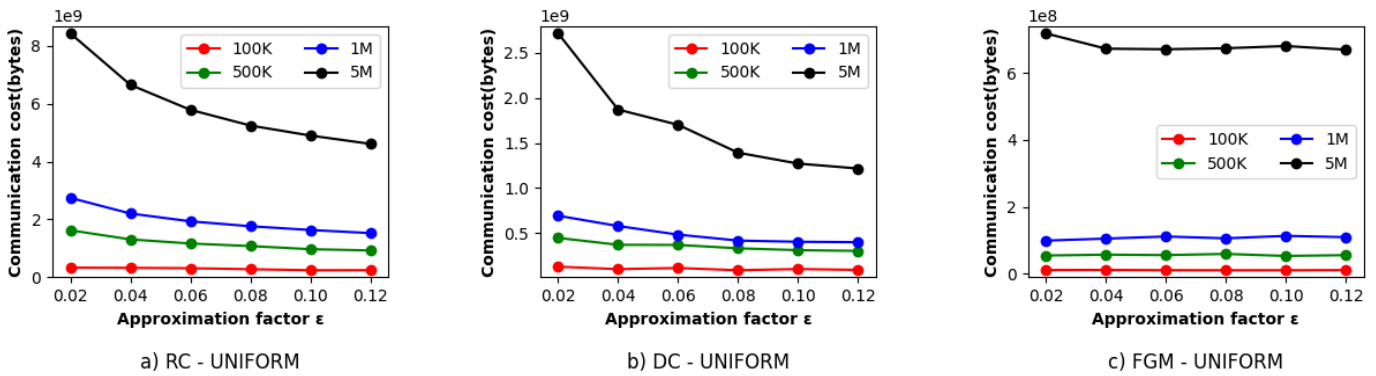
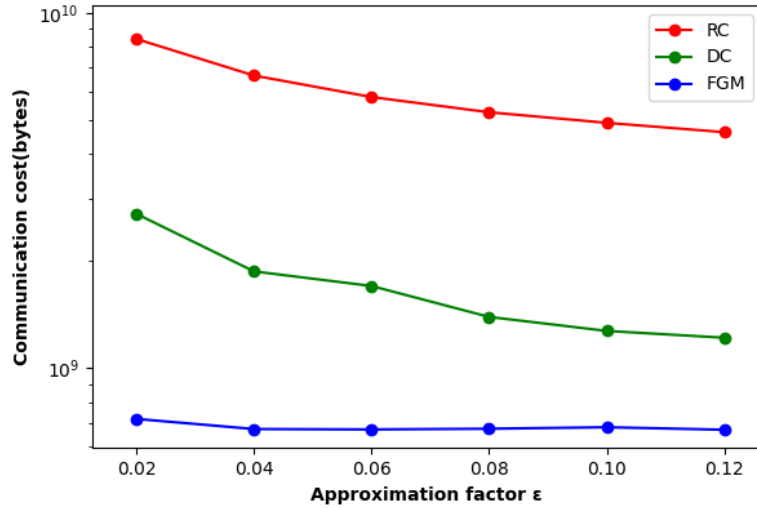


Figure 23: Communication cost related to the approximation factor  $\epsilon$ , for the *HEPAR II dataset* and *UNIFORM* algorithm

By comparing the two approaches in terms of the impact of the *approximation factor*  $\epsilon$ , we find that the proposed approach achieves better results. Both types of counters in the first approach exhibit a similar dependency on the *approximation factor*  $\epsilon$ . Note that the y-axis of the Figure 24 is in logarithmic scale.



UNIFORM, HEPAR II, 5M

Figure 24: Communication cost related to the approximation factor  $\epsilon$  for both approaches, for the HEPAR II dataset and UNIFORM algorithm

Furthermore, we provide a diagram depicting the *Error to GT* in relation to *approximation factor*  $\epsilon$  for both approaches using the UNIFORM algorithm. The results of other algorithms exhibit similar trends. In this case, the error denotes the *mean absolute error* of the *probability queries*.

As the *approximation factor*  $\epsilon$  increases, the *error* does not show significant changes (although some differences exist which are not significant). This is because the *approximation factor*  $\epsilon$  only controls the *approximation error*, which is considerably smaller than the *statistical error* (up to two orders of magnitude smaller). As a result, the impact of the approximation error is overshadowed, leading to negligible changes in the overall error. On the other hand, we do observe changes in the error among different datasets. Specifically, the error decreases as the number of training instances increases. This is attributed to the reduction in statistical error as more training instances are utilized. Lastly, the accuracy achieved is quite good given the probabilities of the queries, the accuracy is less than 6% for both approaches.

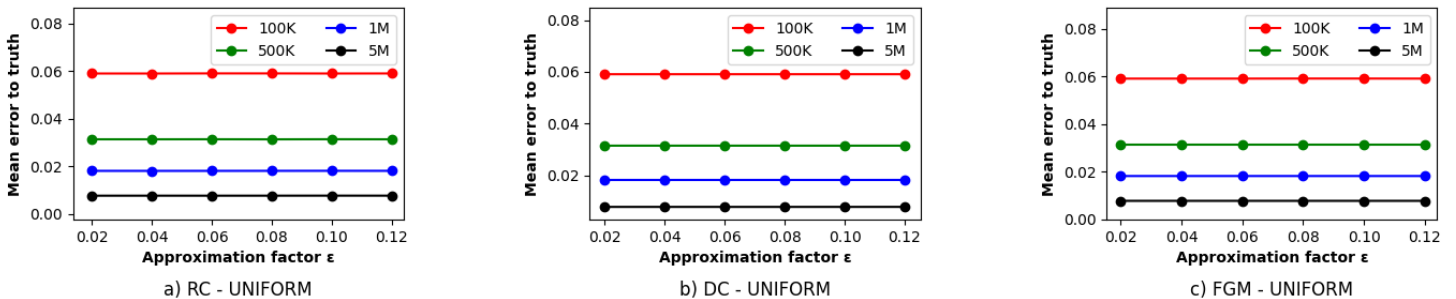


Figure 25: Error to GT related to the approximation factor  $\epsilon$  for both approaches, for the HEPAR II dataset and UNIFORM algorithm



### 6.3.3 Communication cost related to the number of workers

In this section, we compare the *communication cost* in relation to the number of sites. Specifically, the number of sites is varied in the range of [2,64]. The size of the dataset is 500K and refers to the *HEPAR II* dataset. We present the communication cost for both approaches using the *UNIFORM* algorithm, while the results of the other algorithms can be found in Appendix. Note that the y-axis is in logarithmic scale.

Upon analysis, we observe the *communication cost* increases sub-linearly to the number of sites for both approaches. The first approach, utilizing *RANDOMIZED* counters, demonstrates the most favorable dependency on the communication cost, in line with the theoretical results. On the other hand, both the *FGM* and *DETERMINISTIC* counters exhibit similar results in terms of their dependency on the number of sites.

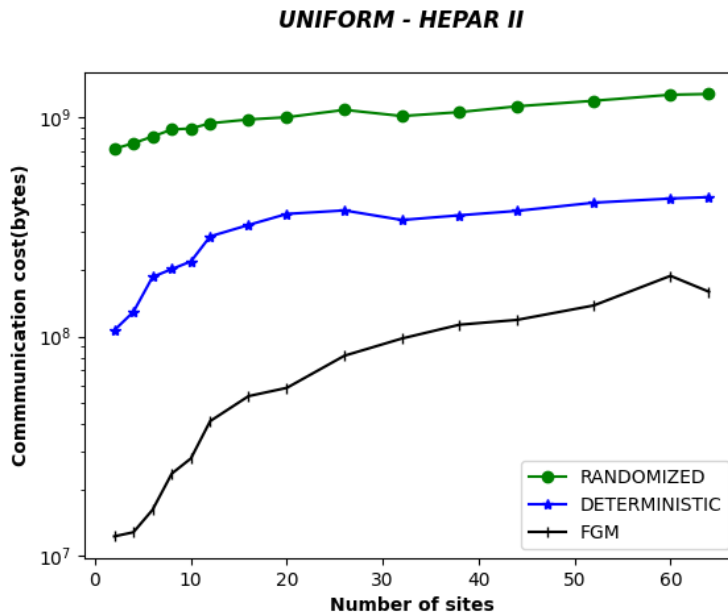


Figure 26: Communication cost related to the number of workers sites for both approaches, from the *HEPAR II* dataset and *UNIFORM* algorithm

### 6.3.4 Scalability

Until now, we measure the *communication cost* for different parameters of our system. In this section, we evaluate the performance of our system for both approaches. using two metrics: *throughput* and *runtime*. The *throughput(events/sec)* denotes the average of training instances that the system can handle per second, while the *runtime(sec)* represents the time between the first and last message received from the Coordinator during the training process.

We conduct two categories of experiments to evaluate the system’s performance using these metrics. In the first experiment, we measure the two metrics varying the number of sites. In the second experiment, we measure the two metrics varying the number of parallelism. Parallelism can be interpreted as the number of subtasks or the size of resources utilized by the system.

In each experiment, we compare the two approaches themselves and each approach with the EXACTMLE. All the experiments are conducted using the *HEPAR II* dataset with a size of 500K instances. We present the results using the metric of *throughput*, the results of *runtime* can be found in Appendix. To mention that the results of *runtime* are similar to the *throughput* but with a focus on reducing the runtime.

In the first experiment, we measure both *throughput(event/sec)* and *runtime(sec)* in relation to the number of sites. The number of sites is varied in the range of [2,10] with step 2. Note that the number of sites does not include the Coordinator.

In the first approach using both *RANDOMIZED* and *DETERMINISTIC counters*, we observe that the throughput does not increase proportionally to the number of sites after the number of sites exceeds 6. This suggests that adding more sites does not significantly improve the system's throughput beyond a certain point. As the number of sites increases, the *communication cost* also increases. Notable, the largest increase in the communication cost is observed when using a small number of sites (*Figure 26*), therefore we choose the number of sites in the range of [2,10].

The augmentation in the *communication cost*, particularly in the initial state where all the counters appeared to have *violations*, resulted in a significant increase in the volume of messages sent to the Coordinator. As a consequence, *backpressure* is generated on the sites. The *backpressure* appears only in the initial state where the counters are quite small accompanied by tight bounds. It is important to note that this backpressure is temporary and tends to diminish over time. In contrast, we observe that the *throughput* of the EXACTMLE remains relatively stable regardless of the number of sites, indicating that its communication cost is indeed substantial, resulting in lasting backpressure on the sites. As a result, it does not scale well in terms of the number of sites.

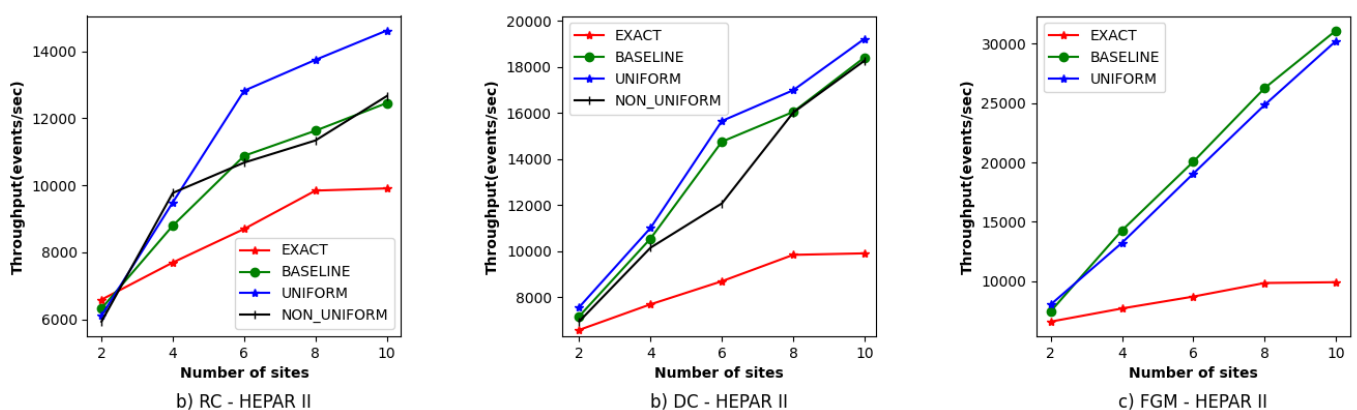


Figure 27: Throughput(event/sec) related to the number of sites for both approaches, for the HEPAR II dataset

Considering the second approach, we observe that the *throughput* increases linearly according to the number of sites, indicating that there is no *backpressure* on the sites. As a result, the proposed approach achieves the ideal scaling without experiencing backpressure issues. The differences observed between the two approaches suggest variations in their communication cost. Specifically, the second approach exhibits significantly lower communication costs compared to the first approach. Furthermore, differences in *throughput* and *runtime* among the *BASELINE*, *UNIFORM*, and *NON\_UNIFORM* algorithms can be attributed to variations in their *communication costs*, which are directly proportional to their differences.

We measure the *throughput* and the *runtime* of the first approach after the initial state, where all the counters appeared with violations. We observe that the *throughput* increases proportional to the number of sites, while the *runtime* decreases proportionally to the number of sites for both types of counters. This confirms that the *backpressure* after the initial state begins to diminish over time, and as a result, the first approach with both types of counters achieves ideal scaling.

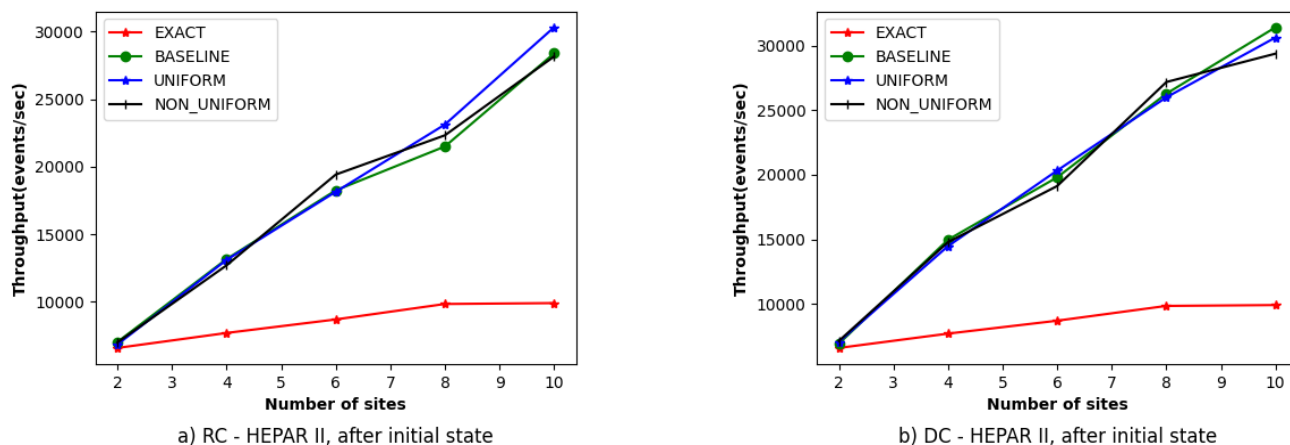


Figure 28: Throughput(events/sec) related to the number of sites for the first approach for the HEPAR II dataset HEPAR II, after the initial state

We present a comparison of the two approaches in terms of throughput (Figure 29). The chosen algorithm for each approach among the *BASELINE*, *UNIFORM*, and *NON\_UNIFORM* algorithms is the one that achieves the best performance i.e., the algorithm with the highest *throughput* for each approach (while for the *runtime* is used the algorithm with the lowest runtime). For the first approach, the *throughput* is measured without considering an initial state. We observe that the proposed approach achieves higher throughput than the first approach, indicating differences in their communication cost are significant. Furthermore, both the second and the first approaches demonstrate higher *throughput* compared to the *EXACTMLE*, indicating substantial differences in their communication cost.

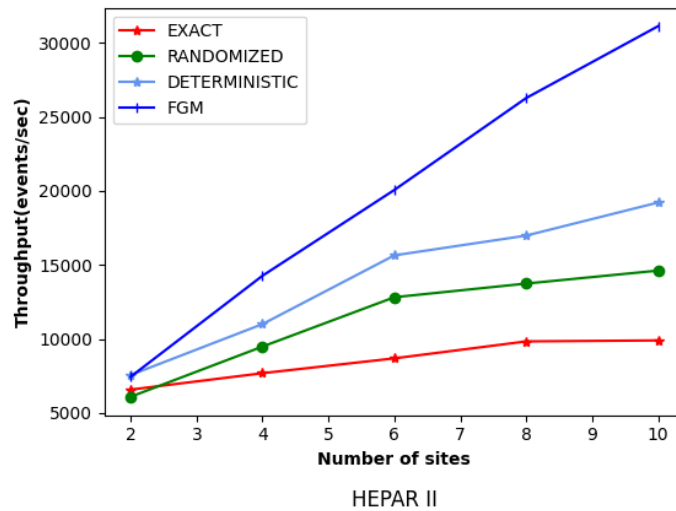


Figure 29: Throughput(events/sec) related to the number of sites for both approaches, for the HEPAR II dataset -Best results

In the second experiment, we measure the *throughput(event/sec)* and the *runtime(sec)* in relation to the number of parallelism. Specifically, the parallelism is varied in the range of [2,16] with a step of 2 while the number of sites remains fixed at 16.

Similar to the findings in the previous experiment, we observe that in the first approach, as the number of parallelism increases, the variations in both *throughput* and the *runtime* do not increase proportionally to the number of sites.

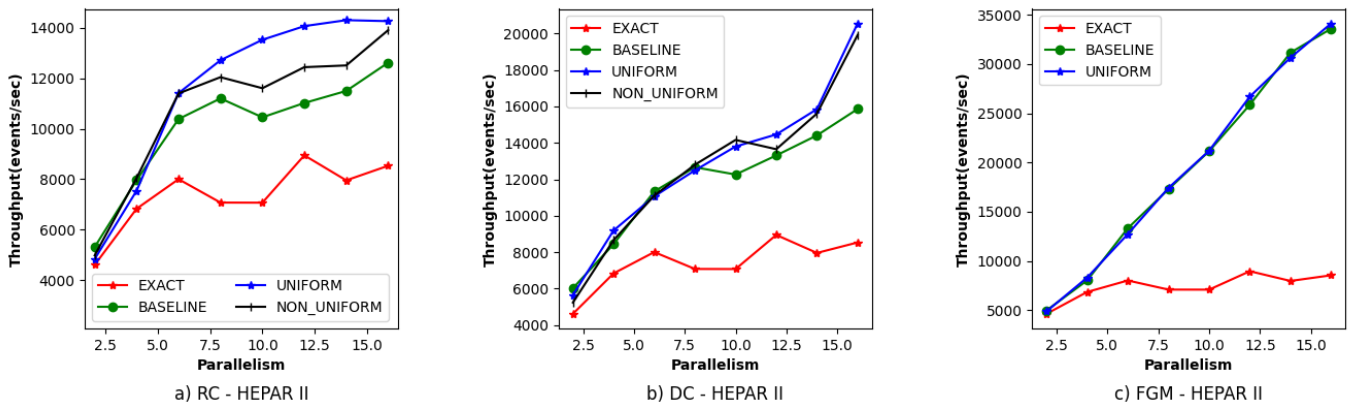


Figure 30: Throughput(events/sec) related to the number of parallelism for both approaches for the HEPAR II dataset

This can be attributed to the fact that as the number of parallelism increases, the rate at which messages arrive at the Coordinator surpasses the rate at which the messages can be processed, resulting in *backpressure* on the sites. The *backpressure* is observed only in the first approach, particularly during the initial state where all the counters have tight bounds and violations are more common. This leads to a significant increase in the communication cost. However, as the system progresses beyond the initial state, the *backpressure* gradually diminishes (Figure 31), where we measure the *throughput* after the initial state for both types of counters in relation to the number of sites.

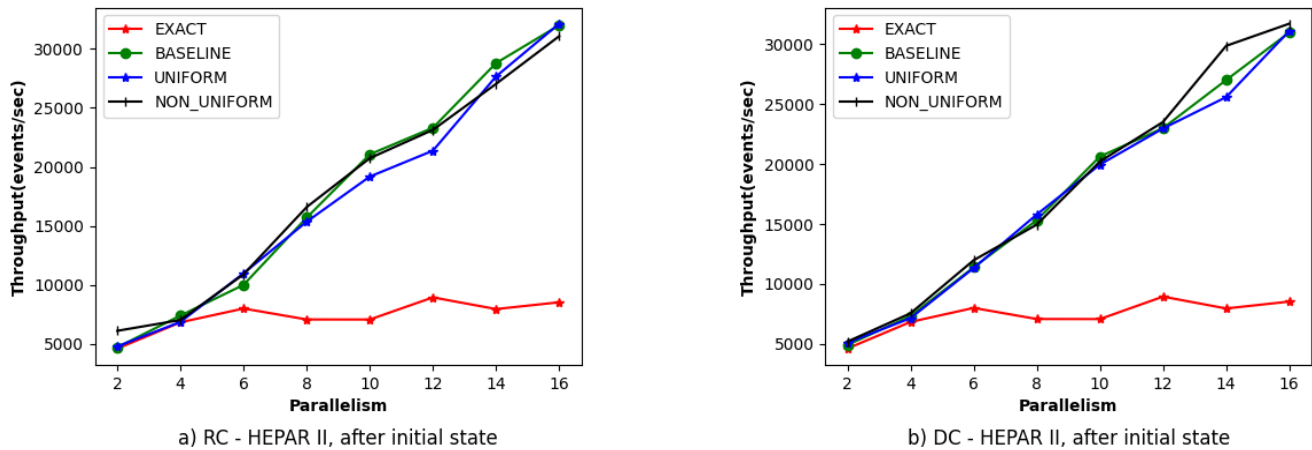


Figure 31: Throughput(events/sec) related to the number of parallelism for the first approach for the HEPAR II dataset, after the initial state

Considering the second approach, we observe that the increase in the *throughput* is linear in relation to the number of parallelism, indicating that there are no *backpressure* issues on the sites. This suggests that the proposed approach achieves ideal scaling in terms of parallelism (i.e., the number of resources that can be utilized by the system).

On the other hand, the *throughput* of the *EXACTMLE* remains relatively stable to the increase of the number of parallelism, indicating that the rate at which the messages arrived at the Coordinator is substantial. This is due to the *communication cost* associated with the *EXACTMLE* is substantial. As a result, *backpressure* is consistently generated on the sites. Therefore, the *EXACTMLE* does not achieve the desired scaling to the number of *resources*.

We present a comparison of the two approaches in terms of parallelism (Figure 32). Among the *BASELINE*, *UNIFORM*, and *NON\_UNIFORM* algorithms, for each approach is used the algorithm that achieves the best performance for each approach is selected (in the *runtime*, the algorithm with the lowest runtime, respectively). The proposed approach outperforms both types of counters in terms of throughput, indicating its ability to effectively utilize the available resources and achieve optimal scaling of the system. The significant difference in throughput between the two approaches highlights the difference in their communication costs. Additionally, the *throughput* of both the first and second approaches are better than that of the *EXACTMLE*, which confirms their differences in *communication cost*.

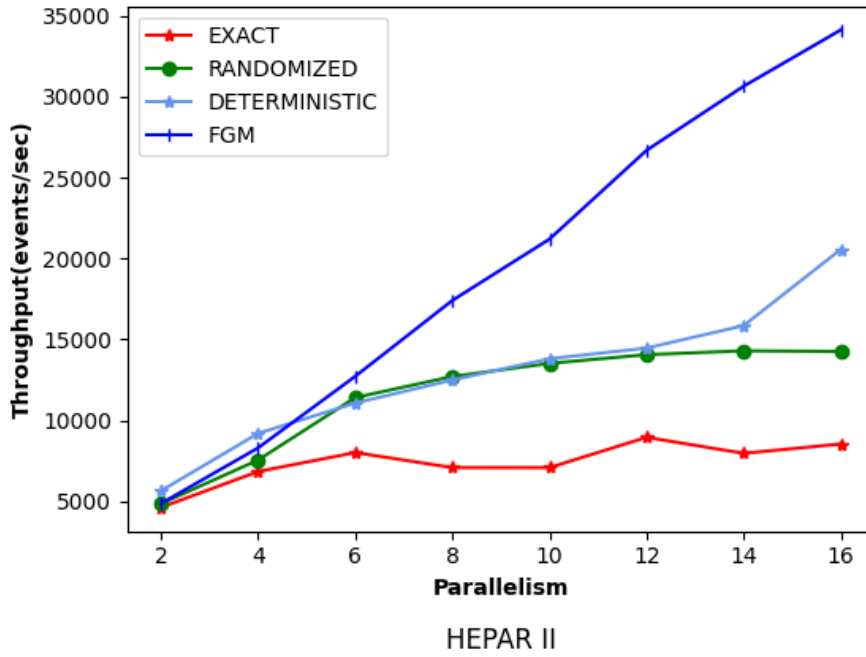


Figure 32: Throughput(events/sec) related to the number of parallelism for both approaches for the HEPAR II dataset- Best results

### 6.3.5 Dummy Father

In this section, we present the results of the *Dummy Father (DF)* method in conjunction with the *NON\_UNIFORM* algorithm. The results are exclusively focused on the first approach (See Chapter 4.2.1).

We measure the *communication cost* of the *NON\_UNIFORM* combined with the *Dummy Father (DF)* method and then compare the results to the *communication cost* of the *NON\_UNIFORM* algorithm, using both the *RANDOMIZED* and *DETERMINISTIC* counters of the first approach. The *datasets* are varied in the range of *5K* to *50M* and are related to the *HEPAR II* and *ALARM* datasets, respectively. Note that the y-axis is in logarithmic scale.

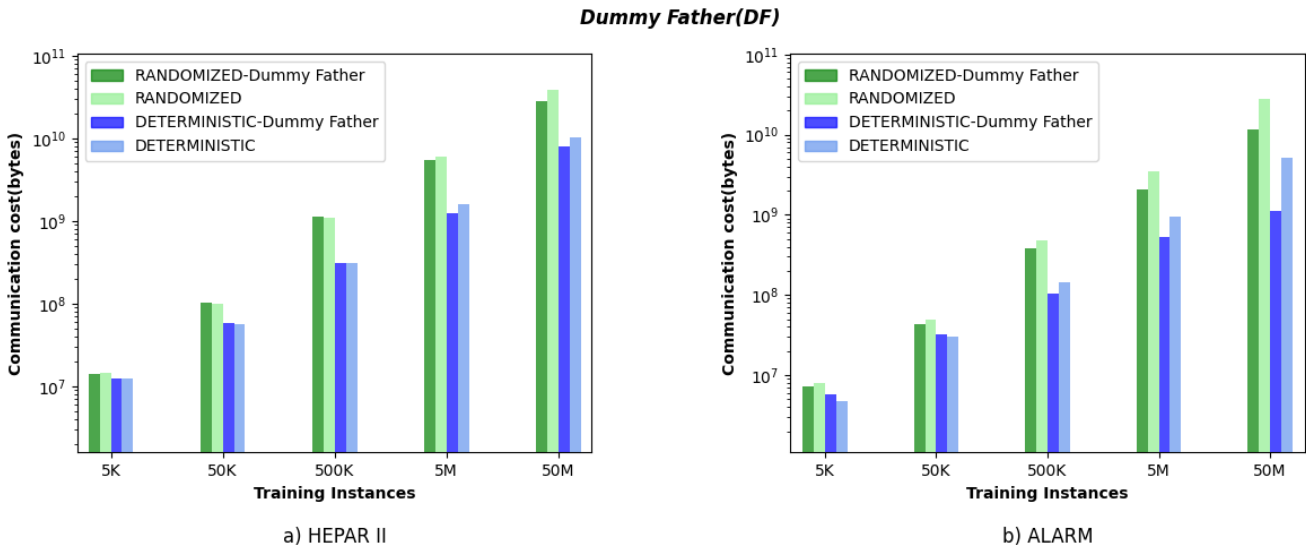


Figure 33: Communication cost in conjunction with the *Dummy Father (DF)* method for *HEPAR II*, *ALARM* datasets

As the number of training instances increases, the difference in communication cost begins to increase as well. This is because each counter is required to send fewer messages. We achieve a 40% reduction in the *communication cost* of the *RANDOMIZED* counters and a 50% reduction in *communication cost* for the *DETERMINISTIC* counters. To mention that the reduction in *communication cost* is mostly attributed to the prevention of maintaining *EXACT* counters for *orphan* nodes. Therefore, the differences may vary among different networks. Specifically, in the case of the *ALARM* dataset, we can prevent the maintenance of 26 *EXACT* counters, while in *HEPAR II* we can prevent the maintenance of 18 *EXACT* counters.

## Chapter 7:

# Conclusions

### 7.1 Conclusions

We propose a different approach for learning parameters ensuring the continuous maintenance of a *Bayesian Network* over the *continuous distributed model*, using the *FGM* method. We achieve a reduction in communication cost compared to the EXACTMLE while ensuring *error guarantees* on the estimation of the *probabilities queries* of the *joint probability distribution* of a *Bayesian Network*, with the latter being confirmed from the experimental evaluation. Additionally, we present a comprehensive and extensive analysis of the first approach using both the RANDOMIZED and DETERMINISTIC counters combined with the BASELINE, UNIFORM, and NON\_UNIFORM algorithms. Finally, we integrate both approaches into a *distributed streaming framework* like *Apache Flink*.

### 7.2 Future Work

Several extensions can be explored for both approaches. One interesting extension is the *structure* learning of a *Bayesian Network*, where the structure of the *Bayesian Network* is assumed to be known in advance, now the learning of the structure is accomplished in an online fashion as new data is received at the sites. Considering the second approach, another interesting extension is the search for “better” safe functions. This could potentially lead to a further reduction in communication cost by optimizing the selection of safe functions. Additionally, the adaptation of the *Graphical Model sketches* [32] could be explored to achieve a reduction in communication cost. *Graphical sketches* provide a *space-efficient* representation of the entire *joint probability distribution*, allowing for more efficient communication compared to *frequency vectors*, while ensuring the appropriate *error guarantees*.



# Appendix

## Experimental results

### Communication cost related to the number of training instances

We present the *mean error to GT* for both approaches. To mention that the differences among the *BASELINE*, *UNIFORM*, and *NON\_UNIFORM* algorithms are not significant because the *statistical error* outweighs the *approximation error*. As a result, the dominant factor is the *statistical error*, and there are no differences among the algorithms. The *error* decreases as the number of *training instances* increases which can be attributed to the decrease in the *statistical error*. In terms of accuracy, both approaches perform quite well. For example, for the datasets exceeding 50K training instances, the error is less than 10% for each approach. The captured error is considered good when considering the *probability queries*.

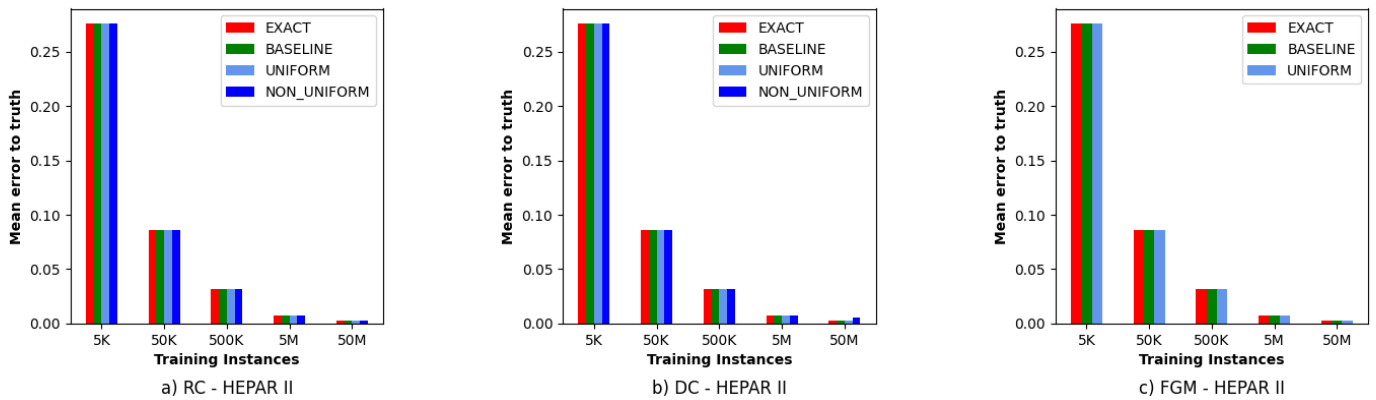


Figure 34: Mean error to GT related to the number of training instances for both approaches for the HEPAR II dataset

We analyze the *communication cost* in relation to the number of *training instances* for both approaches using the *LINK* and *ALARM* datasets. For each approach and network, among the *BASELINE*, *UNIFORM*, and *NON\_UNIFORM* algorithms selected the algorithm with the best performance.

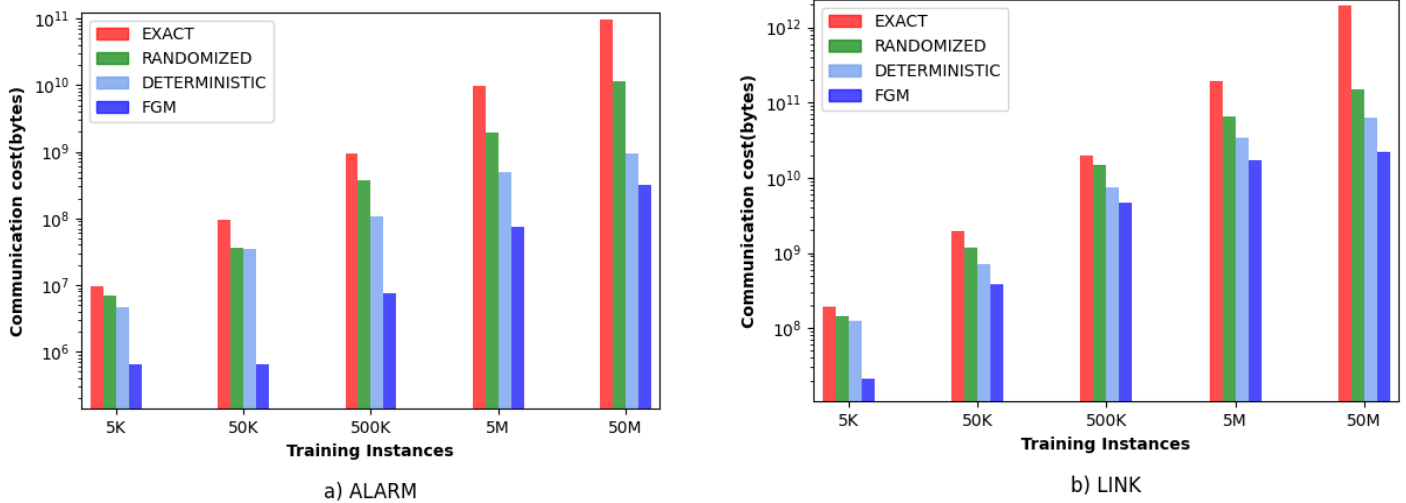


Figure 35: Communication cost related to the number of training instances for LINK, ALARM datasets

### Communication cost related to the approximation factor $\epsilon$

We present the *communication cost* in relation to the *approximation factor*  $\epsilon$ , using the *BASELINE*, *UNIFORM*, and *NON\_UNIFORM* algorithms for each approach.

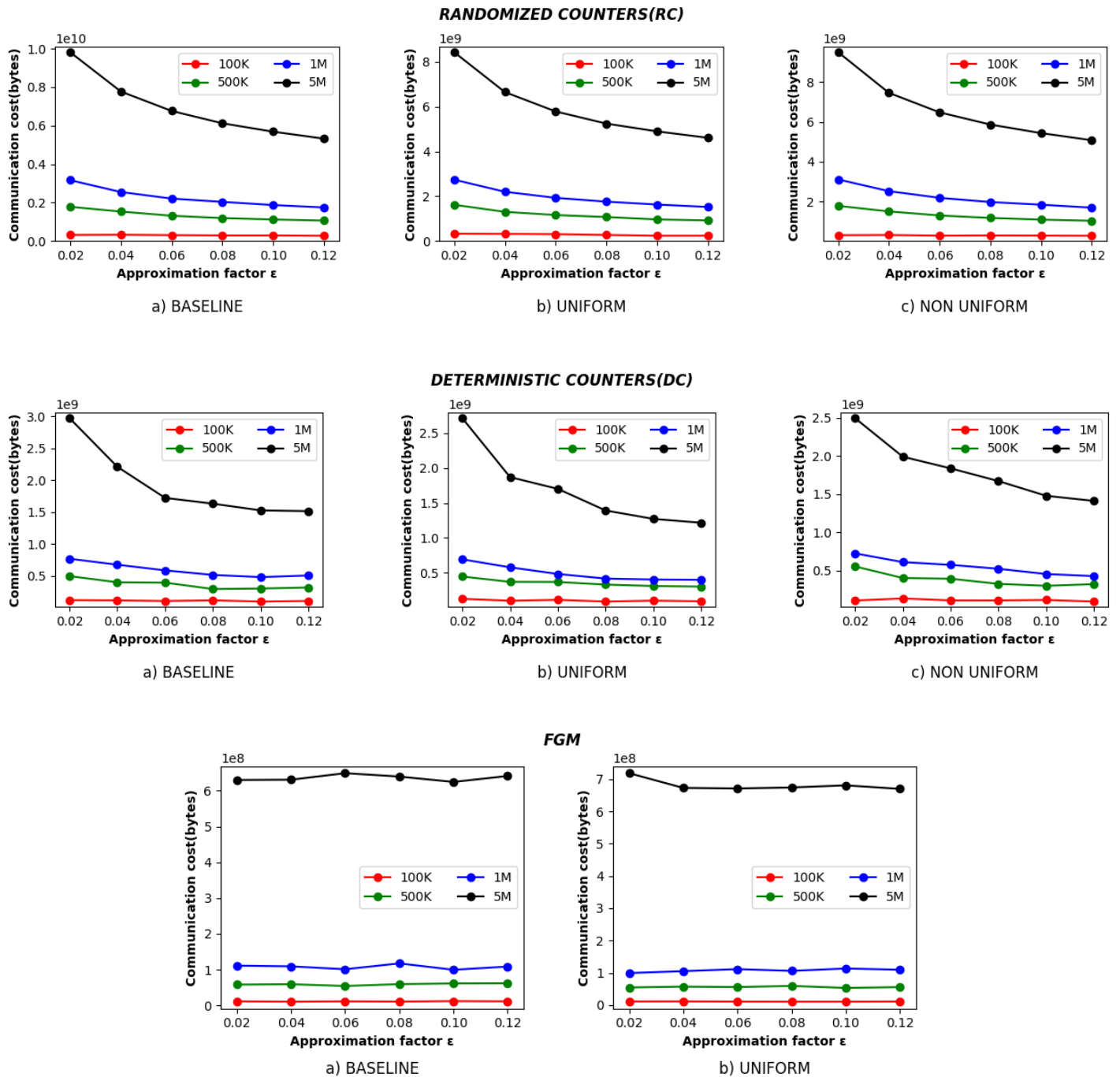


Figure 36: Communication cost related to the approximation factor  $\epsilon$  for both approaches for the HEPAR II dataset

### Communication cost related to the number of workers

We present the *communication cost* in relation to the number of sites, using the *BASELINE*, *UNIFORM*, and *NON\_UNIFORM* algorithms for each approach.

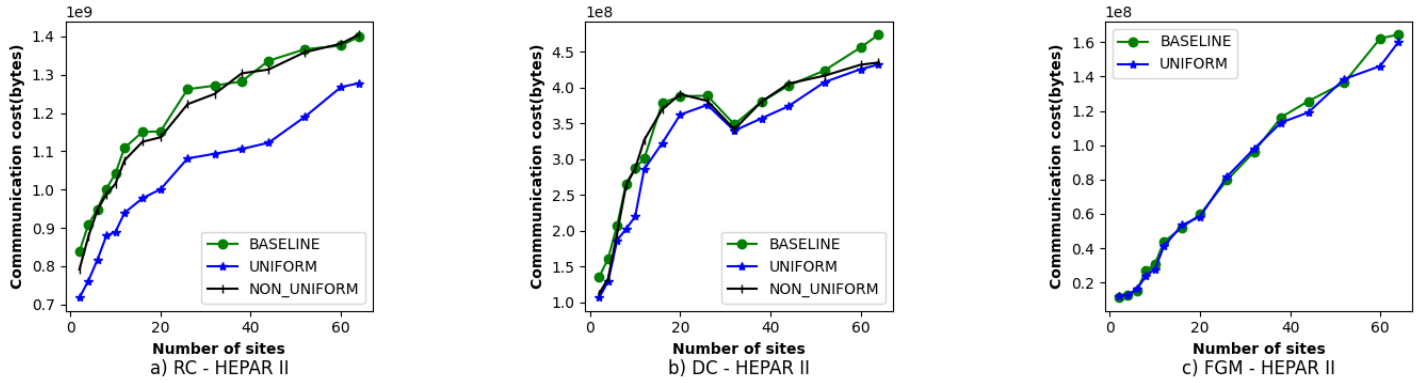


Figure 37: Communication cost related to the number of sites for both approaches for the HEPAR II dataset

### Scalability

#### Runtime related to the number of sites

We present the *runtime(sec)* in relation to the number of sites using the *BASELINE*, *UNIFORM*, and *NON\_UNIFORM* algorithms for each approach.

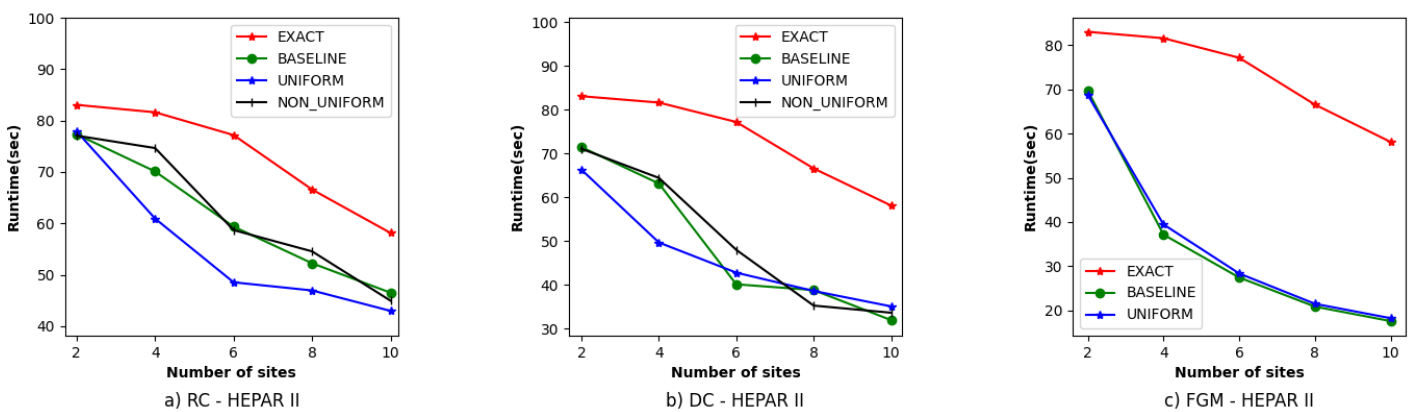
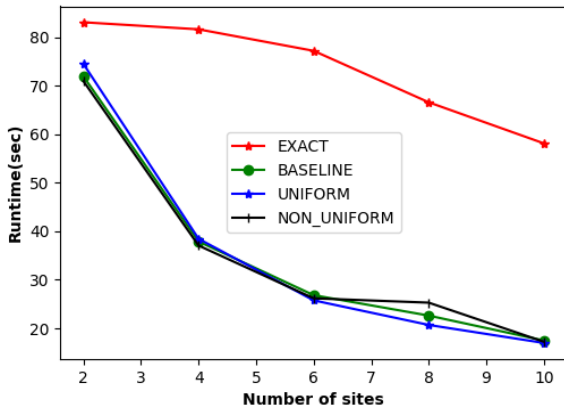
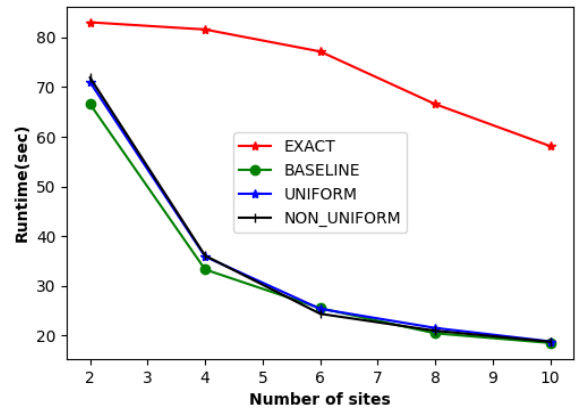


Figure 38: Runtime(sec) related to the number of sites for both approaches for the HEPAR II dataset



a) RC - HEPAR II, after initial state



b) DC - HEPAR II, after initial state

Figure 39: Runtime(sec) related to the number of sites for the first approach for the HEPAR II dataset, after the initial state

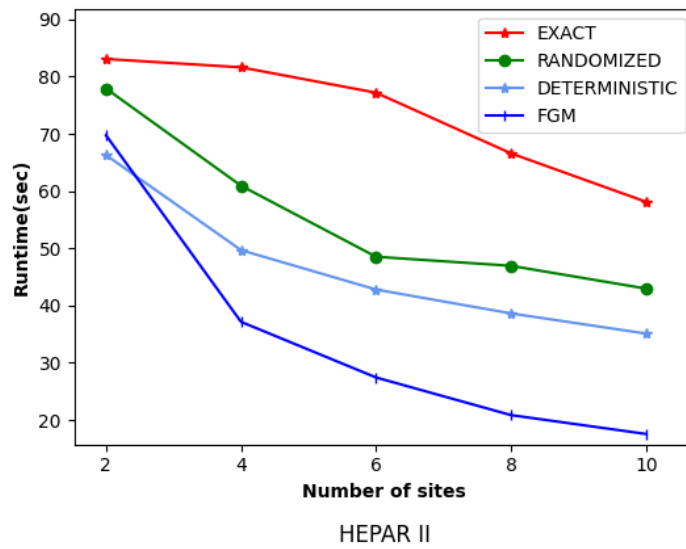


Figure 40: Runtime(sec) related to the number of sites for both approaches for the HEPAR II dataset - Best results

### Runtime related to the number of parallelism

We present the *runtime(sec)* in relation to the number of parallelism using the *BASELINE*, *UNIFORM*, and *NON\_UNIFORM* algorithms for each approach.

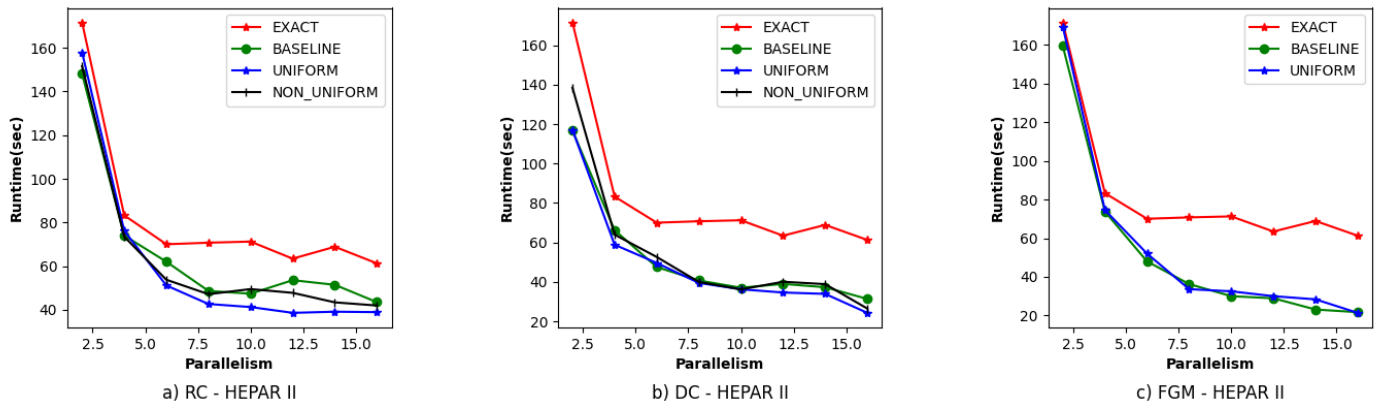


Figure 41: Runtime(sec) related to the number of sites for both approaches for the HEPAR II dataset

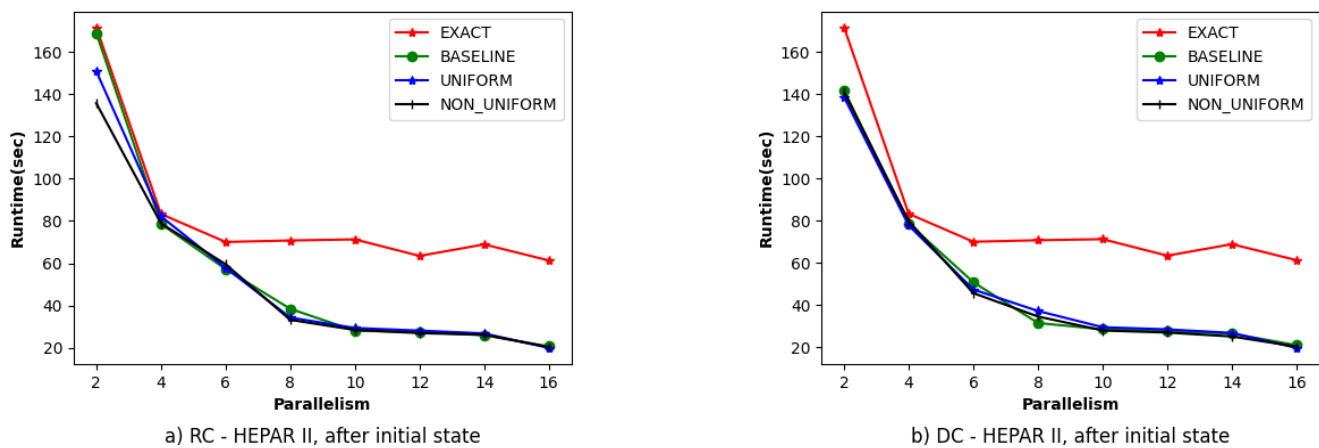


Figure 42: Runtime(sec) related to the number of sites for the first approach for the HEPAR II dataset, after the initial state

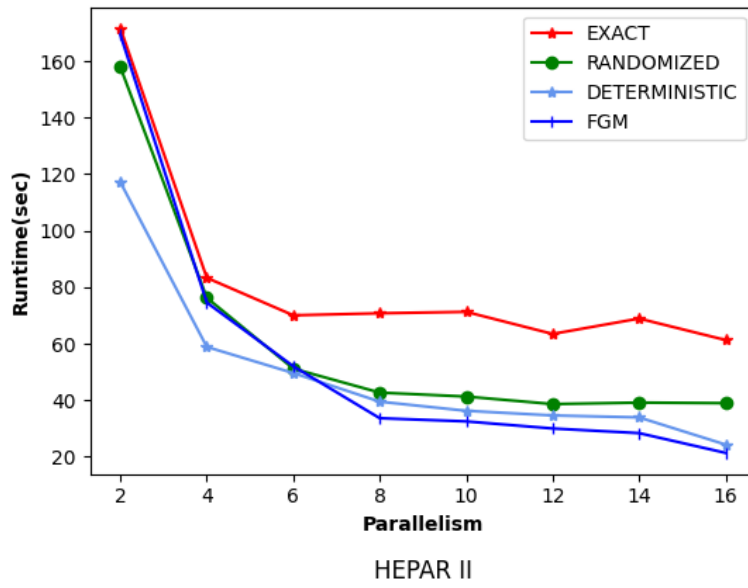


Figure 43: Runtime(sec) related to the number of sites for both approaches for the HEPAR II dataset - Best results

## Project Structure

In the following figure, the abstract structure of the project is presented. It depicts how the packages are organized, which will be discussed in detail below.

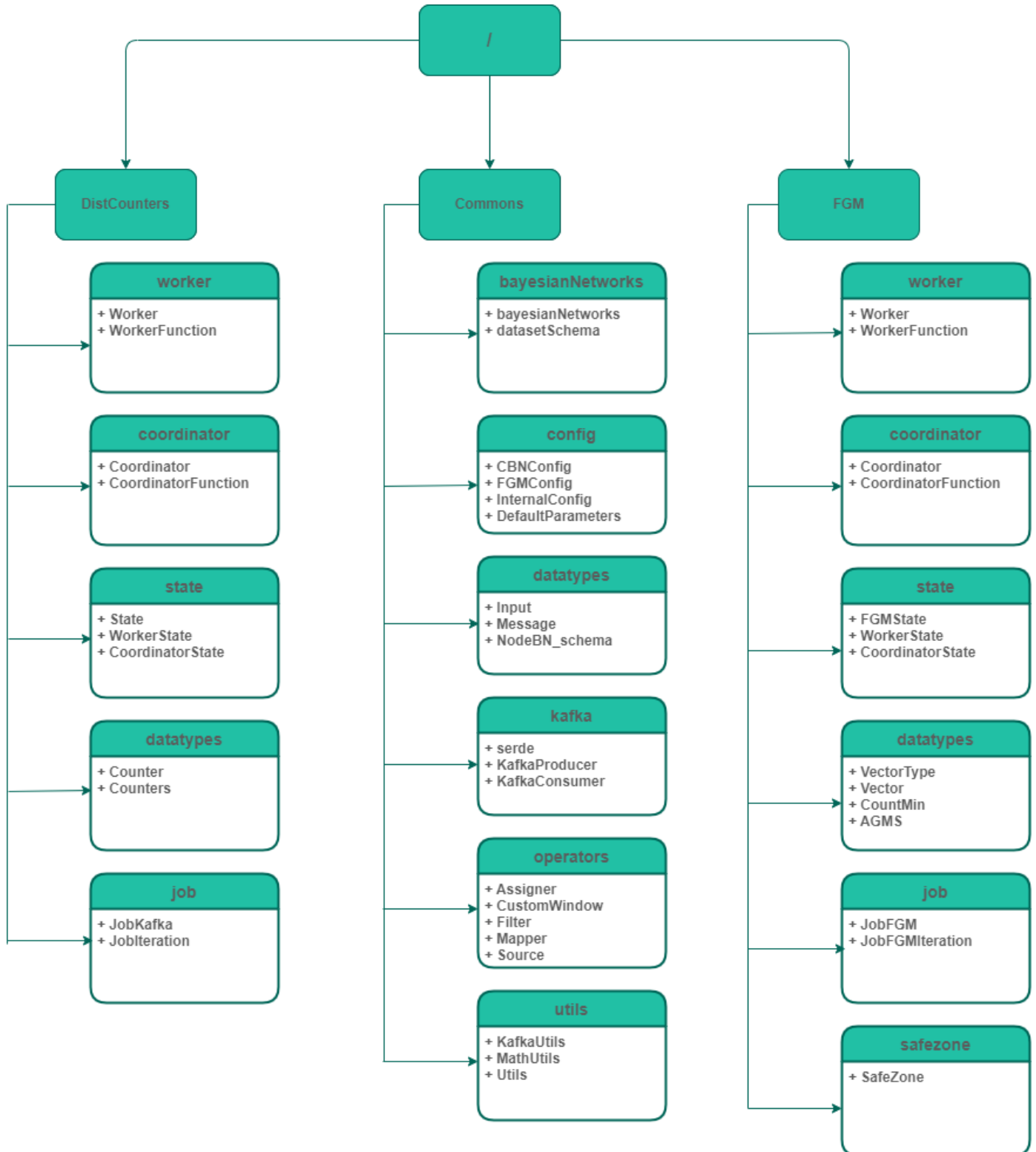


Figure 44: Project Structure



As depicted, the project structure is divided into three essential categories: the *DistCounters*, the *FGM*, and the *Commons*. *DistCounters* represents the Approximate Distributed Counters, *FGM* represents the components specific to the second approach, and *Commons* represents the shared components used by both approaches. We present a summary of the main components in each category.

### ***DistCounters-FGM***

- ***worker***: The package includes two components: the *Worker* and the *WorkerFunction*. The *Worker* component contains the rudimentary logic for each *Worker*, such as managing the *Input Source* and receiving control messages from the *Coordinator(Feedback Source)*. The *WorkerFunction* component provides the necessary functionality specific to each *Worker* and is defined based on the selected approach.
- ***coordinator***: The package consists of two components: the *Coordinator* and the *CoordinatorFunction*. The *Coordinator* component handles the basic logic used by the *Coordinator*, which includes managing messages from the *Workers* and handling the *probability queries (Query Source)*. The *CoordinatorFunction* component provides all the functionality for the *Coordinator* and is defined based on the selected approach.
- ***state***: The package consists of three components, but in this case, there is a hierarchical relation among them: the *State*, the *WorkerState*, and the *CoordinatorState*. The *State* component contains the shared elements used by both sides (*Worker-Coordinator*). It serves as a base component inherited by both *WorkerState* and *CoordinatorState*. The *State* component includes elements related to the setup of the entire pipeline of the system and will be discussed in detail in the following section. The *WorkerState* component includes elements from the *State* component and the additional state for each *Worker*. For example, the *approximate distributed counter* regarding the first approach while the *frequency vectors* consider the *FGM* protocol. The *CoordinatorState* follows a similar pattern, inheriting elements from the *State* component and maintaining a state specific to the *Coordinator* based on the selected approach (*approximate distributed counters* or *frequency vectors*).
- ***datatypes***: The package contains the basic *datatypes* utilized by both the *WorkerState* and the *CoordinatorState*. In the first approach, the fundamental element utilized by the sites is the *Counter*, which represents an *approximated distributed counter*. In the second approach, the key element is the *Vector*, which defines a *frequency vector*. As mentioned earlier, the *FGM* method constitutes an independent monitoring method. Thus, we also implement another essential element called *sketches*. Particularly, it includes two well-known and basic sketches: the *Fast-AGMS sketch* [33] and the *CountMin sketch* [34]. These sketches can be used for monitoring in conjunction with the appropriate safe function.
- ***safezone***: This package is specifically related to the *FGM* method and focuses on *safe functions*. It provides the necessary functionality to define and utilize the appropriate safe function based on the monitoring requirements. The package ensures compatibility with the specific datatype provided by the system.
- ***job***: This package encompasses the entire inferred pipeline by combining all the components. The first pipeline refers to the one that utilizes a *Kafka* topic as a feedback loop, while the second pipeline refers to the one that utilizes the *Iterative*

*Stream operator* provided by Apache Flink. These pipelines orchestrate the flow of data and processing within the system.

### **Commons**

- **bayesianNetworks:** This package serves as a repository for all the available *Bayesian Networks* that are integrated into the system, where each network is accompanied by the *schema* of the dataset. The *schema* defines the format of the *input feature vector* for a *Bayesian Network*. Therefore, for the insertion of a new *Bayesian Network* or *Naïve Bayes Classifier* into the system, it is necessary to define both the *bayesianNetwork* itself and its corresponding *datasetSchema*.
- **config:** This package consists of three individual components: the *InternalConfig*, the *CBNConfig*, and *FGMConfig*. The *InternalConfig* contains shared parameters by both approaches (*DistCounter* - *FGM*) required for the setup of the entire pipeline. The *CBNConfig* includes parameters specific to the *DistCounter*, while the *FGMConfig* includes the parameters related to the *FGM* protocol. These parameters are further discussed in detail in the *Project Setup* section.
- **datatypes:** This package comprises the basic shared *datatypes* utilized by both approaches. Specifically, it includes the format of the *Input Source* and the format of messages (*Message*) exchanged between the Coordinator and the Workers and vice versa.

## ***Project Setup***

This section represents the description of the parameters that need to be defined by the user to run an example of our system.

### *Distributed Counters Configuration*

**Parameter:** typeCounter

**Description:** This parameter defines the type of counter to be used as a basic component during the process. There are four available types of counters: RANDOMIZED, DETERMINISTIC, CONTINUOUS, and EXACT counter.

### *FGM Configuration*

**Parameter:** typeState

**Description:** This parameter defines the type of state to be used by both sides(workers-coordinator) during the process. There are three available types of state: VECTOR, Fast-AGMS, and COUNT\_MIN sketches.

**Parameter:** enableRebalancing

**Description:** This parameter enables/disables the rebalancing mechanism of the FGM protocol. Moreover, it works in conjunction with the value of lambda. The default value of lambda is 2.

**Parameter:** width, depth

**Description:** These parameters are only applicable if the type of state chosen is one of the available types of sketches. They determine the width and the depth of the sketch to be used as the state of the Workers and Coordinator.

### *Common Configuration*

**Parameter:** typeNetwork

**Description:** This parameter defines the type of network to be used during the process. There are two available types of networks: BAYESIAN, NAÏVE

**Parameter:** BNSchema

**Description:** This parameter defines the specific network to be used during the process. There are built-in network options available. Here are some of the available network options: SACHS, ALARM, HEPAR2, LINK, MUNIN, and EARTHQUAKE.

**Parameter:** datasetSchema

**Description:** This parameter defines the schema of the dataset to be used during the process. Similar to the network schema, there are built-in dataset schema options available. Here are some of the available schemas options: SACHS, ALARM, HEPAR2, LINK, MUNIN, and EARTHQUAKE.

**Parameter:** errorSetup

**Description:** This parameter defines the algorithm used to adjust the error between the available counters. There are three available options: the BASELINE, UNIFORM, and NON\_UNIFORM algorithms.

**Parameter:** workers

**Description:** This parameter defines the number of workers/sites (not including the Coordinator) to be used during the process.

**Parameter:** parallelism

**Description:** This parameter defines the parallelism i.e. the number of subtasks to be used by each pipeline operator during the process.

**Parameter:** eps

**Description:** This parameter specifies the epsilon value that defines the accuracy of the estimated joint probability distribution (user-defined error guarantees).

**Parameter:** delta

**Description:** This parameter specifies the delta value that defines the likelihood of the estimated joint probability distribution (user-defined error guarantees).

**Parameter:** inputTopic

**Description:** This parameter defines the Kafka topic to be used as the input for the Workers.

**Parameter:** feedbackTopic

**Description:** This parameter defines the Kafka topic to be used as a feedback loop between the Workers and the Coordinator.

Below are two complete examples demonstrating the application of the aforementioned parameters. The first example corresponds to the DistCounters method, while the second example corresponds to the FGM method.

In both examples, the dataset is the **HEPAR2** using **sourceHEPAR2** as inputTopic and **fdHEPAR2** as feedbackTopic. Furthermore, the number of workers is equivalent to **8** while the number of parallelism is equal to **4**. Finally, the accuracy is set to **0.1** and the likelihood of the estimated joint probability distribution is **90%**(error guarantees).

```
--inputTopic sourceHEPAR2 --feedbackTopic feedbackHEPAR2 --workers 8 --parallelism 4  
--eps 0.1 --delta 0.25 --errorSetup UNIFORM --typeCounter RANDOMIZED --queriesSize 1000  
--bn HEPAR2 --datasetSchema HEPAR2
```

The only change for the previous example to be applied to the second approach is the *typeCounter* to be replaced by the *typeState*, which is accompanied by the appropriate system type provided (for example, *VECTOR*).

## Bibliography

- [1] D. Koller and N. Friedman, Probabilistic graphical models: principles and techniques. Cambridge, MIT Press, 2009.
- [2] J. Joyce, "Bayes' Theorem," Stanf. Encycl. Philos., Jun. 2003, [Online]. Available: <https://plato.stanford.edu/archives/spr2019/entries/bayes-theorem/>
- [3] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, 1988.
- [4] O. Pourret, P. Naïm, and B. Marcot, Bayesian Networks: A Practical Guide to Applications. John Wiley & Sons, 2008.
- [5] O. J. Mengshoel, A. Darwiche, and S. Uckun, "Sensor Validation using Bayesian Networks," Feb. 2008.
- [6] S. Andreassen et al., "MUNIN: an expert EMG assistant," in Computer-Aided Electromyography and Expert Systems, J. E. Desmedt, Ed. Pergamon Press, 1989, pp. 255–277.
- [7] A. Onisko, M. J. Druzdzel, and H. Wasyluk, "A Bayesian Network Model for Diagnosis of Liver Disorders," Mar. 2003.
- [8] P. Xie, J. H. Li, X. Ou, P. Liu, and R. Levy, "Using Bayesian networks for cyber security analysis," in 2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN), Jun. 2010, pp. 211–220.
- [9] J. Dougherty, R. Kohavi, and M. Sahami, "Supervised and Unsupervised Discretization of Continuous Features," 1995, pp. 194–202.
- [10] P. Langley, W. Iba, and K. Thompson, "An analysis of Bayesian classifiers," in Proceedings of the tenth national conference on Artificial Intelligence, San Jose, California, Apr. 1992, pp. 223–228.
- [11] M. Ghazanfar and A. Prügel-Bennett, "An Improved Switching Hybrid Recommender System Using Naive Bayes Classifier and Collaborative Filtering," Apr. 2010.
- [12] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian Network Classifiers," Mach. Learn., vol. 29, no. 2, pp. 131–163, Nov. 1997.
- [13] T. J. Watson, "An empirical study of the naive Bayes classifier," 2001.
- [14] M. Henrion, "Propagating Uncertainty in Bayesian Networks by Probabilistic Logic Sampling," in Machine Intelligence and Pattern Recognition, vol. 5, J. F. Lemmer and L. N. Kanal, Eds. North-Holland, 1988, pp. 149–163.
- [15] R. Fung and K.-C. Chang, "Weighing and Integrating Evidence for Stochastic Simulation in Bayesian Networks," 1990, vol. 10, pp. 209–219.
- [16] G. Cormode, "The continuous distributed monitoring model," ACM SIGMOD Rec., vol. 42, no. 1, pp. 5–14, May 2013.
- [17] K. Yi and Q. Zhang, "Optimal Tracking of Distributed Heavy Hitters and Quantiles." arXiv, Nov. 30, 2008.
- [18] Y. Zhang, S. Tirthapura, and G. Cormode, "Learning Graphical Models from a Distributed Stream," in 2018 IEEE 34th International Conference on Data Engineering (ICDE), Paris, Apr. 2018, pp. 725–736.
- [19] Z. Huang, K. Yi, and Q. Zhang, "Randomized Algorithms for Tracking Distributed Count, Frequencies, and Ranks." arXiv, Dec.

02,2011. Available: <http://arxiv.org/abs/1108.3413>

[20] G. Cormode, S. Muthukrishnan, and K. Yi, "Algorithms for distributed functional monitoring," *ACM Trans. Algorithms*, vol. 7, no. 2, pp. 1–20, Mar. 2011.

[21] V. Samoladas and M. Garofalakis, "Functional Geometric Monitoring for Distributed Streams," 2019, p. 12.

[22] I. Sharfman, A. Schuster, and D. Keren, "A geometric approach to monitoring threshold functions over distributed data streams," *ACM Trans. Database Syst.*, vol. 32, no. 4, p. 23, Nov. 2007.

[23] "Apache Flink." <https://flink.apache.org/> (accessed Feb. 11, 2023).

[24] "Apache Spark." <https://spark.apache.org/> (accessed Feb. 10, 2023).

[25] A. Lazerson, I. Sharfman, D. Keren, A. Schuster, M. Garofalakis, and V. Samoladas, "Monitoring distributed streams using convex decompositions," *Proc. VLDB Endow.*, vol. 8, no. 5, pp. 545–556, Jan. 2015.

[26] "Apache Kafka." <https://kafka.apache.org/> (accessed Feb. 11, 2023).

[27] "Nikolaos Tzimos." <https://github.com/NikolasTz> (accessed Feb. 11, 2023).

[28] "Software Technology and Network Applications Laboratory | SoftNet." <https://www.softnet.tuc.gr/en/> (accessed Feb. 11, 2023).

[29] M. Scutari, "bnlearn – Bayesian Network Repository." <https://www.bnlearn.com/bnrepository/> (accessed Feb. 11, 2023).

[30] I. A. Beinlich, H. J. Suermondt, R. M. Chavez, and G. F. Cooper, "The ALARM Monitoring System: A Case Study with two Probabilistic Inference Techniques for Belief Networks," in *AIME 89*, Berlin, Heidelberg, 1989, pp. 247–256.

[31] C. S. Jensen and A. Kong, "Blocking Gibbs Sampling for Linkage Analysis in Large Pedigrees with Many Loops," *Am. J. Hum. Genet.*, vol. 65, no. 3, pp. 885–901, Sep. 1999.

[32] B. Kveton, H. Bui, M. Ghavamzadeh, G. Theodorou, S. Muthukrishnan, and S. Sun, "Graphical Model Sketch." *arXiv*, Jul. 18, 2016.

[33] G. Cormode and M. Garofalakis, "Sketching Streams Through the Net: Distributed Approximate Query Tracking," presented at the Very Large Data Bases Conference, Aug. 2005.

[34] G. Cormode and S. Muthukrishnan, "An Improved Data Stream Summary: The Count-Min Sketch and Its Applications," Berlin, Heidelberg, 2004, vol. 2976, pp. 29–38.