אוניברסיטת בן-גוריון בנגב | המחלקה להנדסת תעשייה וניהול

Ben-Gurion University of the Negev

Faculty of Engineering Sciences

Department of Industrial Engineering and Management

Ensemble-Based Concept Drift Detection Using Bayesian Networks

THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE M.Sc. DEGREE

By

Yotam Baron

Supervision By: Prof. Boaz Lerner

September 2022

אוניברסיטת בן-גוריון בנגב | המחלקה להנדסת תעשייה וניהול

Ben-Gurion University of the Negev

Faculty of Engineering Sciences

Department of Industrial Engineering and Management

Ensemble-Based Concept Drift Detection Using Bayesian Networks

THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE M.Sc. DEGREE

By

Yotam Baron

Supervision By: Prof. Boaz Lerner

Author: ..... Date: 26.12.22

Supervisor: ..... Date: 26.12.22

Chairman of Graduate Studies Committee: ..... Date: 26.12.22

September 2022

Abstract

The world generates an unfathomable amount of data every minute of every day, and it continues to multiply at a staggering rate. Companies in every industry are quickly shifting from batch processing to real-time data streams to keep up with modern business requirements. Data streams are prone to withhold a phenomenon known as concept drift (CD) where the incoming data's characteristics change in some form over time.

In general, a change can arise for many reasons, e.g., price growth due to inflation, the needs of an aging population, the effect of global warming, or equipment wear. Dealing with CD and maintaining an efficient, accurate, and up to date model is a crucial task in our dynamic world.

Addressing this task, an algorithm based on a Bayesian network (BN) that efficiently deals with CD was introduced. Our algorithm – known as concept drift detection and re-learning (CDDRL) – besides detecting the drift can also be used for knowledge representation via the analysis of the BNs learned by it through time. The CDDRL is a two-stage algorithm. In the first stage, it sequentially monitors statistical distances between BN's conditional probabilities tables (CPT) using statistical process control (SPC) charts to identify parametric and structural changes that represent the detected drift and are needed in the BN to follow the drifted distribution/concept. In the second stage, it re-learns the BN locally via a search and score process looking at a neighborhood of graphs derived from the graph used before the drift with structural changes only in nodes that have been detected as drifted from the previous stage.

In this work, several developments to the CDDRL algorithm were introduced. First, a novel method based on conditional independence (CI) tests is proposed as a replacement for the second stage of the CDDRL, i.e., the search and score process, to reduce algorithm complexity. Second, to increase its prediction power, two ensemble versions of the CDDRL were proposed. The first version makes predictions at each time step by using its previously learned BNs. The second version makes predictions by aggregating predictions of several modifications of the CDDRL, each holding a single change to a specific process in the original CDDRL, e.g., in the way the BN's CPTs are learned, in the way drift thresholds for the SPC are obtained, and by the addition of a tabu list to the search process.

The new proposed methods were implemented and tested together with the original CDDRL. Three synthetic datasets – consisting of different types and magnitude of CD – were used for evaluation, together with seven real-world datasets, three of which were water and fertilizer stress in banana plants experiments conducted with us. Using the Friedman-Nemenyi test, our results show that the two ensemble versions as well as the original CDDRL are ranked at the very top of the overall performance of all datasets and are significantly better than at least five out of the eleven competing algorithms tested. The ensemble versions improved the CDDRL's predictive power though showed a longer running time, whereas the CI version – tested in other experiments – was inferior to the original CDDRL but presented a shorter running time.

Keywords: Bayesian networks, concept drift, search and score, conditional independence tests, ensemble methods, plants stress, Friedman-Nemenyi test

Dedication and Acknowledgements

First and foremost, I would like to thank and express my appreciation to my supervisor, Prof. Boaz Lerner, for his continuous support during my master's study and research. Your uncompromising standards and care have been of great help to me during this time. I am extremely grateful for the time and patience you have invested in me and for the opportunity to be involved in many interesting and important machine learning projects and collaborations under your guidance.

Secondly, I would like to express my appreciation to Liran Nahum, Moriya Cohen and Shon Mendelson, my predecessors, whose great ideas and hard work have established the foundations of this thesis.

Additionally, I would also like to thank all my research colleagues from the lab, I had a great time alongside you and I thank you for your useful advice and support. I would like to particularly thank Shoham Shabat and Ori Ben Yehuda for always being there to help, listen and have fun - you have become much more than friends.

Finally, I would like to sincerely thank my family and life partner Ariel, who always encouraged, consulted, and understood me during this complicated period – thank you for your strength and love.

Author's Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmers and that it has not been submitted for any other academic award. Except where indicated by specific references in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: .....yotam..... DATE: .....26.12.22.....

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

BN	**B**ayesian **n**etwork
CB	**C**onstraint-**b**ased
CD	**C**oncept **d**rift
CI	**C**onditional **i**ndependence
CL	**C**enter **l**ine
CPD	**C**onditional **p**robability **d**istribution
CPT	**C**onditional **p**robability **t**able
DAG	**D**irected **a**cyclic **g**raph
LL	**L**og-**L**ikelihood
S&S	**S**earch and **s**core
SHD	**S**tructural **H**amming **d**istance
SPC	**S**tatistical **p**rocess **c**ontrol
UCL	**U**pper **c**ontrol **l**imit
STM	**S**hort **t**erm **m**emory
LTM	**L**ong **t**erm **m**emory
EWMA	**E**xponential **w**eighted **m**oving **a**verage
CMI	**C**onditional **m**utual **i**nformation
MDL	**M**inimum **d**escription **l**ength
CF	**C**lassification **f**unction
RT	**R**ecovery **t**ime
ME	**M**issing **e**dge
EE	**E**xtra **e**dge
WD	**W**rong **d**irection

1 Introduction

Many machine learning algorithms assume that the data it is using to generate a model come from a stable process, meaning that the inputs' distribution as well as the target value's distribution stay fixed through time. In the real world this conception is not always correct as concepts are often not stable but change over time. This phenomenon of change in the distribution underlying the data is known as concept drift (CD) (Lu et al., 2014). A change can arise for many reasons, e.g., price growth due to inflation and weather prediction rules that may vary radically with the season (Tsymbal, 2004).

Dealing with data streams usually includes dealing with CD as data streams demonstrate continuous activity over long periods of time, and in real-life applications it is natural that the underlying process can change over time, resulting in a CD (Mendelson, 2020). To deal with online data streaming and the problem of CD, special approaches, different from commonly used techniques, which treat arriving instances as equally important contributors to the final concept are needed (Tsymbal, 2004).

A recent method called CDDRL for dealing with concept drift detection and knowledge representation of the change's development was introduced by (Mendelson, 2020). The method uses Bayesian Networks (BN) and a parameter distance comparison to alert when a change has occurred, after a change was detected, the method learns the updated network's structure in a smart and efficient manner. Since the method uses BNs, it can alert not only on when a change occurs, but also on where in the domain it appeared. This characteristic is very useful in understanding the development process of the CD. (Mendelson, 2020) shows that the method was found robust to different types of change, e.g., *gradual* and *sudden*, as well as when dealing with different kind of data distributions, e.g., skewed and non-skewed.

Learning a BN structure from data is a NP-hard problem, since the number of possible graphs grows exponentially with the number of nodes (Chickering, 2003). Therefore, heuristics have been developed to cope with this task. State-of-the-art BN structure learning algorithms can be divided into three groups: 1) Search and score (S&S) algorithms (Cooper & Herskovits, 1992; Heckerman et al., 1995) that perform a search procedure in the possible graphs space and choses the best graph by applying a relevant scoring measure on the possible graphs. 2) Constraint-based algorithms (Spirtes et al.,

2000; Yehezkel & Lerner, 2009) that use statistical conditional independence (CI) tests to discover causal relations between the variables (nodes) in the domain – represented by directed edges between them indicating a cause-and-effect relation. 3) Hybrid methods (Tsamardinos et al., 2006) that apply some procedure that combines the first two methods. The CDDRL originally uses the S&S approach.

The first contribution of this work is a proposed method to efficiently re-learn the BN structure. The novel approach is constraint-based but can also be looked at as a hybrid approach since its final step involves an S&S process. The new method utilizes the information from the first stage of the CDDRL about the changed nodes to perform only limited and specific independence tests between variables in the domain to discover the directed edges between them. CI tests when correctly controlled are less expensive thus making the new method a promising option to decrease run time. We test and evaluate the new method in both structure learning experiments and classification under CD experiments.

The second contribution of this work is the introduction of several developments to the CDDRL algorithm. To improve the CDDRL's predictive power, we propose two ensemble versions of the CDDRL. The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability/robustness over a single estimator by reduction of the base estimators' variance (Louppe & Geurts, 2012).

The first proposed CDDRL-based ensemble version utilizes a predefined number of previously learned BNs to make a prediction at every time step. The BNs in the ensemble provide their individual predictions that are then aggregated into a single prediction using weights calculated based on each BN's past performance. We call this version CDDRL-*Ensemble BNs.*

The second proposed version – named CDDRL-*Ensemble Methods* – consists of four slightly altered methods of the original CDDRL as its base estimators. The first method alters the way in which the CDDRL computes the threshold (upper control limit) to detect a drift in a node in a way that makes it more suitable for our dynamic world – we call this method *Dynamic UCLs*. The second alters the way in which we learn the BN's CPTs at each time step aiming to reduce overfitting of the data – we call this method *Params*. The third adds a tabu list (Glover & Laguna, 1998) to the S&S process helping it escape from local

minima/maxima – named *Tabu Search*. Finally, the fourth method expands the search space for the correct BN structure at each time step, elevating the chance to find a better BN structure but on the other hand increasing run time – we call this method *Robust*. Prediction aggregation of the base estimators is done in the same manner as in *CDDRL-Ensemble BNs*.

We provide experimental analysis and evaluation of the four methods that form the CDDRL-*Ensemble Methods* and then we test and evaluate both CDDRL ensemble versions alongside the original version with three synthetic and seven real-world CD related datasets.

The third contribution of this work is a modified Friedman-Nemenyi test (M. Friedman, 1937; Nemenyi, 1963) to check statistical significance between several algorithms according to several evaluation metrics simultaneously (instead of a single metric as conventionally used) to get an overall ranking of the tested algorithms. To the best of our knowledge, no other work has combined more than one evaluation metric whilst using the Friedman-Nemenyi test. We use the modified test on the datasets mentioned above to obtain an overall rank and statistical significance of each CDDRL version compared to 11 other competing algorithms.

In addition, this work contributes by evaluating the CDDRL's unique ability to act as a knowledge representation algorithm. We demonstrate and discuss this ability using an experiment we conducted in the agriculture field involving water stress induced to greenhouse-grown banana plants.

Computational complexity analysis of the CDDRL, which was never done before, is also provided and discussed.

Finally, an article presenting the main findings of this thesis is under revision for the well-respected *IEEE TNNLS* journal under its special issue on *Stream Learning*. The article as originally submitted can be found in Appendix 9.4.

2 Background

2.1 Concept drift and data streams

A data stream is a data set in which the objects have timestamps, which, depending on the granularity of the stamps, induces either a total or a partial order between observations (Webb et al., 2017). In recent years, advances in hardware technology have facilitated the ability to collect data continuously. Simple transactions of everyday life such as using a credit card, a phone or browsing the web led to automated data storage. The volume of such data is so large that it may not be possible to store all of it at once (Aggarwal, 2007). Since the data is infinite and cannot be stored for a long time, algorithms dealing with data streams are challenged to be designed in a way that they pass each piece of data only once. Another common characteristic of streaming data is that it may change over time. This phenomena of change in the data's distribution over time is commonly known as concept drift. The term concept drift can be sub-categorized into two types: a change in the input's data distribution, known as *virtual* concept drift, and a change in the target variable given the input, known as *real* concept drift (Gama et al., 2014).

2.1.1 Real and Virtual Concept Drift

Lu (Lu et al., 2014) explains the difference between the two terms as follows: If we denote the feature vector as x and the class label as y, then the data stream will be an infinite sequence of (x, y). If concept drifts are present, it means the distribution of $p(x, y)$ is changing between the current data chunk and the yet to come data. If we decompose $p(x, y)$ into the following two parts as $p(x, y) = p(x) * p(y|x)$, we could say that there are two sources of CD: one is $p(x)$, which evolves with time t, and can also be written as $p(x|t)$, and the other is $p(y|x)$, the conditional probability of feature x. The former source is noted as *virtual CD* and the latter as *real CD*.

(Pesaranghader et al., 2017) further evaluates the impact of the two types of CD on the decision boundaries of a classifier as follows: A *real CD* refers to the changes in $p(y|X)$ which affects the decision boundaries or the target concept. On the other hand, *virtual CD* is the result of a change in $p(X)$, and subsequently in $p(X|y)$, but not in $p(y|X)$. That is, a *virtual CD* is a change in the distribution of the incoming data which implies that the

decision boundaries remain unaffected. Pesaranghader et al. (2017) also note that when dealing with a prediction problem, the classifier only needs to be adapted when a *real CD* is detected, since only a *real CD* changes the classifier decision boundaries. The figure below illustrates this concept. However, if the model has to represent the real underlying data distribution, it also has to be adapted in the case of a *virtual CD*.

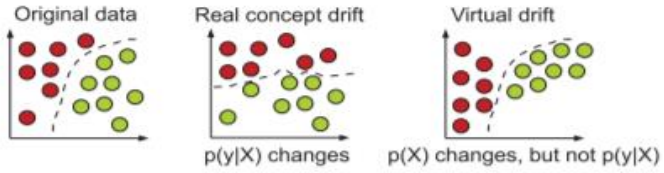


Figure 1: Types of drifts: circles represent instances; different colors represent different classes

Virtual CD and *real CD* can occur separately as well as at the same time. An example of a *virtual CD* is the case of spam categorization, while our understanding of an unwanted message may remain the same over a relatively long period of time, the relative frequency of different types of spam may change drastically with time (Tsymbal, 2004). A typical example of the *real CD* is a change in users' interests when following an online news stream. While the distribution of the incoming news documents often remains the same, the conditional distribution of the interesting (and thus not interesting) news documents for that user changes (Gama et al., 2014).

Virtual and *real CDs* reflect on the source of the drift, i.e., a change in the input distribution or a change in the target function. However, CD can be further categorized into more groups, which will now be discussed.

2.1.2 Concept Drift – Change Types

Change types refer to the configuration patterns of the data sources over time. The structural types of change are usually defined based on those configurations (Žliobaitė, 2010). The change types are modified in different ways in the literature, the most common types are *sudden* (abrupt) and *gradual* change as (Tsymbal, 2004) defines them.

A *sudden* drift refers to the scenario where a current concept is suddenly replaced by a different concept, resulting in a sudden change in the data's distribution. For example, someone graduating from college might suddenly have completely different monetary concerns (Tsymbal, 2004).

A *gradual* drift refers to a slow transition from the current concept to another concept. In this type of change, there are two active concepts and the probability of sampling from the first concept is slowly decreasing as the probability of sampling from the second concept is increasing accordingly (Žliobaitė, 2010).

There are some extensions in the literature to the *gradual CD*. Stanley (Stanley, 2003) divides gradual drift further into moderate and slow drifts, depending on the rate of the changes. He also gives the example of a slowly wearing piece of factory equipment as a cause of a gradual change in the quality of output parts. Another change type often mixed with gradual change, is known as incremental change. An *incremental* drift is such that a sequence of data distributions appear during the transition (Pesaranghader et al., 2017). However, the difference between the different data distributions is very small, thus the drift is noticed only when looking at a longer time period, (Baena-García et al., 2006).

The last common change type mentioned in the literature is known as a *recurring* drift. That is when previously active concept reappears after some time. It differs from common seasonality notion in a way that it is not certainly periodic, it is not clear when the source might reappear (Žliobaitė, 2010). An example of such drift is the demand for ice cream, since the demand goes up every summer and down again after it, but the exact time of the increase in demand is unknown and changes every year.

Finally, a less common approach is to categorize change types into mutually exclusive categories (Minku et al., 2010) based on number of reoccurrences, severity, speed and predictability. In principle the proposed categorization tries to quantify the main aspects of the learner design process into change categorization. However, (Žliobaitė, 2010) argues that the categories cannot be mutually exclusive, because the change frequency count, speed, severity is relative to the length of the subsequence, at which one is looking. In the figure below the most common change types are visualized.

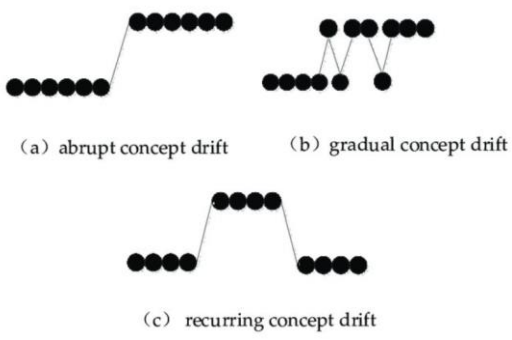


Figure 2: Patterns of change over time

2.2 Concept drift Models

In general, approaches to cope with CD can be classified into two categories: the first, approaches that adapt a learner at regular intervals without considering whether changes have really occurred, named blind or passive learners; and the second, approaches that first utilize a drift detector to detect concept changes, and next, the learner is adapted to these changes, these learners are known as informed or active learners. Furthermore, there are general drift detection approaches that can be placed on any learner and others that cooperate only with a specific machine learning model. Finally, there are approaches that keep one updated learner and others that are ensemble-based. All of these will be reviewed.

2.2.1 Drift Detection Methods

(KIFER et al., 2004) offer a scheme for a general change detection problem in a data stream. They test the difference in distributions between two samples. The first sample includes the first X observations in the data stream, and it remains constant. The second sample includes the last Y observations and is updated with every new observation that arrives. They aim to offer a generalized scheme that makes no assumptions on the form of the underlying distribution, and thus they use a non-parametric test to measure the difference between those two samples and compare it to some threshold α. Another non-parametric test is known as Kolmogorov-Smirnov Windowing [KSWIN; (Raab et al., 2020)], which is a concept drift detection method based on the Kolmogorov-Smirnov (KS) statistical test. KS-test is a statistical test with no assumption of underlying data distribution. KSWIN can monitor data or performance distributions. As these may be excellent schemes for general problems that require no assumptions, they may suffer from some disadvantages if used with a complex parametric model since it will fail to capture the internal relations in the network. It is easy to think of an example in which some conditional distributions of a variable are changed in some way although its marginal distribution remains the same. In this case, the non-parametric test is prone to fail in detecting the change.

Another general approach for drift detection was recently introduced in (Khamassi et al., 2015), they name their method EDIST2. This method is designed to fit every data

streaming classifying learner. The method uses the learner's predictions and monitors the amount of prediction errors between instances in a global adaptive window – the window size increases if no drift was found and decreases otherwise. An increase in the number of errors in the current window indicates of a possible concept drift. This approach also guarantees a preset false alarm rate using statistical testing to flag that a change has occurred. The advantage of this approach is that it is general and can be fitted to all stationary assuming classifiers but suffers from some disadvantages since it must constantly have the true labels of the observations which is not always possible, and the increase in the window size might be challenging as the algorithm saves in memory all the instances in it and therefore can lead to extreme memory consumption, especially when dealing with data streaming tasks.

Besides the EDIST2, there are other common methods that use an adaptive window and many of them use error monitoring to detect a possible drift. Some examples of these methods are: Drift Detection Method (DDM) introduced in (Gama et al., 2004), it uses a single window and continuously adjusts it by monitoring the misclassifying average μ_i and σ_i for each instance i. Then, it compares them to μ_{min} and σ_{min} reached during stable period in order to detect a drift (Khamassi et al., 2015). DDM was later expended by (Baena-García et al., 2006) to the Early Drift Detection Method (EDDM), this approach assumes that if the distribution of the instances is stationary, the learning model will improve its prediction and the time between the occurrences of the errors will increase as the number of instances increases. Thus, a significant decrease in time between errors implies a drift. EDDM calculates the average distance between two errors μ_i and its σ_i and compares them to μ_{max} and σ_{max} reached when the distribution of distances between errors is maximum. (Khamassi et al., 2015) claim that both the DDM and the EDDM are sensitive to noise and therefore to high false alarm rates due to the correlation of the adaptive window size to the min / max μ and σ.

Another similar and common method is the adaptive windowing (ADWIN) introduced by (Bifet & Gavaldà, 2007), the ADWIN splits the window into two sub-windows using a formula to set the cut point and then compares the average error in each of them to detect change. (Khamassi et al., 2015) found that ADWIN is good for *sudden* drifts, but its performance deteriorates when dealing with *gradual* drifts. Also, since the ADWIN alerts on a drift whenever there is any kind of significant difference in the average

error between the two sub-windows, it might wrongly alert on a drift in cases where the average error goes down, as this can be explained by the improvement of the learner due to additional data received and not because of a drift.

2.2.2 Active Learners

As mentioned, active approaches incorporate a drift-detection method accompanied with a base estimator to formulate a model that can deal with CD and streaming data. KNN-ADWIN is such an example, where ADWIN first detects and removes drifted examples, and then a classifier is trained, e.g., the K-nearest-neighbor (KNN). The Very Fast Decision Rules [VFDR; (Kosina & Gama, 2015)] is a rule-based classifier, each rule is a conjunction of conditions based on attribute values and the structure for keeping sufficient statistics. The sufficient statistics will determine the class predicted by the rule. Each rule is accompanied with a drift-detection method (in this paper we used EDDM), once a change is detected, the matching rule is discarded and a new one is learned. The Online Boosting Classifier [OBC; (Wang & Pineau, 2016)] is the online version of AdaBoost, every arriving observation is sampled k times using a Binomial distribution. Since data is infinite, the Binomial distribution tends to act like a Poisson distribution, λ of the Poisson distribution is computed by tracking the total weights of the correctly and misclassified examples. OBC is accompanied with the ADWIN change detector. The Adaptive Random Forest [ARF; (Heitor M. Gomes et al., 2017)] uses Hoeffding Trees as base estimators to form a random forest classifier. The ARF uses a drift-detector (in this paper we used KSWIN) per base tree, which cause selective resets in response to drifts. Neural Network Uncertainty [NN-UN; (Baier et al., 2021)] is a recent active method that uses Monte-Carlo Dropouts during several inferences of each incoming observation to calculate the model's uncertainty (entropy is used as a measure). These uncertainties are the input to an ADWIN change detector that alerts of drifts and triggers retraining of the model. The authors of the NN-UN state that the method will not identify *real CD* if it is not accompanied with a *virtual CD*.

2.2.3 Passive Learners

Passive learners do not incorporate a drift-detector in their process, but rather update their model every predefined number of observations or time steps. The Robust

Soft Learning Vector Quantization [RSLVQ; (Heusinger et al., 2022)] is a passive prototype-based algorithm, that uses a gradient descent method to optimize the objective function. Lately, the Adadelta momentum-based SGD was introduced to the RSLVQ algorithm as a method to deal with streaming data.

2.2.4 Ensemble Methods

Ensemble methods keep more than one learner in memory, they train new models through time: either in a static manner every predefined number of time steps / observations or in a dynamic manner according to some change in the data. The learned models are used as the ensemble's base estimators and are weighted in some form for prediction purposes. The ensemble methods mainly differ in the way they weigh their base estimators and how estimators are added or removed from the ensemble.

2.2.4.1 Active Ensemble Learners

The active ensemble methods include the Self Adjusting Memory coupled with the KNN classifier [SAM-KNN; (Losing et al., 2017)] that builds an ensemble with models targeting current or former concepts and is built using two memories: STM for the current concept, and the LTM to retain information about past concepts. A cleaning process is in charge of controlling the size of the STM while keeping the information in the LTM consistent with the STM. The Streaming random patches [SRP; (Heitor Murilo Gomes et al., 2019)] is an ensemble Hoeffding Tree based algorithm that combines random sub-spaces and bagging to classify under CD. Trees are replaced when they are detected as drifted using an ADWIN change detector.

2.2.4.2 Passive Ensemble Learners

Passive ensemble methods include the accuracy-weighted ensemble [AWE; (Borchani et al., 2016)] that weighs classifier decisions based on their expected classification accuracy on the test data, the dynamic weighted majority [DWM; (Kolter & Maloof, 2007)] uses such weights to remove models from the ensemble and to add new models based on the global performance of the ensemble, and [LNSE; (Elwell & Polikar, 2011)] dynamically weighs the classifiers based on their accuracy in the current environment compared with past environments.

Although all algorithms have good results, especially for *real CD*, these algorithms cannot detect the source of a change. For example, in a marketing problem, searching for CD changes in customer preferences, the ability to identify what caused changes in preferences, which can be used in future marketing campaigns, is missing (Mendelson, 2020). This knowledge representation ability is only present in our BN-based algorithm.

2.3 Bayesian Networks

A BN is a statistical and graphical model that efficiently encodes dependency relations for a large set of variables and allows the identification and understanding of the causality in the problem domain. The structure of a BN is composed of nodes representing the domain variables and directed edges connecting these nodes, representing probabilistic and causal relations between the corresponding variables (Pearl, 2000). Learning the BN structure is NP-hard (Chickering, 2003) which requires substantial computing resources or sub-optimal procedures, but once the structure is learned, the network parameters may easily be learned from the model and data (Heckerman, 2020).

The structure is a directed acyclic graph (DAG) consisting of a set of nodes $\boldsymbol{V}$ corresponding to the set variables $\boldsymbol{X}$, and a set of directed edges $\boldsymbol{E}$ connecting the variables, representing a direct causal influence from one variable to another (Darwiche, 2009). We denote the structure by $G = (\boldsymbol{V}, \boldsymbol{E})$. The set $\boldsymbol{E}$ describes the conditional dependencies between the variables in $\boldsymbol{V}$. So, an edge $e_{ij} \in \boldsymbol{E}$ from X_i to X_j indicates that those variables are dependent. Moreover, we say that X_i is parent of X_j and X_j is a child of X_i. All parents of X_j denotes as $\boldsymbol{PA}_j$.

The set of parameters $\boldsymbol{\theta}$ includes the conditional probability table (CPT) for each node in the network. All the CPTs together define the joint probability distribution of $\boldsymbol{X}$. for any combination of variables $X_1, X_2, \dots X_n \in \boldsymbol{X}$ the joint probability distribution is calculated by:

$$P(X_1, X_2, \dots X_n) = \prod_{i=1}^{n} P(X_i | \boldsymbol{PA}_i)$$

Eq. 1

2.3.1 D-separation (Cheng et al., 2002; Pearl, 1988)

For any two nodes $X, Y \in \boldsymbol{V}$, an adjacency path $p = \{e_{xI}, e_{IQ}, \dots, e_{wy}\}$ between X and Y is a sequence of edges that, if viewed as undirected edges, connect X and Y. For a DAG $\mathcal{G} = (\boldsymbol{V}, \boldsymbol{E})$, for any two nodes $X, Y \in \boldsymbol{V}$ where $X \neq Y$, and an adjacency path $\boldsymbol{S}$ that connects them, we say that X and Y are d-separated in $\mathcal{G}$ given $\boldsymbol{S}$ ($X \perp\!\!\!\perp Y | \boldsymbol{S}$) if and only if for all adjacency paths between X and Y either:

(i) The connection is serial (i.e., X→ $\boldsymbol{S}$ →Y) or diverging (i.e., X← $\boldsymbol{S}$ →Y) and the state of $\boldsymbol{S}$ is known (non-collider with a received evidence), or

(ii) The connection is converging (i.e., X→ $\boldsymbol{S}$ ←Y) and neither $\boldsymbol{S}$ nor have any of its descendants received evidence (collider without received evidence).

2.3.2 Structure Learning of a Bayesian Network

The number of possible BNs structures grows exponentially with the number of nodes in the BN. For instance, a BN containing 4, 5 or 6 nodes has 543, $29{,}281$ or $1.1 * 10^9$ possible structures respectively. These numbers are derived from the following equation (Robinson, 1977):

$$\mathrm{f(n)} = \sum_{\mathrm{i=1}}^{\mathrm{n}} (-1)^{\mathrm{i+1}} \cdot \binom{\mathrm{n}}{\mathrm{i}} \cdot 2^{(\mathrm{n-i})} \cdot \mathrm{f(n-1)}$$

Eq .2

Therefore, learning the most accurate structure is one of the main challenges in learning BN, and there are two main approaches to do that: search and score (S&S) and constraint-based (CB). These two kinds of approaches can be combined together for BN structure learning as: use the learned network from constraint-based methods as the starting point for the S&S methods. This approach is known as hybrid (Yasin, 2013).

2.3.2.1 Search and Score

This approach (Cooper & Herskovits, 1992; Heckerman et al., 1995) implements a procedure to search the space of DAGs and score each DAG based on the posterior probability of the DAG considering the set of variables and the data. The chosen DAG is the one achieving the highest value of a scoring function among only part of the

structures; thus, it is only sub-optimal. We will now present some of the most popular S&S heuristics.

2.3.2.1.1 Greedy Search

Greedy Search [GS; (Chickering, 2003)] is the simplest algorithm based on S&S technique. It is also known as Hill-climbing search (HC). This algorithm traverses the search space of the current graph, by examining only the possible local changes in its neighborhood, which is the set of all alternative graphs that can be generated by applying a single operator; add edge, delete edge, or reverse edge, such that these operators do not create a directed cycle in the new graph, then applying the one that is maximizing the score function. Hill-climbing search starts from an initial graph G_0. It iteratively constructs a sequence of graphs $\{G_0, G_1, G_2, \dots, G_N\}$ by examining their neighborhoods and choosing the graph having best quality function score. It stops when there is no more improvement in the existing graph score and the current graph has the highest score in its neighborhood (Yasin, 2013).

2.3.2.1.2 Tabu Search

Tabu Search [TS; (Glover & Laguna, 1998)] has emerged as one of the leading technologies for handling optimization problems that have proved difficult or impossible to solve with classical procedures. TS was developed to allow hill climbing to overcome local optima. TS includes short-term memory to prevent the reversal of recent moves, and longer-term frequency memory to reinforce attractive components. The basic principle of tabu search is to pursue the search whenever a local optimum is encountered by allowing non-improving moves; cycling back to previously visited solutions is prevented by the use of memories, called tabu lists, that record the recent history of the search. It is important to note that Glover did not see tabu search as a proper heuristic, but rather as a metaheuristic, i.e., a general strategy for guiding and controlling inner heuristics specifically tailored to the problems at hand.

2.3.2.1.3 Simulated Annealing

Simulated Annealing [SA; (Kirkpatrick et al., 1983)] is an effective and general optimization algorithm. It is useful in finding global optima in the presence of large

numbers of local optima and for escaping local optima. This algorithm is basically hill-climbing except instead of picking the best graph in each iteration, it picks a random graph. If the selected graph improves the existing graph score, then it is always accepted. Otherwise, the algorithm makes the move anyway with some probability less than 1 (Press et al., 1992). The probability decreases exponentially with the badness of the graph score, which is the amount ΔE by which the solution is worsened.

$$P(accepting\ a\ neighbor\ graph) \sim 1 - \exp\left(\frac{\Delta E}{\alpha * T}\right)$$

Eq. 3

A parameter T is also used to determine this probability. It is analogous to temperature in an annealing system. At higher values of T, uphill moves are more likely to occur. As T tends to zero, they become more and more unlikely, until the algorithm behaves more or less like hill-climbing. In a typical SA optimization, T starts high and is gradually decreased according to an "annealing schedule". The parameter α is some constant that relates temperature to energy, which in nature it is Boltzmann's constant (Press et al., 1992).

2.3.2.2 Constraint-based

In contrast to S&S methods, which compare complete DAGs to find the one that fits the data the best, most CB methods construct a DAG in two consecutive stages (Pearl, 2000). This approach (Spirtes et al., 2000; Yehezkel & Lerner, 2009) uses first the data for CI tests to determine if the corresponding edges should be learned or removed. Then, some orientation rules are used to direct all edges that can be directed.

One of the most notable CB structure learning algorithms is PC (Spirtes et al., 2000). Structure learning is performed using the traditional phases of global skeleton learning, v-structure orientation, and inductive orientation. The PC algorithm is based on the principle (Nam et al., 2007):

- Construct a skeleton of BN, having the relations between variables using conditional independence tests.
- For edge orientation, first find the v-structure (e.g., $X \rightarrow Y \leftarrow Z$) then infer some arrow orientations shared by all DAGs Markov equivalents. Further it propagates orientations of remaining edges.

2.3.2.2.1 Thresholding for independence tests

Using independence tests requires two things. The first is to choose an independence test and the second is to choose a threshold for the test that determines whether two variables are dependent or not given the test's statistic. For this matter, we base our method on a paper presented by (Lerner et al., 2013). The paper uses conditional mutual information (CMI) tests and suggests an adaptive threshold for the test that changes with respect to the given dataset size and the involved nodes' cardinality. The statistic of a CMI test between two variables **X** and **Y** given a separation set **S** is given by:

$$CMI(x; y|S) = \sum_{x \in X, y \in Y, s \in S} p(x, y, s) \log \frac{p(x, y|s)}{P(x|s) \cdot P(y|s)}$$

Eq. 4

(Lerner et al., 2013) suggested a few adaptive thresholds and empirically tested them against each other. The s_1 adaptive threshold showed the best results and hence was chosen for the current work. The s_1 is given by:

$$s_1 = \left(\frac{1}{2N}\right) \cdot X^2_{(|X|-1)\cdot(|Y|-1)\cdot\prod_{\forall z_i \in S}|z_i|,\alpha}$$

Eq. 4.1

Where N is the dataset size and X^2 is the chi-square critic value with degrees of freedom that are decided by the cardinality of both tested variables (X & Y), the cardinality of every node in the separation set (Z_i) and a user defined parameter α, which was set to be 0.01 in our work has it has shown the most accurate results.

2.3.2.3 Hybrid Methods

Both S&S and CB methods have their own advantages e.g., S&S based algorithms can deal efficiently with small datasets, whereas the ability to scale up to hundreds of thousands of variables is a key advantage of constraint based algorithms (De Morais & Aussem, 2010). These two kinds of approaches can be combined together for BN structure learning approach, i.e, a hybrid approach.

The Max-Min hill Climbing (MMHC) algorithm suggested by (Tsamardinos et al., 2006), combines both CB and S&S based approaches. The algorithm combines ideas from local learning, CB, and S&S techniques in a principled and effective way. It first

reconstructs the skeleton of a BN using a CB local algorithm Max-Min Parent Children (MMPC), which discovers the set of CPC (candidate parent-children, without distinguishing among both) for a target variable Y. The efficiency of MMPC is obtained by restricting the parent set. If two variables are found independent, then it assumes that they will not be connected in the final network. MMPC returns a very concise search space. Later on, in its second step the MMHC algorithm performs a Bayesian-scoring greedy hill-climbing search to orient the determined edges (Yasin, 2013).

2.3.3 Parameter Learning of a Bayesian Network

The parameters of a BN are conditional probabilities, reflecting the probability of a node to get a certain value conditioned on its parents' configuration. Every node in a BN has a conditional probability table (CPT) of its own, holding the probability of the node to receive every possible value given each possible parents' configuration (every combination of the parents' possible values). A CPT has a number of rows equal to the number of parents' configurations, each row is denoted as a conditional probability distribution (CPD). Learning the parameters (denoted as $\boldsymbol{\theta}$) from the data is most commonly done using Maximum Likelihood Estimation (MLE).

2.3.3.1 MLE for Bayesian Networks

The principle of maximum likelihood estimation, originally developed by R.A. Fisher in the 1920s, states that the desired probability distribution is the one that makes the observed data most likely. In general, we seek, as the name implies, the parameter values set that maximize the likelihood function. The likelihood function (denoted as *L*) measures how likely it is to obtain a particular data set, given a chosen model of interest. This expression contains the unknown model parameters $\boldsymbol{\theta}$. The values of these parameters that maximize the likelihood function are the maximum likelihood estimators. Loosely speaking, one can look at the likelihood as the probability to obtain that particular set of data. Suppose we have a data set **D** with m observations of set of domain variables **X**. The likelihood function is the probability that the model assigns to the data we observed. Using the joint probability distribution for BN, the likelihood function for BN is:

$$L(\boldsymbol{\theta}|D) = \prod_{t=1}^{m} P(\boldsymbol{X}_{[t]}|\boldsymbol{\theta}) = \prod_{t=1}^{m}\prod_{i=1}^{n} P(X_{i,[t]}|Pa_{[t]}(X_i), \boldsymbol{\theta}) = \prod_{i=1}^{n}\left[\prod_{t=1}^{m} P(X_{i,[t]}|Pa_{[t]}(X_i), \boldsymbol{\theta})\right]$$

Eq .5

The structure of BN allows us to reduce the parameter estimation problem to a set of local, smaller problems. Note that the term in the square brackets refers to the local conditional likelihood of a particular variable given its parents in the network. It turns out that we can maximize each local likelihood term independently of the others and combine these local maxima to get a global MLE solution for the whole network (Koller & Friedman, 2009). That is, the BN likelihood function can be decomposed as a product of independent terms, one for each CPT in the network. The local likelihood function for node X_i can be further decomposed into a product of simple likelihood functions corresponding to the node CPDs, each is a multinomial likelihood of variable X_i given a particular configuration of its parents.

We denote $Pa_j(X_i)$ as the j_{th} configuration of X_is parents, $Pa(X_i)$, and $\theta_{x_i|pa_{j(X_i)}}$ as the parameters of the multinomial distribution of X_i given its parents j_{th} configuration, which is compatible with the j_{th} row in the CPT of X_i. Using the variable multinomial MLE, we get:

$$\hat{\theta}_{x^k|Pa_j(X_i)} = \frac{C_{[x^k, Pa_j(X_i)]}}{C_{[Pa_j(X_i)]}}$$

Eq. 5.1

where $C_{[x^k, Pa_j(X_i)]}$ is the frequency of observations with $X_i = x^k$ and $Pa(X_i) = Pa_j(X_i)$ in the data, and $C_{[Pa_j(X_i)]} = \sum_k C_{[x^k, Pa_j(X_i)]}$ is the frequency of observations with $Pa(X_i) = Pa_j(X_i)$ in the data. Hence, $\hat{\theta}_{x^k|Pa_j(X_i)}$ is the proportion of observations with $X_i = x^k$, given $Pa_j(X_i)$.

2.3.4 Concept Drift in Bayesian Networks

The adaption of BN to new data is also related to sequential updating of the network, regardless of a possible drift. (Friedman & Goldszmidt, 1997) stated that because of errors in model construction and the dynamics of the domain, we cannot afford to ignore the information of new data and suggested sequential updating, which can also

overcome errors made in learning the initial model. They offer an incremental learning procedure to learn BN from an online stream of data. As for the part of sequential updating of the network structure, they compare the network structure with its neighbors' structures every K observations. Meaning, they do not deal with the task of detecting if a change occurred or not and instead, continually update the network whether a change has occurred or not. As mentioned earlier, learning, and updating the network may require considerable computing resources. In order to reduce the computational complexity, (Nielsen & Nielsen, 2008) offer a structure updating procedure (also called structural adaptation) with a change detection method, in which structure updating is performed only if a change is detected. To detect changes, they measure how well each of the last K observations fits the local structure of the monitored node in comparison with the fitness of the previous K measurements. If a significant difference is discovered between these two groups of K measurements, then it is assumed that a change in the network has happened. However, two main limitations arise with this approach. First, the measure of fitness of the local structure may detect nodes that do not change but their local structures do. For example, in Figure 3, (Nielsen & Nielsen, 2008) show that a change is detected for nodes A, C, and E, although A has not changed. Moreover, when an edge is reversed, but the linked nodes keep their Markov blankets, e.g., edge F→D, the change is not detected.

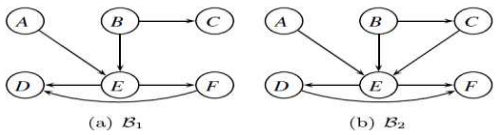


Figure 3: BN change [Nielsen & Nielsen, 2008]

Second, a limitation arises from the comparison of the last K observations only with the K observations before them. In the case of a gradual parameter shift, when the parameters are slowly drifted, this method may fail to detect a shift. However, it needs to be noted that gradual parameter-shift detection is not part of the goals of (Nielsen & Nielsen, 2008).

Another recent BN-based approach is locally adaptive-MB-MBC (Borchani et al., 2016), applying the Page-Hinkley test to the average log-likelihood in order to update locally around each node that is detected as changed.

2.4 CDDRL – Concept Drift Detection and Re-Learning

The CDDRL method was introduced by (Mendelson, 2020) and is in fact a union of two previous works, the first was done by (Nahum, 2015) and it offers a concept drift detection method using BN. The second work was done by (Cohen, 2018) and it offers an efficient method for re-learning the BN structure after a change was detected, using the first method's output.

Hence, the CDDRL includes two phases: First, the network parameters are learned at each time point t using MLE and the recent observations from the data stream. Then, the statistical distances are computed by measuring the distances of network CPDs from their corresponding values in the previous time step, and then aggregating these distances to a single distance per node.

For detecting changes in the BN as revealed by the measured distances, a sequential monitoring methodology is used, which is based on statistical process control (SPC) charts. By this methodology, each node is monitored separately and continuously, which allows to detect not only when a change has occurred but also where in the network it occurred, i.e., which nodes have been affected by the change. At a preliminary stage, the same distance calculations are done per node at a period that is known to be stable (no drifts) – we call this the *stable period*. The distances obtained at the *stable period* are used as the thresholds for alerting of a drift, i.e., if a node's calculated distance at any time step exceeds the node's threshold – the CDDRL alerts that this node is drifted. The threshold is called the upper control limit (UCL) and is calculated using the chosen SPC.

(Nahum, 2015) has tested different types of distance functions, aggregation methods and techniques for setting the UCLs and reported on a preferred combination with another slightly different combination that tradeoff between them in detection and false alarm rates. With respect to the aggregation method, the weighted mean function is preferred, the method gives weights to each distance measure by the proportion of its parent's configuration and then averages all the distances with the appropriate weights. We denote by d_i^j the statistical distance corresponding to the j^{th} configuration of the parents' node X_i, and by d_i^A as the aggregated distance of d_i^j over all parents configurations, thus the aggregation function is given by the formula:

$$d_i^A = \sum_{j=1}^{J} w_j * d_i^j, \qquad (\sum w_j = 1)$$

Eq. 6

The SPC chart that out-performed all others was the Exponentially weighted moving average (EWMA), by this method at each time point the distance used for comparison to the change detection threshold is calculated not only with the current distance but also gives weight to past distances with a smoothing parameter λ that is suggested to be set between 0.7 and 0.9. The EWMA distance is calculated by:

$$Z[t] = \lambda * d[t] + (1 - \lambda) * Z[t-1]$$

Eq. 7

where $Z_{[0]}$ is the target value of the process.

Finally, two different distance functions were suggested, the first was the Chi-Squared function yielding the highest detection rate but also higher false alarm rates and the second was the total variation function which yielded slightly lower detection rates accompanied with slightly lower false alarm rates. We will focus on the Chi-Squared function. The Chi-Squared function is given by (Cochran, 1952):

$$\mathrm{X}^2 = \sum_{k=1}^{K} \frac{(O_k - E_k)^2}{E_k}$$

Where K stands for the number of parents' configurations the node has, O_k stands for the observed parameters set of the Kth parents' configuration (the current parameters) and E_k stands for the expected parameters set of the Kth parents' configuration (the initial network parameters).

The second phase of the CDDRL as proposed by (Cohen, 2018) is a S&S methodology exploring the space of possible BN structures. Its input is the BN and the nodes that have been detected as changed (by the algorithm's first stage) from the previous time step. At each time step, the algorithm in the second phase searches for local changes in the previous graph only around the nodes which have been detected as changed, thus making the task of learning the BN structure less computationally expensive, as it doesn't learn the structure from scratch. At each iteration, a list of candidate graphs is created by either removing, adding, or reversing an edge from a node which was identified as changed at the first stage, avoiding possible cycles. All graphs in the neighborhood are scored using the Bayesian Dirichlet with likelihood equivalence and a uniform joint distribution [BDeu;

(Buntine, 1989)] and the best scoring graph is chosen to be the baseline graph for the next iteration, the iterations proceed until no improvement is found. Optimization of the process is done by simulated annealing.

Finally, (Mendelson, 2020) combined the two into one framework and conducted experiments to test the algorithm. Mendelson reported that in supervised settings, compared with streaming classification algorithms, the CDDRL was slightly better for both *sudden* and *gradual* as well as for *real* and *virtual* CDs, even though, the CDDRL was not developed for classification as were the other competitors. In unsupervised settings, which the former competitors cannot handle, the CDDRL was significantly superior compared to other static approaches.

In this work, the EWMA SPC chart with $\lambda = 0.8$ is used, coupled with the weighted mean aggregation and the Chi-Squared distance functions as suggested in (Nahum, 2015). Figure 4 summarizes the CDDRL's workflow. For a more detailed explanation of the CDDRL procedure as well as its theoretical correctness and the preliminary experiments done to test the different SPC charts, aggregation and distance functions - please refer to the article in Appendix 9.4 to the *Concept Drift Detection and Re-Learning, Theoretical Correctness* and *Preliminary experiments* sections, respectively.

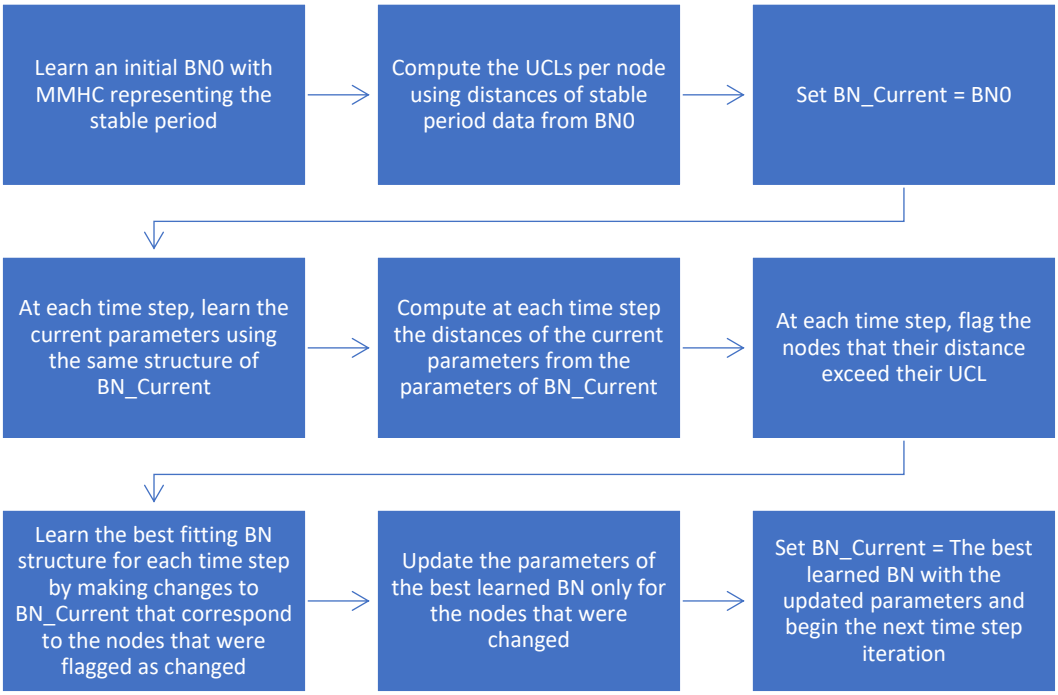


Figure 4: CDDRL workflow summary

2.5 Precision Agriculture

Although not being the main subject of this thesis, we conducted several experiments in the precision agriculture field, where we tested the CDDRL on the task of detecting stress in banana plants grown in a greenhouse. We also tested the CDDRL's ability to become a knowledge representation algorithm in these experiments. Therefore, we shortly explore the field of stress in plants.

Plants live in constantly changing environments that are often unfavorable or stressful for growth and development. These adverse environmental conditions include biotic stress, such as pathogen infection and herbivore attack, and abiotic stress, such as drought, heat, cold and nutrient deficiency (Jian-Kang Zhu, 2016). Plant diseases cause immense damage to the agriculture industry—damage that is reflected in productivity and economic aspects. In the USA, crop losses due to plant pathogens are estimated at $33B every year (Savary et al., 2012). One of the causes of such plant diseases is water stress (Osakabe et al., 2014). Water stress is developed in crops when the evaporative demand exceeds the supply of water from the soil (Slayter, 1967). Previous research, e.g., [(González et al., 1999; Sibomana et al., 2013)] has shown that water stress affects the total yield; therefore, early detection of plant stress is critical to minimize the loss of productivity. The severity of the damage depends on the duration between onset and time of detection. At the orchard level, the effectiveness of any remedial measures depends on the timely detection and identification of the cause of stress (Kim et al., 2011).

Early stress detection is crucial in bananas since this is the fourth largest fruit crop in the world and is considered to be a staple food in many countries (Surendar et al., 2013). Stress in banana plants cause visual and non-visual changes to its leaves. Water stress, for example, closes stomata and impedes photosynthesis and transpiration, resulting in changes in leaf color and temperature (Nilsson, 1995). Other symptoms of water stress include morphological changes such as leaf curling or wilting due to loss of cell turgidity (Kim et al., 2011).

3 CDDRL Developments

In this section, we introduce several developments to the CDDRL algorithm. In Section 3.1 we thoroughly introduce our new proposed method named constraint-based (CB) that is aiming to be an alternative for the second step of the CDDRL, i.e., the S&S process. In Section 3.2 we present four less intensive modifications to the CDDRL – the first alters the way in which the UCLs are calculated, two modifications slightly alter the S&S process, and the last proposed method changes the way in which the parameters of the learned BN are calculated at each time step. We test and evaluate all proposed methods with a series of CD-related experiments.

Finally, in Section 3.3 we attempt to elevate the CDDRL's predictive power by proposing two ensemble versions of the CDDRL. The first ensemble version uses all CDDRL modifications presented in Section 3.2 as its base estimators, whereas the second ensemble version uses previously learnt BNs as its base estimators. We test and evaluate the ensemble versions in three synthetic experiments as well as in seven real-world experiments in Sections 4.3 and 4.4, respectively.

3.1 Constraint-Based (CB) Re-Learning

The CDDRL currently uses a S&S method where possible graphs are iteratively generated. In each iteration, the previous graph is modified with all possible changes (addition, removal, and reversal of an edge). Every one single change to the baseline graph is saved as a graph in the current's iteration neighborhood of possible graphs. The changes applied are only those that hold with the information about the changed nodes from the first part of the CDDRL. Then all graphs in the neighborhood are given a BDeu score, and the best scoring graph is chosen as the baseline graph for the next iteration. When there is no improvement in the current neighborhood of graphs and a simulated annealing process does not apply – the process terminates, and the best scoring graph ever visited is returned.

The current version has two main limitations, the first is that the re-learning process might take a long time as the search space (depending on the number of variables in the domain and the number of those reported as changed) can still get quite large (many graphs might be generated in each iteration). The second limitation is that the method

might miss the optimal graph, since at each iteration the best scoring graph is chosen, and the algorithm keeps on searching greedily, where the optimal graph might be in a different path that could be reached if we were to choose at some point a slightly inferior graph in the neighborhood.

To deal with these limitations, the current work frame proposes a new method for re-learning the BN structure in the CDDRL algorithm. The new method utilizes the information from the first stage of the CDDRL about the changed nodes to perform only limited and specific CI tests between variables in the domain in order to discover the directed edges between them. CI tests when correctly controlled are less expensive thus making the new method a promising option to decrease run time. Since the new method is mainly based on statistical independence tests, it can be looked at as a constraint-based approach. However, at the last stage of the method, the final graph is generated by a scoring procedure, therefore the new method can be considered as a hybrid approach. Choosing a threshold for the independence tests is a tricky task that is influenced by many aspects; therefore, we use a dynamic threshold that changes with respect to the dataset's size and the cardinality of the features involved. The threshold is suitable for conditional mutual information (CMI) tests, and it was suggested and tested in (Lerner et al., 2013).

3.1.1 Suggested Method

The method proposed in this work has six main steps. The method receives from the first part of the CDDRL the previous known BN and the nodes that were detected as changed in the current time step. The method uses CMI tests, edge orientation rules and final BDeu scoring to reach the current BN which is the output of the algorithm. The general algorithm goes as follows:

Input: *Previous BN, Changed Nodes*
Output: *Current BN*
Step 1: Keep must stay edges (parents of non-changed nodes)
Step 2: CMI tests for previous edge removal
Step 3: CMI tests for possible edge addition (edges pointing to changed nodes)
** We don't know the orientation for edges between two changed nodes*
Step 4: V-Structures orientation
Step 5: Two extra edge orientation rules
Step 6: Final edge orientations using BDeu scoring

Algorithm 1: The CB algorthim summary

We will now go through all steps of the algorithm in depth, using the following example:

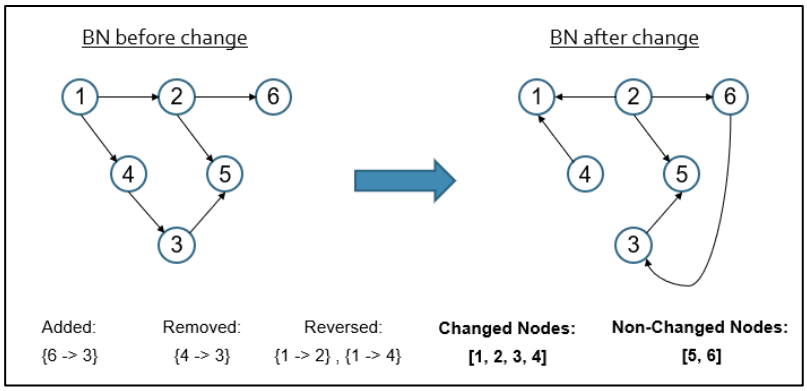


Figure 5: The example used to explain the CB algorithm. The figure shows the BN before (left) and after (right) the change. Added, removed and reversed edges are presented along with the changed and unchanged nodes.

In the given example a BN from the previous time step is given. The new underlying BN was constructed by making four changes to the previous BN (one edge addition, one edge removal and two edge reversals. Following the changes, nodes 1, 2, 3 and 4 have experienced change (they either had a parent added to them or removed from them). For our purposes, the first stage of the CDDRL is assumed to be successful, meaning that it alerts of all and only the nodes that have actually been changed. Therefore, the correct changed nodes are given as input to our method.

In addition, we set two critical things for any CB method; 1) the **maximum order** of the CMI tests – meaning the maximum number of nodes that can be in a conditioning set of any test. We set this to be the rounded down average of the **max-fan-in** and the **max-fan-out** of the known previous BN. Max-fan-in and max-fan-out are the maximum number of edges going into a single node and out of a single node, respectively. This measure gives an approximation of the density of the previous BN, thus indicating at some level the density of the unknown current BN (we assume that very drastic changes don't occur after one time step). Limiting the order of the tests is important since high order tests are less reliable and consume more run time. 2) Nodes that can be **candidates to enter the conditioning set** of any CMI test between two variables. We set these nodes to be the parents and the children in the known previous BN of both nodes currently being tested. The number of CMI tests needed to be done grows with the number of such nodes leading to an increase in run time, thus some limit is needed. Again, our assumption that tremendous changes won't occur in one time step, gives us the legitimation to rely on the previous BN for this matter.

Step 1: *Keep must stay edges (parents of non-changed nodes)*

As explained, a node is detected as changed if a parent was either added to it or removed from it. Therefore, all the edges pointing to non-changed nodes must stay (if they were removed, the pointed at nodes would have been detected as changed).

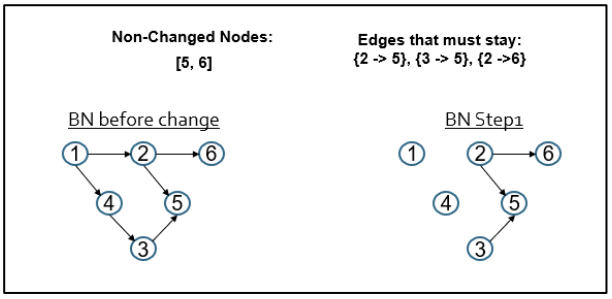


Figure 6: (left) the previous BN, (right) the current learned BN after step 1 – keeping the edges that point to non-changed nodes.

Step 2: *CMI tests for previous edge removal*

When performing a CMI test between two variables, we need to decide which nodes can be in the conditioning set. As explained, we take these nodes to be the parents and children in the previous BN of the nodes being tested. In our example the parents and children in the previous BN of all nodes are:

PC1 = [2,4], PC2 = [1,5,6], PC3 = [4,5], PC4 = [1,3], PC5 = [2,3], PC6 = [2]

For example, when taking a CMI test between nodes 1 and 2, the nodes that can be in their conditioning set are the union of PC1 and PC2 (without the nodes themselves) – nodes 4, 5 and 6.

At this step, we look at the edges that appear in the previous BN but are not kept at step 1, i.e., the edges pointing to changed nodes. We perform CMI tests between all two nodes that are at the head and tail of such edges. The possible nodes in the conditioning set are determined as explained above. The maximum order of the CMI tests in our example is 2 (average of max-fan-in and max-fan-out of the previous BN). For each pair of these nodes, we start with a CMI test that has an empty conditioning set and increase the number of nodes until the maximum order is reached. If the CMI values of all tests performed in this manner exceed the threshold (the variables are dependent) – the edge between them is kept. If the two nodes were both detected as changed – the kept edge is kept unoriented. If at any test the CMI value is below the threshold (the nodes are independent), then the edge is removed and the nodes that were in that test's

conditioning set are saved as the separation set of the two tested variables (this is useful for later stages). See Figure 7 for a visualization of this step in our example.

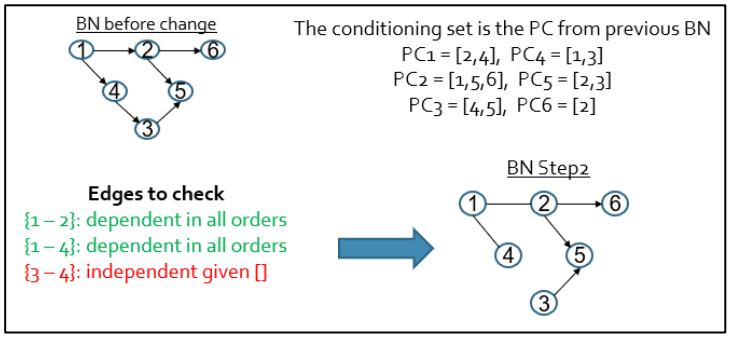


Figure 7: (top left) the previous BN, (top right) all PC sets, (bottom left) edges to check in the current step and their results, (bottom right) the current learned BN after step 2. In green are edges kept and in red those removed.

Step 3: *CMI tests for possible edge addition (edges pointing to changed nodes)*

This step completes the previous step. After looking at all the edges that were present in the previous BN, we now look at all possible edges that we can add. The possible edges are those that are not present in the previous BN, they point to a changed node and cycles in the current graph are not created with their addition. The edge addition process at this step is identical to the edge removal process explained in step 2. This step with respect to our example is presented in Figure 8.

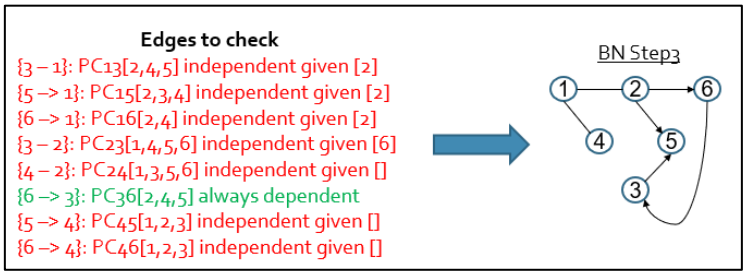


Figure 8: (left) all possible edges to add and their results, (right) the current learned BN after step 3.

Step 4: *V-Structures orientation*

According to (Verma & Pearl, 1990) - if nodes A, B, C hold the following rules:

- A - B - C / A -> B - C / A - B <- C
- A & C are not directly connected by an edge
- B is not in separation set of {A, C}

Then: A -> B <- C

We check for such situations and orient the edges that hold these rules.

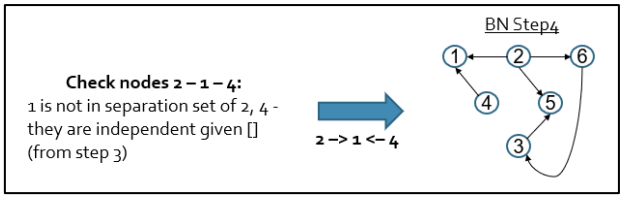


Figure 9: (left) all candidates for v-structure orientation, (right) the current learned BN after step 4.

Step 5: *Two extra edge orientation rules*

At this step we apply two extra edge orientation rules (if needed). In our example all edges are already oriented, therefore this stage is not applied to it. The rules are based on (Meek, 1995).

Rule 1: (Serial connections) if

- A -> B - C / A - B <- C
- A & C are not directly connected by an edge
- B is in separation set of {A, C}
  Then: A -> B -> C / A <- B <- C

This rule is correct since a different orientation will result in a v-structure that we assume we have discovered in the previous step.

Rule 2: (No cycles allowed) if

- A - B
- There is a directed path from A to B
  Then: A -> B

A BN cannot have cycles; therefore, this stage is correct. We show a simple example of this situation:

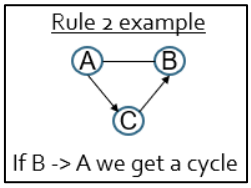


Figure 10: an example of the second rule – no cycles allowed in a BN.

Step 6: *Final edge orientations using BDeu scoring*

In case there are still unoriented edges, we use a S&S method to choose the final graph to return. The process goes as follows:

- Create all possible graphs (orienting the edges with no cycles)
- Give a BDeu score for all created graphs
- Choose the graph with the best BDeu score

The number of graphs needed to be checked at this step is: $2^{number\_of\_unoriented\_edges}$

*** Regarding our example, steps 5 and 6 are not needed, thus the outputted BN is the one achieved after step 4. This BN is identical to the real current BN presented in figure 2.**

3.1.2 A known Extra Edge Error

Since we take a subset of nodes that can be in the conditioning set of a CMI test between two variables (their parents and children in the previous BN), there can be some special cases where we will get an extra edge (EE) error. This situation will take place under the following circumstances:

- The previous BN holds a node (Z) that is not in the parents-children set of the two tested variables.
- The current changed BN holds a path (P) between the tested variables that goes through (Z).
- (Z) is not a collider in (P).
- There is no direct edge between the two tested variables.

If all of these hold up, we will mistakenly learn a direct edge between the two tested variables. This happens because (P) will not be blocked given any tested conditioning set - (Z) is the only one that can block it, but as assumed (Z) will not enter any tested conditioning set in this specific scenario. An example of such case is presented in Figure 11.

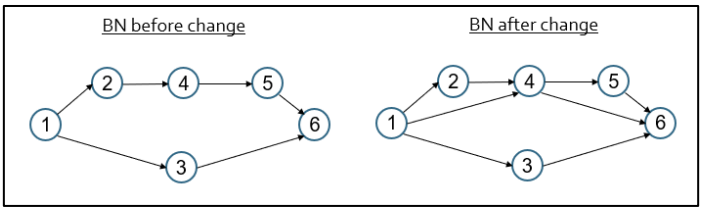


Figure 11: an example of the extra edge error.

Node 4 is not in PC {1,6} in BN before change, therefore we will not check it as a condition set between 1 and 6. There is a path between 1 and 6 that goes through 4 in BN after change, where 4 is not a collider in that path. Therefore, we will mistakenly learn $1 \rightarrow 6$.

3.1.3 Complexity of the CB Algorithm

The complexity of the suggested algorithm is almost completely bounded by the complexity of the PC algorithm, which is bounded by the largest degree in the graph, i.e., the maximum number of neighbors of any node. Our equivalence of the largest degree is the **maximum order** calculated as the rounded down average of the max-fan-in and max-

fan-out of the previous BN, which we note as mo. In the worst case, the number of CI tests in our algorithm (based on the PC algorithm) is bounded by (Spirtes et al., 2000):

$$2 * \binom{n}{2} * \sum_{i=0}^{mo} \binom{n-1}{i} \leq \frac{n^2(n-1)^{mo-1}}{(mo-1)!}$$

3.1.4 Constraint-Based Experiments

3.1.4.1 Structure Learning Experiments

In this set of experiments, five known BNs were taken from the 'Bayesian Network Repository' (Scutari, n.d.). We took two small networks – 'Asia' and 'Sachs', two medium networks – 'Alarm' and 'Barley' and one large network – 'Andes'. 'Sachs' and 'Barley' have a relatively large number of parameters. For each network an experiment was conducted. We made several manual changes to each original network and by that created a new 'changed' network. The number of manual changes increased with the size of the original network (number of nodes and edges). For each new 'changed' network, datasets of increasing size were sampled – 10 permutations of the data were sampled for each network and each dataset size. For the small networks, datasets of the following sizes were sampled – 500, 1500, 5000 and 15000 where for the medium and large networks, an additional dataset size of 50000 was sampled. Tested algorithms in the experiment were given all the datasets sampled and for each dataset learned a BN structure that is supposed to be a representative of the new 'changed' network. For both CDDRL versions (the old S&S-based version and our new CB version) the known original network was given as input as well as the correct nodes that were manually changed.

3.1.4.1.1 Competitors

- CDDRL_S&S – The original S&S-based CDDRL (Mendelson, 2020).
- K2 – S&S-based structure learning algorithm (Cooper & Herskovits, 1992).
- PC – Constraint-based structure learning algorithm (Spirtes et al., 2000).
- MMHC – Hybrid-based structure learning algorithm (Tsamardinos et al., 2006).

* The K2 algorithm is sensitive to the ordering of the nodes that it is given. The ordering of the nodes means that a node can only have another node as a parent if it appears before it in the ordering. We split the K2 algorithm into two algorithms – 'K2_Before' which is fed with the ordering of the previous known BN (which is a valid methodology) and 'K2_After'

which is fed with the ordering of the new unknown 'changed' BN (this methodology is not valid, since in a real scenario this BN is not known thus the ordering is also unknown). We show results of 'K2_After' as a baseline that outperforming it will be an impressive achievement.

3.1.4.1.2 Performance measures

- **Structural Hamming Distance (SHD)** – How many steps needs to be taken to reach the true underlying BN from the learned BN. A step is an edge addition / removal / reversal.
- **Runtime** – How long it took the algorithm to learn the BN structure (in seconds).
- **Number of CI Tests** – How many CI tests were done during the structure learning process. This is only relevant for the new CB version of the CDDRL and for PC.

* For each network, dataset size and algorithm, results were averaged over the dataset size's 10 permutations.

3.1.4.1.3 BNs Description

Asia is a BN that represents the connection between lung cancer and visits to Asia. Asia is a relatively small network with few nodes, edges, and parameters. The Asia experiment was divided into three experiments. The first was an edge addition, the second experiment was an edge removal and the third was an edge reversal. Results of the three experiments were aggregated and reported as one.

Sachs is a BN that represents the causal protein-signaling networks derived from multiparameter single-cell data. Sachs is a relatively small network in terms of number of nodes and edges but has a relatively high number of parameters.

Alarm is a BN that represents probabilistic reasoning for patients monitoring. Alarm is a medium size network in terms of number of nodes, edges, and parameters.

Barley is a BN that represents the production process of beer from Danish malting barley grown without the use of pesticides. Barley is a medium size network in terms of number of nodes and edges but has a very large number of parameters (nodes have high cardinality).
* The original version of the CDDRL was not tested in this experiment, since the high cardinality resulted in many BDeu calculations that exceeded the available memory.

Andes is a BN that represents an online student modeling for coached problem-solving. Andes is a large size network in terms of number of nodes and edges but has a relative medium number of parameters compared to its size (low cardinality of the nodes).

Table 1 summarizes the BNs' characteristics, i.e., the number of nodes, edges, and parameters in each BN, as well as their average degree (average number of neighbors) and their max in-degree (maximum number of parents any node has). For a more detailed explanation of the BNs' structure and characteristics, as well as the exact changes we manually induced to each BN, please refer to Appendix 9.1.

Experiment	Num. of Nodes	Num. of Edges	Num. of Parameters	Avg. Degree	Max in-degree
Asia	8	8	18	2	2
Sachs	11	17	178	3.09	3
Alarm	37	46	509	2.49	4
Barley	48	84	114005	3.5	4
Andes	223	338	1157	3.03	6

Table 1: Chracteristics of all BNs used in the structure experiments.

3.1.4.1.4 Results

Here we present the results of all five BN structure change experiments we conducted. Table 2 presents the SHD results of every tested algorithm under every experiment and dataset size. Marked in bold is the best algorithm at each combination of experiment (BN) and dataset size. Table 3 shows the same analysis with regards to the average run time of the algorithms. Finally, Table 4 holds the number of CI tests, which is a measure of complexity. Here, only our CB version and the PC algorithm are evaluated, since they are the only ones that perform CI tests.

Experiment	Dataset Size	CDDRL-CB	CDDRL-S&S	MMHC	K2-Before	K2-After	PC
Asia	500	0.6	**0.07**	2.83	1.53	1.27	2.53
	1500	0.3	**0**	2.03	1.0	0.57	1.3
	5000	0.03	**0**	1.7	0.73	**0**	0.2
	15000	0.03	**0**	1.67	0.77	**0**	0.17
Sachs	500	3.3	2.0	4.2	4.5	**1.6**	5.2
	1500	2.8	1.2	3.6	5.7	**0.1**	3.8
	5000	0.9	1.2	1.0	4.0	**0**	2.0
	15000	1.5	1.3	1.0	4.0	**0**	2.1
Alarm	500	9.5	**2.0**	17.4	15.5	9.8	20.9
	1500	6.5	**0.2**	11.6	9.5	3.7	12.7
	5000	1.5	**0**	7.8	8.4	2.4	7.0
	15000	0.8	**0**	6.5	7.8	1.8	6.7
	50000	**0**	**0**	9.0	7.1	1.1	3.6
Barley	500	**17.5**	-	73.8	47.9	46.3	77.6
	1500	**15.0**	-	69.2	30.7	28.5	64.4
	5000	**16.3**	-	59.3	22.3	18.3	55.3
	15000	14.6	-	52.9	17.8	**11.8**	47.0
	50000	12.9	-	39.8	16.0	**9.0**	41.7
Andes	500	29.9	**13.5**	218.5	147.2	140.2	171.5
	1500	21.9	**6.6**	145.8	56.0	44.4	113.6
	5000	12.4	**3.6**	84.0	35.3	23.2	74.1
	15000	10.2	**3.4**	60.6	26.1	12.7	61.3
	50000	6.3	**3.3**	55.3	27.5	9.0	38.3

Table 2: SHD results of all algorithms in every experiment (BN) and dataset size.
In bold is the best algorithm in a specific experiment and dataset size.

Experiment	Dataset Size	CDDRL-CB	CDDRL-S&S	MMHC	K2-Before	K2-After	PC
Asia	500	**0.1**	0.2	0.3	0.2	0.2	**0.1**
	1500	**0.1**	0.3	0.4	0.3	0.3	0.3
	5000	**0.1**	0.4	0.5	0.4	0.4	0.4
	15000	**0.2**	0.4	0.6	0.4	0.4	0.6
Sachs	500	0.5	**0.3**	0.4	0.5	0.6	0.8
	1500	0.7	**0.3**	0.6	0.6	0.6	1.5
	5000	1.4	**0.4**	1.4	0.9	0.9	3.9
	15000	1.9	**0.5**	3.0	1.1	1.1	7.9
Alarm	500	**1.4**	5.8	**1.4**	14.1	14.6	2.4
	1500	1.7	5.9	**1.6**	16.0	17.3	4.7
	5000	**3.1**	6.2	3.2	18.1	16.8	6.6
	15000	**4.4**	6.7	5.5	20.3	19.3	12.4
	50000	**12.0**	16.2	18.4	34.9	34.4	44.8
Barley	500	**1.2**	-	1.3	6.3	6.3	3.4
	1500	**2.0**	-	2.3	8.8	8.8	7.4
	5000	**5.1**	-	7.0	15.1	15.6	23.0
	15000	**13.6**	-	25.5	28.5	28.4	55.8
	50000	**55.8**	-	138.0	73.0	71.1	273.5
Andes	500	**4.1**	981.5	14.3	932.1	932.2	16.4
	1500	**8.5**	686.0	24.1	1049.3	1042.5	32.0
	5000	**19.7**	670.1	81.5	1363.7	1355.0	92.6
	15000	**50.9**	653.2	564.4	2230.1	2210.5	268.7
	50000	**193.6**	700.8	6095.6	5023.1	4939.6	1263.4

Table 3: Run Time (seconds) of all algorithms in every experiment (BN) and dataset size.
In bold is the best algorithm in a specific experiment and dataset size.

Experiment	Dataset Size	CDDRL-CB	PC
Asia	500	**25**	90
	1500	**28**	114
	5000	**31**	145
	15000	**33**	163
Sachs	500	**169**	471
	1500	**230**	722
	5000	**274**	1003
	15000	**277**	1114
Alarm	500	**577**	173304
	1500	**709**	2353
	5000	**884**	3113
	15000	**1013**	3758
	50000	**1157**	4725
Barley	500	**973**	3766
	1500	**1264**	5853
	5000	**1710**	9821
	15000	**2382**	13735
	50000	**3482**	24447
Andes	500	**6180**	36479
	1500	**8836**	52355
	5000	**12261**	71285
	15000	**14535**	88200
	50000	**17947**	122950

Table 4: Number of CI tests (complexity measure) of CB and PC in every experiment (BN) and dataset size. In bold is the best algorithm in a specific experiment and dataset size.

Asia Analysis:

In Table 2 it can be observed that regarding the Asia BN, the SHDs of all algorithms in all datasets sizes is low, this is expectable since the changes done to the original network are minimal (1 or 2 nodes changed in each experiment). Both versions of the CDDRL are far superior to all other algorithms. However, the old version of the CDDRL is superior to our new suggested version. This superiority decreases as the dataset size increases. Regarding run time (Table 3), all algorithms deal with the task relatively fast (simple task), but still, it is noticeable that our new version is the fastest algorithm. In Table 4, we can see a great difference in the number of CI tests done between our algorithm and the PC algorithm, showing that our algorithm reaches better results with less complexity than PC.

Sachs Analysis:

The Sachs network has many parameters relative to the number of nodes and edges, this causes results that show a different pattern than most other networks we tested. We can see at Table 2 that 'K2_After' is far superior over all other algorithms in all dataset's sizes. It is important to mention again that 'K2_After' is not a valid method in real life scenarios, since we fed it the ordering of the underlying changed network, which will not be known to us in real life – we took 'K2_After' as an 'elite' competitor to try to beat. Other than 'K2_After', both versions of the CDDRL perform best in all dataset sizes. Our algorithm improves with the increase of the size of the dataset and outperforms the old CDDRL version when the dataset has a size of 5000. In contrast to most other results, at this experiment our algorithm takes longer to run than most other algorithms in most dataset sizes. With alignment to all other results, our algorithm performs significantly less CI tests than PC.

Alarm Analysis:

Alarm is a well-known and evaluated medium-size network, thus further analysis was done on it and results had improved (Appendix 9.1). Regarding the current results, as expected, our algorithm improves significantly with the increase of the dataset's size, outperforming all other algorithms in almost every dataset size except for the old version of the CDDRL that is still superior. At the largest dataset size, both new and old versions of the CDDRL achieve SHD of 0. Our algorithm is the faster than all other algorithms for all dataset sizes (Table 3) and again outperforms PC with regards to complexity (number of CI tests - Table 4).

Barley Analysis:

Barley is a medium size network with a very large number of parameters (over 100,000), this caused the old version of the CDDRL to be too complex and memory-consuming and therefore it is excluded from the analysis. Our new method CDDRL-CB outperforms all other algorithms in the smaller datasets and outperforms all algorithms except 'K2_After' in the larger datasets. We can see an increase in SHD at the dataset size of

5000, this fact stood out and led to further analysis – results were improved (displayed in Appendix 9.1). Our algorithm outperforms all others in terms of runtime and complexity.

Andes Analysis:

Andes is a large network (number of nodes and edges) but not many parameters (low cardinality), thus the CDDRL old version was able to deal with it. Both CDDRL's versions outperform all other algorithms (including 'K2_After') for all dataset sizes, however, the CDDRL old version outperforms our new suggested version throughout the experiment. Starting from dataset size of 5000, the old CDDRL version stops improving in contrast to the new version that keeps on improving with the increase of the datasets size, allowing it to narrow the gap in performance to the old version. Here, the superiority of our algorithm regarding runtime is the most noticeable, being over 250 times faster than the old version at most. Again, our algorithm performs significantly less CI tests than PC.

3.1.4.1.5 Discussion and Further Analysis

We tested our new proposed CB method in five unsupervised structure learning experiments. Three state-of-the-art structure learning algorithms were used as competitors. The old search-and-score method of the CDDRL was also taken as a competitor and was the most important one we aimed to outperform. Unfortunately, the new algorithm was not able to beat the old version with respect to structure learning accuracy (SHD) and was inferior to it at most times. On the bright side, the new algorithm was significantly faster than all other algorithms, including the old version of the CDDRL. In addition, besides the old version, the new algorithm showed the best overall results out of all other competitors. Our new algorithm was also much less complex (less CI tests) than the PC algorithm.

Following the inferior performance of the new version compared to the old one, we broadened the analysis of the results. In particular, we broke down the SHD measure into its parts, i.e., missing edges (ME), extra edges (EE) and wrong directed edges (WD). The three measures were calculated after each step of the CB algorithm, thus giving us a clearer view of what kind of mistakes are more common and to what extent each part of the algorithm can be held accountable. In depth analysis of the above can be found in Appendix 9.1.1.

In short, the analysis showed that the most common error the CB algorithm makes is ME, meaning that edges that should have been learned – were not. The ME error decreases as the dataset's size grows, whereas the other two errors show a moderate increase with the dataset's size. This led to the conclusion that our threshold should start smaller than our current threshold and should get bigger than our current threshold with the increase of the dataset's size. This kind of threshold will supposedly have a better balance of missing and extra edges. I came up with a change to the current threshold's constant – replacing $2 * N\ (size\ of\ the\ dataset)$ by $80 * sqrt(N)$. This was achieved through trial and error and was successful in improving some of the results – again, proof of this improvement can be found in Appendix 9.1.1.

In addition, results showed that most of our ME error come from the second step, meaning that many edges that were present in the previous known BN had mistakenly been erased by our algorithm. This led to the thought of creating a different threshold for different parts of the algorithm. It is a reasonable assumption to make that networks won't change entirely in one timestep, thus a smaller threshold for the second step (easier to be dependent and learn an edge) can be a good solution – it will cause more edges from the previous BN to stay in the new learned BN. In addition, I found that most wrong direction mistakes come from step 5 (V-structures) and from the last step of the algorithm (BDeu scoring for final edge orientation). V-structures contradiction settlement technique as offered in (Simchon, 2011) and using a different score function such as the Minimal Description Length (MDL) measure (Lam & Bacchus, 1994) might be good candidates for WD reduction. These two suggestions as well as a dynamic threshold for each step of the algorithm is left for future research.

3.2 Minor Developments

We present new versions of the CDDRL. The new methods evolved and came to need during our work on synthetic applications as well real-world applications, especially in the fields of precision medicine and precision agriculture. In this work, we present the suggested changes to the original CDDRL and our motivation for these changes. We test all the suggested methods compared to the original version and also compared to other state-

of-the-art algorithms on a number of different experiments. The experiments results are also presented.

3.2.1 Dynamic UCLs

3.2.1.1 The Motivation

As mentioned, in order to detect changed nodes, we use a Statistical Process Control (SPC). At the original version, we calculate the upper control limit (UCL) for each node separately at the beginning of the process using the initial stable period. These UCLs are the boundaries of the SPC, when a node's parameters distance between the current timestamp and the previous one exceeds the node's pre-calculated UCL, then that node is reported as changed. We use the same initial UCLs throughout the whole process. Assuming the initial UCLs are adequate boundaries even after a substantial period of time has passed is not necessarily correct, especially since we are dealing with domains that can change drastically through time.

3.2.1.2 The Solution

We introduce a new version of the CDDRL called *Dynamic UCLs*. In this version, instead of using the same UCLs calculated at the very beginning of the process, the UCL of a node is recalculated in case it was not alerted of being drifted for a predefined number of time steps (a hyperparameter we call UCL_days). This way, we get UCLs that more adequately represent the current world.

3.2.1.3 Process and Hyperparameters

In the original CDDRL version, the UCLs are calculated at an initial stable period. We learn a BN using data from the entire stable period (we call it $BN_{baseline}$), then we use every *window* observations to learn the parameters of a new BN whilst keeping the structure of $BN_{baseline}$. We calculate the distances between the CPTs in $BN_{baseline}$ and those in each *window* observations BN. Finally, we use the calculated distances to obtain our UCLs using EWMA. These initial UCLs are kept constant throughout the entire data stream.

In the $Dynamic\_UCLs$ version, we use the same process to calculate the UCLs, except that we don't keep them constant, but rather we recalculate them every time we

observe another stable period, i.e., every node that is not detected as changed by the CDDRL for UCL_days is determined to be in a stable period and its UCL is recalculated in the same manner, using data from the prior UCL_days.

3.2.1.4 Pros

- We get a better representation of our process, considering the fact that our world is changing and thus our UCLs should be treated accordingly.
- In cases where our current environment is a bit noisier than the initial one, the new version will be more robust to false positive change detection – as the UCLs will be broader and will not alert of a change.

3.2.1.5 Cons

- The UCL_days hyperparameter can have a big impact on performance and therefore should be chosen carefully. It is dependent on the characteristics of the current environment. Setting a high value will make the UCLs more robust if the data for these time steps is relatively stable, however if there is a drift in the data, the UCLs will be relatively large (dependent on the drift strength) and thus will probably not detect a change in the following time step even if one actually occurred. In contrast, setting a small value might solve this problem, however the UCLs might be less reliable since they will be calculated using a small number of distances (std for example will not be reliable).

3.2.2 Tabu Search

3.2.2.1 The Motivation

The CDDRL algorithm uses a search & score method to determine the best fitting graph to the current data. The algorithm considers the changed nodes reported from the previous stage and generates all the possible minimal changes (graph neighborhood) of these nodes. Then the CDDRL scores all the graphs in the neighborhood using BDEu score and selects the highest scoring graph. Finally, the chosen graph is taken to be the baseline graph and another iteration begins. The new neighborhood holds graphs built by making changes to the new baseline graph according to the reported changes as before. Having this

workflow in mind, it is very probable that we will iterate back and forth between two or more graphs, causing additional unnecessary iterations and perhaps sticking us in a local minimum.

3.2.2.2 The Solution

We introduce a tabu list into the search and score process. The tabu list will hold graphs that we have already visited in the recent past, thus preventing them from being unnecessarily selected again. It is important to mention that the best scoring graph ever seen is independently saved at the side, hence preventing a revisit to any graph does not prevent it from ultimately being selected as the best fitting graph.

3.2.2.3 Process and Hyperparameters

The tabu list method has one hyperparameter – the tabu list's size. Saving all graphs ever seen in the tabu list will have a very negative effect with respect to runtime and complexity of the algorithm. Therefore, only the appropriate number of the most recently visited graphs needs to be saved in the tabu list - introducing the tabu list's size hyperparameter. The resulting process goes as follows: first the graph with the highest score in the neighborhood is selected, then the tabu list is observed:

- If the tabu list is empty – place the selected graph at the top of the tabu list and start another iteration with the selected graph as baseline.
- If the tabu list is not empty – check whether the selected graph is already in the tabu list. If it is in the tabu list, select the next highest scoring graph in the neighborhood and repeat the process. If the selected graph is not in the tabu list, then place it at the top of the list.
- Check if the tabu list's size exceeds the tabu list's size hyperparameter, if it does – remove the oldest graph from the list.
- Begin another iteration with the best scoring graph that was not in the tabu list.

3.2.2.4 Pros

- Might save runtime by not allowing repetition of processes we have already done i.e., revisiting graphs.
- Helps escaping from a local minimum during the search for the best fitting graph.

3.2.2.5 Cons

- Adds a hyperparameter to the process. Wrongfully selecting the tabu list's size may lead to undesirable results – if the list is too short then it is practically irrelevant and unnecessary. On the other hand, if the list is too large, it will resolve in a significant increase in runtime as we need to go through the whole list at every iteration to check if the selected graph is in it.

3.2.3 Robust Search

3.2.3.1 The Motivation

During the S&S process, the CDDRL uses simulated annealing. This way, if there is no graph in the neighborhood that yields a higher score than what we have already reached, the process does not automatically stop. At this scenario, the simulated annealing comes into play – the best score in the neighborhood is taken (again, this score is lower than what we already have), and then a probability is computed. This probability will increase as the current score is closer to the best score seen so far. We will make another iteration, selecting the graph that has a lower score than what we have already seen with the computed probability. The purpose of doing this supposedly worse step is to help us escape from a local minimum. However, this stochastic process might lead to slightly different overall results on different runs of the algorithm, depending on a random seed pre-selected. The robust version of the CDDRL is meant to deal with this issue.

3.2.3.2 The Solution

We introduce a version that reduces – but does not eliminate completely – the stochastic effect of the simulated annealing process. We broaden our search space, thus increasing the chance that different runs will reach the same graphs. The process goes as follows:

- Find the best scoring graph in the neighborhood – same as before.
- While there is a score improvement in every iteration – continue as usual.
- At the first time that there is no better score in the neighborhood than what we currently have – we do not go to the simulated annealing part but rather we search parallelly in three directions:
    - We take the best scoring graph in the current neighborhood (though lower than what we already have).
    - We take the second-best scoring graph in the current neighborhood.
    - We take a random graph from the current neighborhood.
- We use each of the mentioned three graphs above as a new baseline and start the search from the beginning separately for each graph. Inside each of the separate searches, the search process is identical to the original search – using the simulated annealing.
- Each of the three independent searches reports the best graph it had find. If the highest scoring graph between the three has a higher score than the score we had before the search split, then it is selected and reported back, otherwise the best graph that we observed prior to the search split is ultimately selected.

3.2.3.3 Pros

- Reduces the stochastic part of the simulated annealing, resulting in more consisting results between different runs.
- We are looking at a broader search space, though not too broad as each search is still bound to look only at graphs created with respect to the known changed nodes. Therefore, we are increasing our chances to reach a graph that better fits the current data.
- No hyperparameters are needed.

3.2.3.4 Cons

- Since we expand our search space, the runtime of the robust algorithm is increased.

3.2.4 Parameter Learning

3.2.4.1 The Motivation

The CDDRL uses MLE to learn the parameters of the learned BN at each time step. The issue with MLE, is that it may overfit the data (Moons et al 2004, Guiver & Snelson 2009). In our case, it might happen given a small *window* size, or a specifically noisy *jump*. In those cases, it may be harmful to the algorithm's process, as the next *jump* changed nodes will be determined by the distance between current parameters and the next, which in turn might trigger an unnecessary learning process at best, or a wrong one at worst. Additionally, if the BN is used for prediction, it may harm the network's ability to generalize.

3.2.4.2 The Solution

We suggest integrating the previous data-jump while learning the parameters of the current time step, in order to make the parameter learning of the current network more robust. In this case, we view the past jump as a prior distribution of our theoretical distribution, and the current one as evidence of change. In our case, parameters ($CPTs$) are fitted using a weighted average between those learned in the current and in the previous time steps. That is:

$$New\_CPT\{node\} = Current\_CPT\{node\} * weight + Previous\_CPT\{node\} * (1 - weight)$$

For each node in the network. It is important to note that both the $Current\_CPTs$ (i.e., the ones learnt using the current data jump) and the $Previous\_CPTs$ (previous data jump) are learned through MLE, while copying the structure of the current network, to produce $CPTs$ of the same size.

3.2.4.3 The Weight Hyperparameter

We developed an automatic procedure to determine the $weight$ hyperparameter that controls the contribution of the current and past learned $CPTs$ to the final reported $CPTs$. The automatic procedure tries to quantify the amount of change from the previous time step each node went through and gives bigger weights to nodes that changed more than others. The motivation is that if a node changed drastically, this means that the past is less relevant.

The number of changes from the previous time step within each node's Markov Blanket (that is, how many new nodes are in the Markov Blanket, and how many nodes that used to be there are no longer there), is used as a measure to the extent of change the nodes went through. The final weight attached to each node is relative to the amount of measured changes the node went through compared to all other nodes - more change results in bigger weights.

3.2.4.4 Pros

- This method "relaxes" the changes in parameters, hence lowering the probability of making a false alarm while identifying changed nodes.
- Helps the model generalize better by using a prior distribution for parameter robustness.
- Performed after structure learning "ad-hoc". This promises that we do not affect the structure learning process, which is the most complicated area of BN learning.

3.2.4.5 Cons

- In sudden drift situations, it will take longer for the learned network to perform well, as past data is used.
- Makes the algorithm slightly slower due to calculating the parameter of the older network using the current network structure.

3.2.5 Minor Developments Experiments

3.2.5.1 Alarm Experiments

The Alarm network is a popular medium sized network with 37 nodes, 46 edges, and 509 parameters. It is frequently used as a benchmark for experimentation. A more detailed explanation about the alarm BN was discussed in Section 3.1.4.1.3. We simulated two different experiments with the Alarm Network. In both, 10 permutations of the data were sampled and in both, the first step is to adjust the network to a change, as shown in the illustration below:

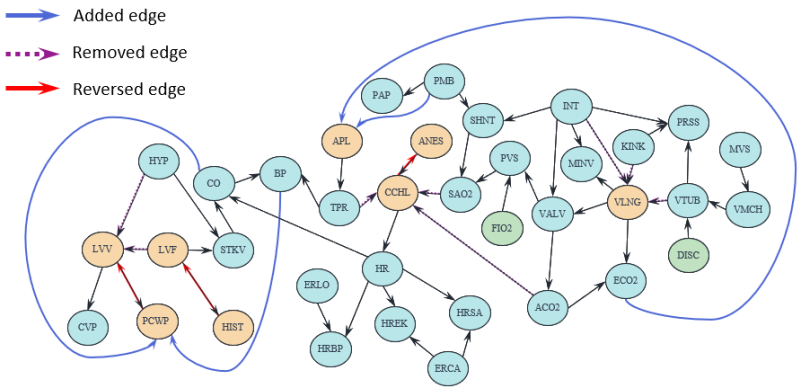


Figure 12: The Alarm netowrk and changes done to its structure.

3.2.5.1.1 Alarm Experiment 1

Overview: In the first experiment, we sampled 10000 observations from the original Alarm network, and then 5000 samples from the adjusted Alarm network. This is referred to as Sudden drift, as the data is generated only from the new network from the 10001st observation moving forward. The goal in this experiment is to see whether the CCDRL versions manage to adjust to the new changed network. Each network is learned using the last 500 samples, giving us 30 time steps.

Evaluation Metric: To quantify our success, we used the Structural Hamming distance (SHD) that counts the number of steps (add, remove, or reverse an edge) needed in order to reach the real underlying network from the learned network. Up to the change point (the 10000th observation), we compare the learned networks to the original Alarm network, whereas starting from the 10001st observation we compare the learned networks to the adjusted Alarm network.

Results:

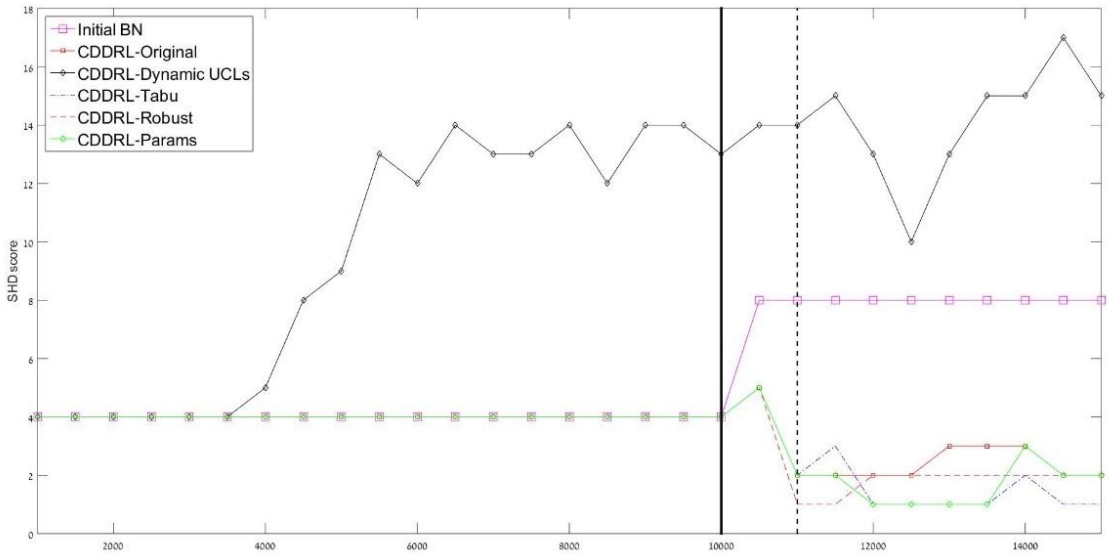


Figure 13: Alarm SHD results of all CDDRL versions

The purple line represents the initial BN, learned by MMHC. That network, to begin with, had a SHD score of 4 compared to the real underlying Alarm network. Due to our algorithm using it as reference, it is understandable that during the stable stage (up to 1000), SHD remained at a constant 4. This is true for all algorithms except for the *Dynamic_UCLs*, which performed strangely very poorly in this specific experiment, perhaps due to hyperparameter choice.

Once the stable threshold is crossed, the networks are compared to the adjusted network, which has a SHD score of 8 compared to the original reference network. As expected, all of the algorithms (except *Dynamic_UCLs*), managed to adjust and learn networks closer to the real post-change network. Unfortunately, it seems that all of them, including the original algorithm, managed to achieve scores between 1-3, and not 0. However, it is mentionable that the *Tabu, Robust and Params* versions (although not significantly) had a better performance than the original version; the *Tabu* version had a harder time adjusting to the change but starting from the 12000th observation was almost stable till the end on a SHD of 1, having the best performance in the last time steps. The *Robust* version adjusted best to the change and then was the most stable version – stabling on a SHD of 2. Finally, the *Params* version was a bit noisier but was never outperformed by the original version in the post-change time steps and outperformed the original version in most cases.

3.2.5.1.2 Alarm Experiment 2

Overview: In the second experiment, we focus on prediction. To do so, we needed to manually change the Alarm network, as there is no target variable in the original network. Therefore, we created a new node, which does not exist in the original Alarm network – this node is referred to as the class node. Up to the 10000th observation, the class node is assigned with class $'0'$ with a 70% chance and class $'1'$ with a 30% probability. However, all of the sample are generated from the same original Alarm network. Starting from the 10001st observation until the 15000th observation, the samples are generated either from the original Alarm network or from the adjusted one, with a 50% chance for each. If the sample was generated from the original Alarm network, we label it as class $'0'$. Otherwise, the sample was labeled as class $'1'$.

Process and Evaluation Metric: For this experiment, every *jump* of 500 data points a network is learned using the past *window* of 1000 samples. Then the learned network predicts the label of the class node for the next *jump* of samples, which acts as a test set. Again, this procedure produces 30 time steps. For this experiment, we use the Accuracy metric to judge the quality of predictions.

Results:

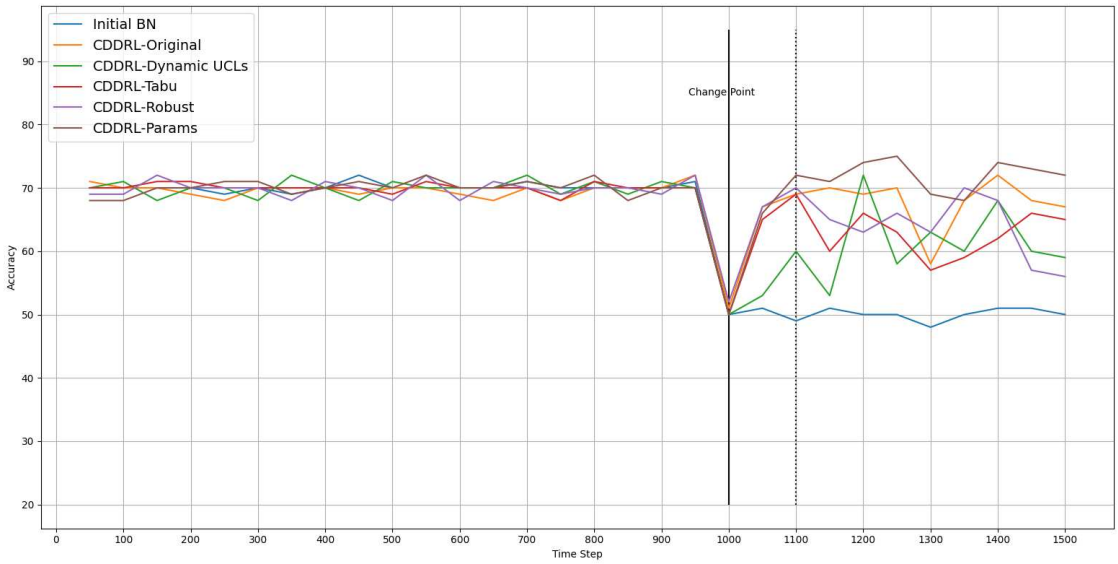


Figure 14: Alarm accuracy results of all CDDRL versions

During the stable phase, all networks always predict the first class, as it is a-priori more likely, and not dependent on any other nodes. After the change point, the *Initial BN*

still only predicts the first class, which reduces its accuracy to 50%. The other algorithms learn the new dependencies and manage to keep the accuracy higher. Unfortunately, most of our new methods do not show superiority over the original CDDRL version. While on average the *Params* version seams to outperform all other versions including the original version, this superiority is not significant. Once again, the *Dynamic_UCLs* version continues to be the least stable and shows the noisiest results. Perhaps creating an Alarm version with bigger changes, which will create bigger dependencies and hence room for improvement, will allow us to reveal the potential differences between the algorithms in this regard.

Runtime:

As part of the experiments, it was important for us to understand how the different changes we made affect runtime. Runtime was calculated in seconds as an average over all 20 permutations (10 permutations for each one of the two Alarm experiments) and an error bar representing the std. It is important to note runtime of each algorithm was calculated including the scoring of the network, which is a constant added to all networks. Hence, this should only be taken as a comparison between them, and not the actual run time of learning the networks.

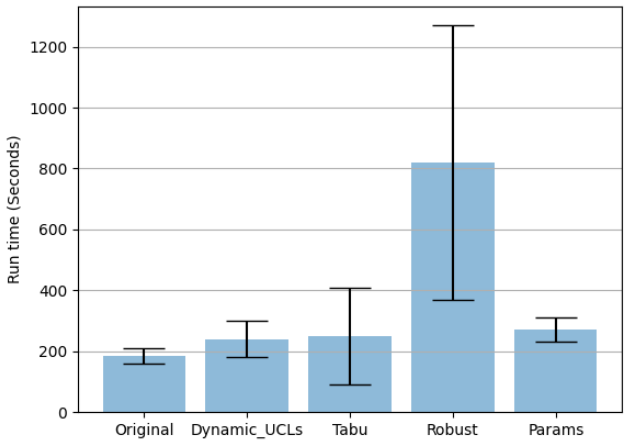


Figure 15: Avarage and std of Alarm run time over 10 permutations of all CDDRL versions

As expected, the original algorithm is the quickest, which makes sense, as the other versions only added steps to it. However, only the *Robust* version is significantly higher and the *Tabu* version is relatively noisier.

3.2.5.2 STAGGER Experiment

Overview: Stagger is a synthetic number generator. Stagger produces 3 variables: color, size, and shape. Each variable can get 3 different values: color (red, green, blue), size (small, medium, large), shape (circular, square, triangle). We used two of the Stagger's classification functions: the first classifies a positive observation if the observation is red and small and the second classifies an observation as positive if its green and circle.
We produced 20,000 observations in each of the 10 permutations. The first 10,000 observations are 'stable' using only the first classification function, then a *real CD* was introduced by changing the classification function gradually by using the probability to use the second classification function (CF) as given by:

$$P(CF_2) = \begin{cases} \dfrac{1}{1 + e^{-4\cdot(t-p)/\eta}}, & if \geq p \\ 0, & else \end{cases}$$

Where t is the current time step, p is the change point (the 10,000th observations) and n is the drift's width (5,000 observations), which means that starting from the 15,001st observation, all samples are label according to the second CF.

Process and Evaluation Metrics: We used *jump* of 100 observations, leading to 100 time steps before the drift and another 100 after the drift. The performance metrics used were accuracy and std between the accuracy in the 10 permutations, the minimum accuracy ever reached, the recovery time (how many time steps it took the algorithm to reach back 95% accuracy) and run time.

Results:

Algorithm	Avg. Accuarcy	STD Accuarcy	Min Accuacry	Recover Time	Run Time (Sec)
CDDRL-Original	0.9721	0.064	0.702	**21**	6.55
CDDRL-Dynamic UCLs	0.9736	0.060	0.707	**21**	7.06
CDDRL-Tabu	0.9737	0.060	0.704	**21**	6.86
CDDRL-Robust	0.9726	0.065	0.693	**21**	17.31
CDDRL-Params	**0.9741**	**0.059**	**0.709**	**21**	**6.52**

Table 5: CDDRL versions results in the STAGGER experiment with gradual drift of 5,000 samples

In general, no significant difference can be seen between the CDDRL versions. However, the *Params* version can be pointed out since it outperformed (although slightly) all other CDDRL versions including the original one in all evaluation metrics. The *Params*

version received the highest average accuracy, the lowest std of accuracy between the permutations (the most stable version), the lowest recovery time (together with all other versions) and surprisingly had the shortest run time (even shorter than the original version). The *Robust* version should be pointed out as having the worst performance, coming in last place in all categories except the average accuracy where it had the second worse performance after the original version. It is also worth mentioning that all versions outperformed the original CDDRL in the average accuracy metric. The *Tabu* version with a list size of 5 and the *Dynamic_UCLs* version showed average performances, interchanging the second place among themselves in almost all categories.

3.2.5.3 Hyperplane Experiment

Overview: In this experiment we wanted to test the CDDRL performance in a domain that is relatively large – 60 features with 5 categories each and 2 classes. The hyperplane generator is a synthetic data generator implemented in *scikit* using the idea introduced in (Huang & Dong, 2006). The idea is that each variable gets a weight (randomly at the beginning). The separation plane is in a d-dimensional space (d is the number of features) that is built by the set of points X_i that satisfy: $\sum_1^{di} W_i X_i = W_0$, where W_0 equals the sum of all weights. An observation is classified as positive if $\sum_1^{di} W_i X_i > W_0$ and negative otherwise. All features receive a uniformly distributed value between 0 and 1, which we later discretize using equal bins with 5 bins.

The drift is introduced by selecting a number of drifted features and a magnitude of change. The weight of each drifted feature is recalculated at every observation generated by the stream in this manner: $W_i = W_i + d\sigma$, where d is the magnitude of change (between 0 and 1) and σ is either 1 or minus 1 according to a preset probability that the rotation of the hyperplane will change its direction i.e., the weights will decrease rather than increase. This results in a rotation of the separation plane, thus introducing a *real CD*.

Data and Process: In our experiment we created a dataset where the magnitude of change was set to be 0.1. We chose the total number of features to be 60 and the number of drifted features was set to 15. In addition, the probability that the hyperplane will change its direction of rotation was set to 20%. Finally, a 5% noise was added to the data.

We used a total of 3,000 observations: the first 1,000 were generated with no drift (magnitude of change = 0), the next 1,000 were generated with the drift as explained above and the last 1,000 were set to stable on the new drifted concept i.e., magnitude of change = 0 but the weights were kept as they were at the end of the drift (after 2,000 observations).

Competitors and Evaluation Metric: Since this is the only place we evaluated the hyperplane experiment (Alarm and STAGGER were are also evaluated in the previous and next sections respectively), we wanted to test the CDDRL in general against other competitors as well. Therefore, five other competitors were evaluated (KNN-ADWIN, AWE, DWM, LNSE, SRP). Since the dataset became at some parts slightly imbalanced, we used the F1-score which is an adequate performance measure for these kinds of tasks.

Results:

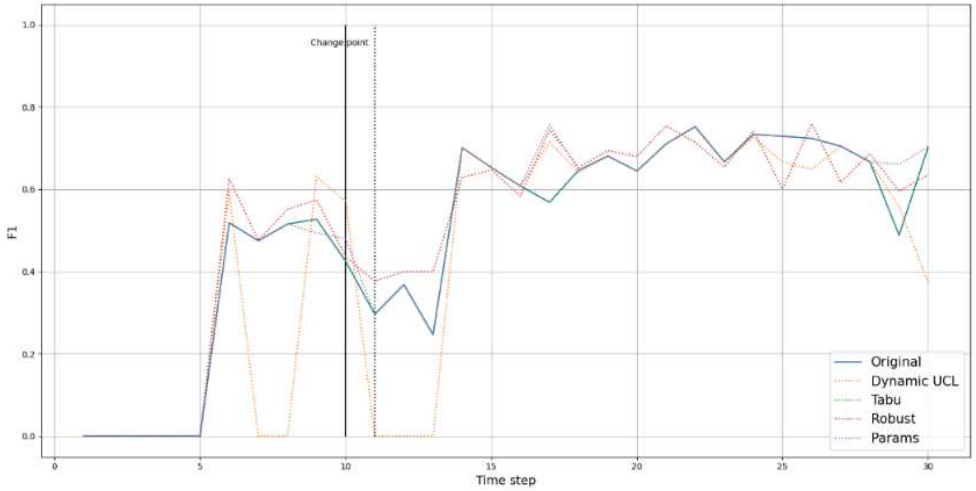


Figure 16: Hyperplane F1 results for all CDDRL versions

All CDDRL versions show similar results with no clear winner, however the *Params* version can be pointed out as it performed at least as good as the original version through out the whole experiment and outperformed it on several occasions. The robust version can also be pointed out as it outperformed all other versions on several time steps, but was also the worst one on several other time steps. Finally, the dynamic UCL version had the worst overall performance and presented noisy results.

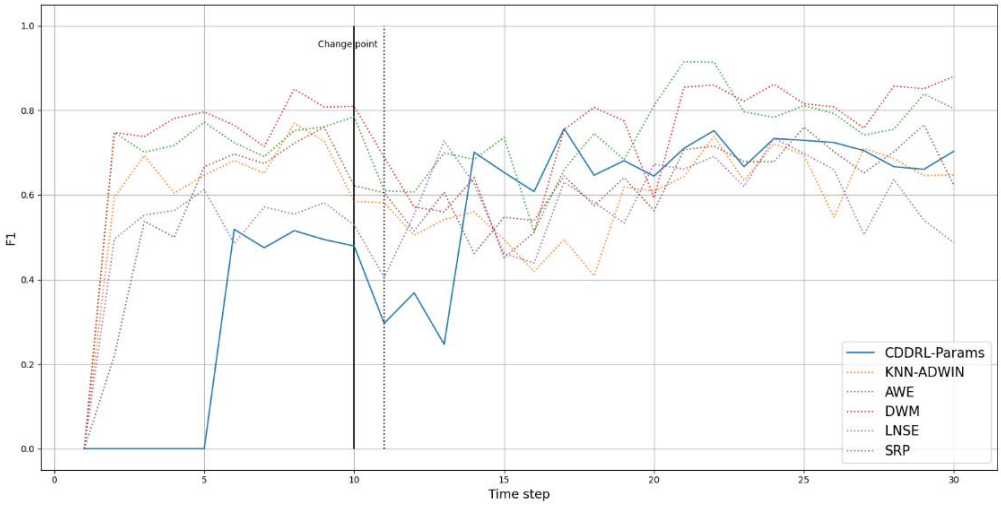


Figure 17: Hyperplane F1 results of the CDDRL Params version compared to five competitors

We took the CDDRL *Params* version and compared it to the other five competitors. It can be observed that the CDDRL's version performed at least as good as KNN-ADWIN, LNSE and SRP, outperforming them on many cases. On the other hand, AWE and DWM outperformed all others including the CDDRL's version during most of the time steps of the experiment.

3.2.5.4 Conclusions and Discussion

In this section we introduced four developments to the CDDRL original algorithm. These versions were developed due to encounters with certain situations where we thought the original algorithm might not be sufficient. Each new proposed version introduced a single change to the original CDDRL version; the *Dynamic_UCLs* changed the way in which we learn our drift-thresholds, i.e., UCLs, and made them dynamic instead of constant. The *Tabu* version introduced a tabu list to the BN structure search process, prohibiting the algorithm from revisiting recently observed solutions. The *Robust* version expended the search space after the BN structure and the *Params* version changed the way in which we learn the BN's parameters, i.e., CPTs - in the new version, opposed to the original version, past CPTs affect the currently learned ones.

We tested the new proposed versions with several CD related experiments and compared them to the original CDDRL. We used the well-known Alarm network to conduct two separate experiments; the first was an unsupervised *sudden* drift experiment with the task of adjusting to the suddenly changed network. The second experiment was a

supervised experiment, where we added a class node to the network that represented the origin of the sample, i.e., the original or the changed network.

The STAGGER benchmark was also used to test the algorithms in a supervised task that included a *gradual* drift. Finally, the rotating hyperplane generator was utilized to create another experiment, where the decision boundaries are rotated, thus creating a scenario of a *real CD*.

In all four experiments, there was no version that was significantly superior over the others. However, in the first Alarm experiment, all versions (except for the *Dynamic_UCLs*) showed a better SHD or were more stable than the original CDDRL. In the second Alarm experiment, the *Params* version was superior to all others (again, not significantly) and the *Dynamic_UCLs* continued showing inferior and noisy results. Aggregated run times of both Alarm experiments revealed that as expected the *Robust* version had a significantly higher run time compared to all others, whereas the rest of the versions had similar run times and the original version being slightly quicker than all others.

The *Params* version once again showed the best results during the STAGGER experiment, slightly superior to all others in all criteria tested, and was even faster than the original version. Regarding the hyperplane experiment, results were very close, where in general we can say that again the *Params* version was slightly superior to all others, achieving an F1-score at least as good as the original version and outperforming it on several occasions. Compared to five external competitors, the *Params* version was superior to three of them (KNN-ADWIN, LNSE and SRP), but was inferior to the other two (AWE and DWM), thus making us partially satisfied with the overall CDDRL's performance in a high dimension task.

To sum up, we can state that most of the proposed developments did not harm the performance of the CDDRL (except for the *Dynamic_UCLs* regarding prediction capabilities and the *Robust* version with respect to run time), and in many cases were slightly superior to the original version. In an overall perspective, we can declare that the *Params* version showed the most promising results. Finally, since the new versions did not provide a satisfactory improvement to the CDDRL's prediction capabilities on their own, a combination of them was suggested to form a CDDRL-based ensemble. This ensemble, along with another proposed ensemble version of the CDDRL are discussed and evaluated (as well as the original CDDRL) in the next section using a series of synthetic and real-life datasets.

3.3 CDDRL-Based Ensembles

Since the developments offered in the previous section did not significantly improve the CDDRL's performance, we turned to ensembles applications. Ensembles are very common in state-of-the-art CD algorithms. Ensembles are known for their predictive capabilities, combining several base estimators to obtain elevated classification performance by reducing the variance of the model. Two CDDRL-based ensembles will now be presented.

3.3.1 Ensemble Methods

We introduce the first ensemble method of the CDDRL. Each one of the four CDDRL versions introduced in the previous section is a base estimator in the ensemble. At every time step we run all four versions separately and save their individual predicted class probabilities. We then aggregate their predicted class probabilities using a weighted average to obtain single predicted class probabilities and use those to predict the observations in the current time step. At first the weights are equally distributed between the versions, but at every time step transition, each weight is multiplied by the normalized AUC performance of the corresponding base estimator on the previous time step (normalization is done by dividing each estimator's AUC by the sum of all estimators' AUC), therefore, increasing the influence of better performing versions on the ensemble in future time steps. Algorithm 2 shows a pseudo-code of this explained version's process that we call $Ensemble\_Methods$.

input: $Weights, D_{[t]}, Base_{estimators}$

$ProbabilitiesArray \leftarrow []$
$AUCsArray \leftarrow []$
for $estimator = 1$ to length$(Base_{estimators})$ **do**
  $ProbabilitiesArray[estimator] \leftarrow$ Predict class probabilities using $estimator$
end for
$Probabilities_{Agg} \leftarrow []$
for $i = 1$ to length$(Weights)$ **do**
  $Probabilities_{Agg} \leftarrow Probabilities_{Agg} + Weights[i] * ProbabilitiesArray[i]$
end for
$Predictions \leftarrow$ Predict $D_{[t]}$ using $Probabilities_{Agg}$
$AUCsArray \leftarrow$ Calculate AUC for each estimator on $D_{[t]}$ using its predicted class probabilities
for $j = 1$ to length$(Weights)$ **do**
  $Weights[j] \leftarrow Weights[j] * AUCsArray[j]$
end for
$Weights \leftarrow$ Normalize $Weights$ by dividing each weight by the sum of $Weights$
$return \quad Predictions, Weights$

Algorithm 2: Ensemble Methods algorithm

3.3.2 Ensemble BNs

The second ensemble version of the CDDRL works in a similar manner to the first $Ensemble\_Methods$ version. We call this version $Ensemble\_BNs$. Here, the base estimators used in the ensemble are the previous BNs learned by the original CDDRL version. The number of BNs in the ensemble is a predefined number. At every time step the currently learned BN contributes 50% to the final prediction of the ensemble, whereas the existing BNs in the ensemble divide among themselves the other 50% by their weights representing their past performance. The new learned BN enters the ensemble if its AUC performance in the current time step is higher than the worst performing BN in the ensemble, in that case the worst performing BN is removed from the ensemble. Weights updating and prediction aggregation is done in the same manner as in the first ensemble method. The motivation for this ensemble is that when dealing with data streams and CDs, past knowledge might be influential e.g., in case of *recurring* drifts.

4 Experiments

In the following sections, we validate the original CDDRL as well as the two ensemble versions of it with a series of experiments. First, we conducted three synthetic classification experiments where *real* and *virtual CD* were present. The magnitude of the drift, i.e., *sudden* or *gradual*, was also a variant between the experiments. Second, we test our algorithms using seven real-world experiments. Four of the experiments are based on known CD related datasets gathered from real-life applications, and the other three were datasets created by us while conducting experiments in the precision agriculture field, i.e., experiments where fertilizer and water stress were induced to banana plants.

4.1 Competitors

In all 10 experiments (3 synthetic and 7 real-world applications), 11 different competitors were tested against our algorithms. The chosen competitors vary with respect to their passive / active manner, where the active competitors differ among themselves regarding the drift-detector in use. In addition, some of the competitors deal with CD via constant adaption of a single model, whereas the others hold several models learned through time, categorizing them as ensemble algorithms. Each ensemble algorithm has its unique way to aggregate its base estimators and to add or remove base estimators from the ensemble. Finally, the competitors use different types of base estimators, e.g., decision trees, KNN and neural networks. We intentionally selected an extensive number of competitors varying in many CD-related aspects, making our evaluations and conclusions about our algorithms' performance much more robust. All competitors were implemented using the python library (*Scikit-Multiflow*), except for the last competitor that was introduced in and implemented using (Baier et al., 2021). We will now present all competitors and the parameters we used for each, for a more detailed introduction of the competitors please refer to Section 2.2:

- KNN-ADWIN – the ADWIN drift-detector coupled with the known KNN classifier.
- AWE – a passive ensemble method that weighs classifier decisions based on their expected classification accuracy on the test data.
- DWM – a passive ensemble method that uses weights to remove models from the ensemble and to add new models based on the global performance of the ensemble.

- LNSE – a passive ensemble method that dynamically weighs the classifiers based on their accuracy in the current environment compared with past environments.
- SRP – an active ensemble method that uses Hoeffding trees and random subspaces together with the ADWIN drift-detector to deal with CD. We used 10 trees instead of the default 100.
- SAM-KNN – an active ensemble method that builds KNN classifiers that correspond to short and long-term data, helping it adjust to data streams.
- VFDR – an active learner using the EDDM drift-detector and rules in decision trees as the base estimator to address CD.
- RSLVQ – a passive learner using the *Adadelta SGD* to adjust to CD, we use a momentum of 0.9.
- OBC – a boosting classifier coupled with the ADWIN drift-detector to cope with CD.
- ARF – an active ensemble method using decision trees to create a random forest. We use the KSWIN as a drift-detector.
- NN-UN – an active learner based on neural networks that uses the ADWIN drift-detector and the model's predictions uncertainties to deal with CD.

* Other than the parameters values mentioned above, all algorithms were used with their default parameters values.

4.2 General Methodology

We assume data arrives in batches of one time step and update the models after each batch arrives using the current and previous time steps data. For example, at time step t we use data from time step t and time step $t-1$ to train the models and then we test time step $t+1$ using these trained models. After data from $t+1$ is predicted, it is then used again together with the data from t to update the models before predicting $t+2$, and so on. The first time step is predicted randomly since we don't have any data to train the models with beforehand.

For the following experiments, we set *jump* to 1 and *window* to 2. There is also no known BN_0 that represents BN^*_{pre} of the stable process. Thus, we learned it directly from the data using the first four time steps (which are considered stable for all of the

experiments in this section). The structure was learned using min-max hill climbing [MMHC; (Tsamardinos et al., 2006)] and the parameters (CPTs) using MLE.

Finally, the evaluation metrics changed between the experiments with respect to the task in hand and to the balanced / imbalanced nature of the datasets. The evaluation metrics will be discussed separately for each group of experiments.

4.3 Synthetic Experiments

4.3.1 STAGGER Experiments

To test the performance of the CDDRL (the original version as well as both ensemble versions) as a streaming-classification algorithm in *real CD*, we used the STAGGER benchmark. STAGGER is thoroughly explained in Section 3.2.5.2, the only difference is that here we conducted two experiments; the first is identical to the one presented in Section 3.2.5.2 with a drift width (η) of 5,000, this drift width is referred to as the *gradual* STAGGER experiment. We also wanted to evaluate the algorithms under a *sudden* drift, thus another experiment with a drift width (η) of 500 was done. The evaluation metrics in both experiments were the same metrics as in Section 3.2.5.2 – run times will be discussed together with the rest of the experiments at the end of the section.

4.3.1.1 STAGGER Sudden Results

Algorithm	Average	Std	Min	RT
CDDRL-Original	0.9955	0.034	**0.683**	**4**
CDDRL-Ensemble$_{Methods}$	**0.9960**	**0.032**	**0.683**	**4**
CDDRL-Ensemble$_{BNs}$	0.9912	0.057	0.510	5
KNN-ADWIN	0.9890	0.060	0.579	6
AWE	0.9892	0.063	0.579	**4**
DWM	0.9915	0.053	0.646	**4**
LNSE	0.9667	0.120	0.445	13
SRP	0.9936	0.045	**0.683**	**4**
SAM-KNN	0.9916	0.053	0.595	**4**
VFDR	0.9893	0.052	0.660	8
RSLVQ	0.8772	0.125	0.622	*100*
OBC	0.9919	0.049	0.678	5
ARF	0.9919	0.047	**0.683**	5
NN-UN	*0.7376*	*0.264*	*0.421*	*100*

Table 6: STAGGER Accuracy results with a sudden drift width of 500 observations. In bold is the best algorithm and in italic and underscore is the worst algorithm in every criterion.

Table 6 shows that the *RSLVQ* and *NN-UN* algorithms were drastically inferior to all other algorithms, where RT (Recovery Time) of 100 represents that these algorithms never reached back an accuracy of 95% after the drift has occurred. A possible explanation for their bad performance is that as the authors of the *NN-UN* algorithm stated, *NN-UN* will not

detect well *real CD* when it appears without a *virtual CD* as well. The *RSLVQ* has no active drift detection mechanism, but rather uses a momentum-based SGD to deal with the streaming data. The rest of the algorithms achieved similar very good results in terms of the average accuracy (besides the LNSE, which was significantly inferior), the *Ensemble Methods* version of the CDDRL was slightly better than the other algorithms. These results are not trivial, since the others were designed for streaming-classification, whereas the CDDRL is for general (unsupervised) CD detection. In addition, the original CDDRL as well as the *Ensemble Method* CDDRL versions achieved the least noisy results over the stream (lowest Std). These two versions also had the best performance in terms of RT together with the AWE, DWM, SRP and SAM-KNN algorithms with only four time steps to return to high accuracy. However, at the change point, where all of the algorithms dropped their performance, AWE, DWM and SAM-KNN dropped to a lower (Min) accuracy compared to both CDDRL's versions and the SRP. The *Ensemble BNs* version of the CDDRL was slightly worse than the other two CDDRL versions in this task and had an average performance compared to the other competitors.

4.3.1.2 STAGGER Gradual Results

Algorithm	Average	Std	Min	RT
CDDRL-Original	0.9721	0.064	0.702	**21**
CDDRL-Ensemble$_{Methods}$	**0.9747**	**0.057**	**0.712**	**21**
CDDRL-Ensemble$_{BNs}$	0.9681	0.076	0.693	24
KNN-ADWIN	0.9676	0.073	0.697	26
AWE	0.9599	0.091	0.690	26
DWM	0.9654	0.082	0.691	23
LNSE	0.9472	0.119	0.566	25
SRP	0.9707	0.066	**0.712**	24
SAM-KNN	0.9699	0.069	0.700	23
VFDR	0.9466	0.100	0.614	35
RSLVQ	0.8575	0.151	0.536	*100*
OBC	0.9610	0.081	0.629	27
ARF	0.9669	0.068	**0.712**	28
NN-UN	*0.7919*	*0.214*	*0.468*	*100*

Table 7: STAGGER Accuracy results with a gradual drift width of 5000 observations. In bold is the best algorithm and in italic and underscore is the worst algorithm in every criterion.

Table 7Table 7: STAGGER Accuracy results with a gradual drift width of 5000 observations. shows similar results to those achieved for the *sudden CD*. The original CDDRL and the *Ensemble Methods* CDDRL outperformed the other algorithms in terms of average Accuracy, Std and RT. The *Ensemble BNs* version of the CDDRL once again was inferior compared to the other two CDDRL versions and had an average performance compared to the other competitors.

4.3.2 Sin Experiment

Since *virtual CDs* are not as common as *real CDs*, we designed an experiment to test the algorithms' performance under *virtual CD*. In this experiment, we sampled 30 time steps, each consisting of 1,000 samples. We sampled three input variables according to the following distributions:

$$X_1 \sim \begin{cases} B(p = 0.2, n = 4), & \text{if } t < d \\ B(p = min(1, 0.2 + 0.01 * t), n = 4), & \text{else} \end{cases}$$

$$X_2 \sim \begin{cases} BOL(\lambda = 1, N = 3), & \text{if } t < d \\ BOL(\lambda = \frac{1}{1+0.1*(t-d+1)}, N = 3), & \text{else} \end{cases}$$

$$X_3 \sim \begin{cases} BB(a = 2, b = 1, n = 4), & \text{if } t < d \\ BB_{(a=2+0.1*(t-d+1), b=1, n=4)}, & \text{else} \end{cases}$$

where d is the change point, p and n are the parameters of the binomial (B) distribution, λ and N are the parameters of the Boltzmann (BOL) distribution, and a, b, and n are the parameters of the beta-binomial (BB) distribution. With all three variables, the change in distribution caused them to gain higher values with a higher probability. This is reasonable if we think of them, for example, in terms of size, which naturally grows with time. Thus, for X_1, we increased p (the probability of success in one trial for a binomial variable), since the expected value for the number of successes is $n * p$ (where n is the number of trials), and as p grows, $n * p$ also grows for fixed n. Similar ideas have been applied for X_2 and X_3, with their own distributions.

The target variable was uniformly sampled in $[0, 1]$ until the change point (i.e., in the first 10 time steps). In the 10th time step, a *real CD* has been applied, and the target variable is 1 if one of the following conditions is fulfilled:

$$\{X_2 = min(X_2)\} \;\&\; \{X_3 \leq Q_1[X_3]\} \text{ or}$$

$$\{X_2 = min(X_2) + 1\} \;\&\; \{(X_1 > \overline{X_1} + 1 \;||\; X_1 < \overline{X_1} - 1)\}$$

and 0 otherwise, where $Q_1[X]$ is the 0.25th quantile of X. The indicators for all of the variables are computed at each time step (e.g., $min(X_2)$ and $Q_1[X_3]$). The first change is *real CD* since $p(y|x_1, x_2, x_3)$ changes from uniform (which is independent in x_1, x_2, x_3) to dependent in x_1, x_2, x_3, with the rules defined above. However, from time step 11 on, the distribution of x_1, x_2, x_3 keeps changing, whereas $p(y|x_1, x_2, x_3)$ remains the same. Hence,

from time step 11 on, *virtual CD* is introduced. The experiment design resulted in an imbalanced classification task; hence, we report here the F1 measure.

4.3.2.1 Sin Results

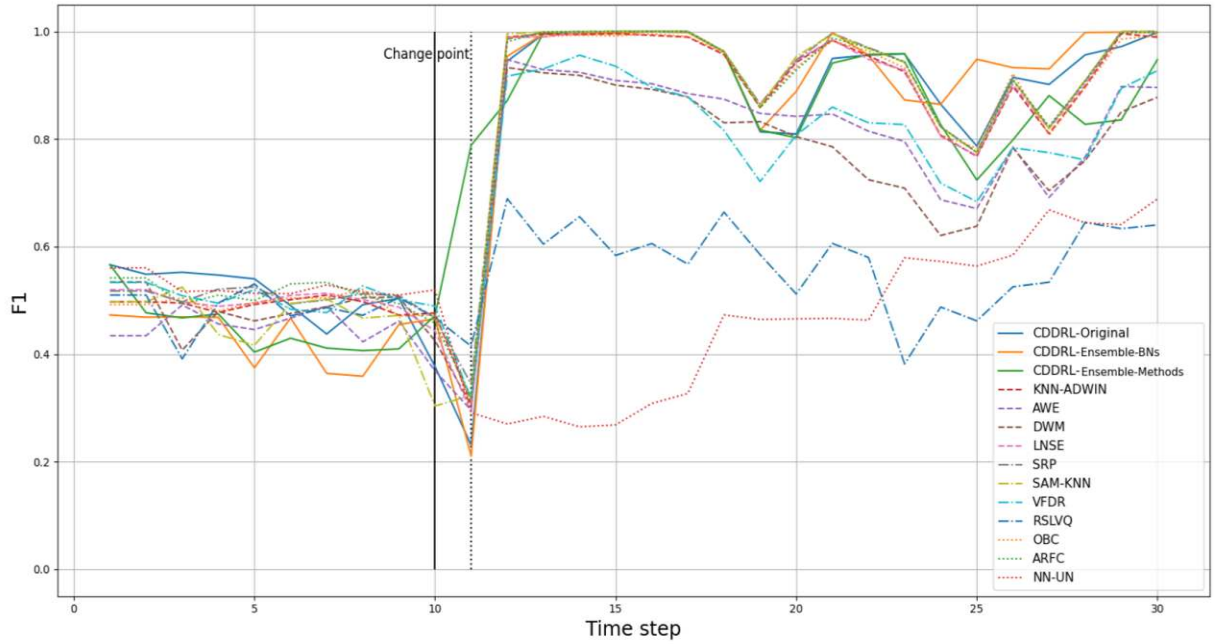


Figure 18: F1 results of the Sin (virtual CD) experiment of the CDDRL and all competing algorithms

Figure 18 shows that in the first 10 time steps, all of the algorithms achieved F1 of about 0.5, resulting from random guessing of the target variable as either 1 or 0 (it was randomly assigned in this period). Since *window* $= 2$, at time step 11, the data used to train the algorithms is mixed (half of it is drifted and the other is not), therefore all algorithms drop their performance except for the *Ensemble Methods* version of the CDDRL that managed to significantly improve its performance at time step 11 but is too still adjusting to the changed data. From time step 12 on, all of the algorithms improve their performance, whereas the *Ensemble BNs* CDDRL version is superior compared to all other algorithms and achieves a higher or equal F1 in almost all time steps. The original CDDRL is the second-best algorithm and is inferior to the *Ensemble BNs* version starting from day 23. The *Ensemble Methods* version although showing a strong performance at time step 11 (*real CD*), falls in performance compared to the other two CDDRL versions starting from day 23 but is still at least as good as most of the other competing algorithms.

4.3.3 Summary of Synthetic Experiments

In general, both the original and the ensemble versions of the CDDRL present promising results in both *real* and *virtual CD* tested in this section compared to all evaluated competitors. It is important to note again that the CDDRL was not initially created for supervised settings, thus the superiority of the different CDDRL versions over algorithms that are built for supervised settings, is very encouraging. Regarding both STAGGER experiments, the *Ensemble Methods* was the best performing algorithm in all criteria tested, whereas the original version was the second-best in terms of average accuracy in both experiments. The *Ensemble BNs* was less successful than the former two versions in the two STAGGER experiments, nevertheless, in terms of average accuracy, there were only two other competitors that outperformed it in the *gradual CD* experiment and five in the *sudden CD* experiment.

Concluding the Sin experiment, the previous relatively inferior *Ensemble BNs* version is now the most successful algorithm, presenting the highest F1-score in most time steps. Here, the *Ensemble Methods*, previously the best performing algorithm, is now inferior with respect to the two other CDDRL versions (although it is the only algorithm that did not drop its performance after the change point, which is also remarkable). All three CDDRL versions are part of the superior group of algorithms (together with *KNN-ADWIN, LNSE, SRP, SAM-KNN, OBC* and *ARFC*), where *AWE, DWM* and *VFDR* are in the second-best performing group and finally *RSLVQ* and *NN-UN* continue to show inferior results.

Summing up, we carefully state that the ensemble versions were successful in improving the CDDRL's predictive power, where it seems that the *Ensemble Methods* is more suitable for *real CD* (STAGGER) and the *Ensemble BNs* is more suitable for *virtual CD* (Sin).

4.4 Real-World Experiments

Up to this point, all experimentations with the CDDRL were done using synthetic data. Although synthetic experiments are important since CD can be controlled and manipulated, it is at least as important to test CD algorithms in real-life applications (even though we don't always know the behavior of the CD in them), which eventually are the main goal of such algorithms. In these experiments we test and evaluate the original CDDRL as well as

both ensemble versions on seven real-world datasets. Four of the experiments used CD-related datasets collected from known sources and the other three were experiments done by us in the agriculture field.

4.4.1 Datasets – Real Life Applications

Here we present the four known datasets we used that include natural or manual induced CD and the source of each of them.

4.4.1.1 Electricity

Electricity is a widely used dataset described by (Harries, 1999). The data is collected from the Australian New South Wales Electricity Market. In this market, prices are not fixed and are affected by demand and supply of the market. They are set every five minutes. Electricity transfers to / from the neighboring state of Victoria were done to alleviate fluctuations. The class label identifies the change of the price (UP or DOWN) in New South Wales relative to a moving average of the last 24 hours.

4.4.1.2 AWS Spot Price

Amazon Web Services [AWS; (*Amazon Web Services*, 2022)] provides virtual computing environments via their EC2 service that can be configured with up to 16 GPUs. You can launch instances with your favorite operating system, select pre-configured instance images or create your own. However, you can request Spot Instance Pricing. Which basically charges you for the spot price that is in effect for the duration of your instance running time. They are adjusted based on long-term trends in supply and demand for Spot instance capacity. Our goal here was to predict Spot pricing for two regions in Central California between March and May 2017 (Visser, 2017).

4.4.1.3 Usenet

The Usenet dataset is based on the 20 newsgroups collection. They simulate a stream of messages from different newsgroups that are sequentially presented to a user, who then labels them as interesting or junk, according to his / her personal interests. The user's interest changes among the batches (we have five batches each contains 300 instances),

there are three newsgroups type of articles: *Medicine*, *Space* and *Baseball*. Each observation holds 99 binary features that state whether certain words are present or not in the specific article. Predicting the interests of the user was the task in hand, where the changes in the user's preferences introduce a *real CD* to the problem. Table 8 shows the user's interest in each time step (Katakis et al., 2008).

Topic \ Step	0 - 300	301 - 600	601 - 900	901 - 1200	1201 - 1500
Medicine	+	-	+	-	+
Space	-	+	-	+	-
Baseball	-	+	-	+	-

Table 8: Usenet dataset - The topic in which the user is interested in (+) and not interested in (-) at each time step

4.4.1.4 NYC Subway Traffic

This dataset holds data about hourly passenger traffic within the NYC subway system with each station having its regular seasonal, daily, and weekly fluctuations. The dataset includes the number of subway station entries, at 4-hour intervals, for 469 subway stations from Feb. 4th 2017 to Aug. 13th 2021. In addition to 5 time and stations' attributes, each of the 469 stations in the dataset were referenced to one of 51 neighborhoods, each associated with 12 aggregated financial and demographic variables. Our task here was to predict for every time step (month) whether the number of entries will be low (entries in the 0-1/3 percentile), medium (entries in the 1/3-2/3 percentile) or high (entries above the 2/3 percentile) using previous time steps' data (Gerber, 2021). Table 9 gives an overview of the characteristics of the four datasets.

Dataset	# Instances	# Attributes	# Time Steps	# Classes
Medicine	45,312	8	83	2
Space	199,800	6	54	3
Space	1,500	99	15	2
Baseball	105,000	17	70	3

Table 9: Real-world datasets overview

4.4.2 Datasets – Precision Agriculture

In addition, we performed three real-life scenarios experiments in the agriculture field. The goal of the experiments was to detect manually induced stress in banana plants that were grown in a greenhouse. In each of the three experiments we grew (together with our partners from Rahan Meristem) a different number of banana plants for a different number of days. We divided the plants into four equally sized groups and treated each with a

different fertilizer / water regime, intentionally causing plants to get fertilizer / water stressed in different levels. Every experiment started with a few days where all plants received the same treatment with the recommended amount of fertilizer / water the plants need. We relate to these days as the stable period and use them to learn BN_0. In each experiment, we collected different data about the plants from cameras and sensors. Table 10 sums up the settings and objectives of each experiment. A more detailed explanation of the entire water stress experiment process can be found in Appendix 9.2.

Experiment Name	# of Plants	Stable Days	Total Days	Treatment Regime	Data Collected	Objective
Fertilizer Stress	201	7	28	A - 200%, B - 100%, C - 67%, D - 0%	RGB images - Features of size and color	D Vs. All
Fertilizer + Water Stress	120	4	17	A - 100%, B - 80%, C - 60%, D - 40% (of water and fertilizer)	Sensors - Temperature, SPAD (color) and Water left in the plant's tray	Multiclass - A, B, C, D
Water Stress	188	13	41	A - 100%, B - 80%, C - 60%, D - 40%	RGB, Thermal, Depth images - Features of size, color, temp and height	D Vs. All

Table 10: A summary of the three stress experiments - number of plants, stable days and total growing days are presented as well as the data collected and the objective for each experiment

4.4.3 Evaluation Metrics

Since several experiments had the task of binary class predictions (Electricity, Usenet, Fertilizer Stress, Water Stress) and the rest had the task of multiclass predictions (AWS Spot Price, NYC Subway Traffic, Fertilizer + Water Stress), we used different evaluation metrics for each task. For the binary tasks, we used the following metrics: False Alarm Rate (FA) defined as the number of false positives out of the total ground truth negatives, F1-Pos (when class 1 is treated as the positive class), F1-Neg (when class 0 is treated as the positive class), F1-Avg (the average of F1-Pos and F1-Neg), Geometric Mean (G-Mean), Balanced Accuracy (B-Accuracy), Area Under the ROC Curve (AUC) (except for AWE, DWM, SAM-KNN and RSLVQ that cannot produce class probabilities, thus cannot be evaluated using AUC). For the multiclass tasks, we used: Weighted Precision, weighted Recall, weighted F1 and weighted AUC, as well as the Kappa and Mathew's coefficients. All evaluation metrics are built to deal with imbalanced data, which is the case in most of our datasets. A detailed explanation of the different evaluation metrics we used can be found in (Espíndola & Ebecken, 2005; Grandini et al., 2020), whereas the mathematical formulation of the metrics can be found in Appendix 9.3.

4.4.4 Results

Table 11 and Table 12 show the experiments' average results (over all time steps) of the three CDDRL versions (original and two ensemble versions) and all competing algorithms. In bold is the best performing algorithm for the specific metric and in italic + underscore is the worst performing one. Cells are colored in a green-red color range, where the darkest green and darkest red fill the best and worst performing algorithms respectively, separately for each metric.

Algorithm	FA	F1-Pos	F1-Neg	F1-Avg	G-mean	B-Accuracy	AUC
CDDRL-Original	0.19	0.59	0.75	*0.67*	0.64	*0.71*	*0.77*
CDDRL-Ensemble-Methods	0.14	*0.56*	0.78	*0.67*	*0.62*	*0.71*	0.79
CDDRL-Ensemble-BNs	0.14	**0.69**	**0.82**	**0.75**	**0.73**	**0.76**	**0.86**
KNN-ADWIN	0.19	0.65	0.78	0.72	0.70	0.73	0.80
AWE	0.16	0.63	0.79	0.71	0.68	0.73	-
DWM	0.12	0.60	0.80	0.70	0.66	0.72	-
LNSE	0.21	0.68	0.78	0.73	0.72	0.74	0.85
SRP	0.17	0.67	0.80	0.73	0.72	0.74	0.83
SAM-KNN	0.15	0.60	0.79	0.69	0.66	*0.71*	-
VFDR	0.19	0.62	0.76	0.69	0.66	*0.71*	0.81
RSLVQ	*0.29*	0.65	*0.72*	0.69	0.67	*0.71*	-
OBC	0.21	0.64	0.77	0.71	0.70	0.72	0.80
ARF	0.17	0.68	0.80	0.74	0.72	0.75	0.85
NN-UN	**0.11**	0.58	0.80	0.69	0.65	*0.71*	0.81

Algorithm	W-Precision	W-Recall	W-F1	W-AUC	Kappa	Matthews
CDDRL-Original	**0.95**	0.94	0.94	**0.96**	**0.94**	**0.94**
CDDRL-Ensemble-Methods	0.94	0.94	0.94	0.95	0.92	0.92
CDDRL-Ensemble-BNs	**0.95**	**0.95**	**0.95**	**0.96**	**0.94**	**0.94**
KNN-ADWIN	0.74	0.74	0.73	0.87	0.61	0.62
AWE	*0.47*	0.62	0.51	-	0.42	0.50
DWM	*0.47*	0.60	0.50	-	0.39	0.47
LNSE	0.94	0.94	0.94	**0.96**	0.93	0.93
SRP	0.90	0.89	0.89	0.93	0.85	0.85
SAM-KNN	0.70	0.70	0.68	-	0.54	0.55
VFDR	0.66	0.55	*0.49*	*0.71*	0.32	0.37
RSLVQ	0.53	*0.54*	0.52	-	*0.30*	*0.31*
OBC	0.79	0.79	0.79	0.90	0.68	0.69
ARF	0.92	0.91	0.91	0.95	0.89	0.89
NN-UN	0.88	0.85	0.84	0.91	0.78	0.79

Algorithm	FA	F1-Pos	F1-Neg	F1-Avg	G-mean	B-Accuracy	AUC
CDDRL-Original	0.37	0.52	0.57	0.54	0.46	0.59	0.61
CDDRL-Ensemble-Methods	0.39	0.57	0.55	0.56	0.52	0.60	0.63
CDDRL-Ensemble-BNs	0.29	0.45	0.57	0.51	0.46	0.57	0.57
KNN-ADWIN	0.48	0.49	0.49	*0.49*	0.49	*0.54*	0.55
AWE	0.27	0.55	**0.66**	**0.60**	0.57	0.62	-
DWM	0.47	0.58	0.55	0.56	0.55	0.59	-
LNSE	0.36	*0.43*	0.55	*0.49*	0.46	*0.54*	*0.54*
SRP	0.49	0.58	0.48	0.53	0.46	0.58	0.61
SAM-KNN	0.42	0.49	0.53	0.51	0.41	0.57	-
VFDR	*0.59*	**0.67**	0.50	0.59	**0.58**	**0.63**	**0.66**
RSLVQ	0.38	0.44	0.54	0.50	0.45	0.55	-
OBC	0.50	0.53	0.51	0.52	0.54	0.55	0.55
ARF	0.49	0.54	*0.47*	0.50	*0.40*	0.56	0.55
NN-UN	**0.22**	0.46	0.65	0.55	0.53	0.60	0.61

Algorithm	W-Precision	W-Recall	W-F1	W-AUC	Kappa	Matthews
CDDRL-Original	0.83	0.82	0.82	0.94	0.71	0.71
CDDRL-Ensemble-Methods	**0.85**	**0.84**	**0.84**	**0.95**	**0.75**	**0.76**
CDDRL-Ensemble-BNs	0.82	0.79	0.80	0.92	0.68	0.68
KNN-ADWIN	0.81	0.81	0.80	0.93	0.69	0.69
AWE	0.60	0.64	0.58	-	0.42	0.45
DWM	*0.57*	*0.60*	*0.55*	-	*0.38*	0.41
LNSE	0.81	0.81	0.81	0.94	0.70	0.70
SRP	0.75	0.75	0.74	0.89	0.60	0.61
SAM-KNN	0.82	0.82	0.81	-	0.71	0.71
VFDR	0.60	0.61	0.56	*0.78*	*0.38*	*0.40*
RSLVQ	0.63	*0.60*	0.59	-	0.39	0.41
OBC	0.80	0.79	0.79	0.92	0.67	0.67
ARF	**0.85**	**0.84**	**0.84**	**0.95**	**0.75**	0.75
NN-UN	0.78	0.74	0.72	0.90	0.58	0.59

Table 11: Results of real-life datasets; (top-left) Electricity, (top-right) AWS Spot Price, (bottom-left) Usenet and (bottom-right) NYC Subway Traffic. In bold is the best performing algorithm for the specific metric and in italic + underscore is the worst performing one. Cells are colored in a green-red color range, where the darkest green and darkest red fill the best and worst performing algorithms respectively, separately for each metric.

Algorithm	FA	F1-Pos	F1-Neg	F1-Avg	G-mean	B-Accuracy	AUC
CDDRL-Original	*0.06*	0.88	0.95	0.92	**0.92**	**0.92**	0.95
CDDRL-Ensemble-Methods	**0**	0.88	**0.97**	**0.93**	0.90	0.91	0.95
CDDRL-Ensemble-BNs	**0**	0.84	0.96	0.90	0.86	0.88	0.95
KNN-ADWIN	0.02	0.87	0.96	0.92	0.89	0.90	0.96
AWE	0.02	0.88	**0.97**	**0.93**	0.91	**0.92**	-
DWM	0.01	0.77	0.95	0.86	0.79	0.85	-
LNSE	0.02	**0.90**	**0.97**	**0.93**	**0.92**	**0.92**	0.96
SRP	**0**	0.86	**0.97**	0.91	0.88	0.90	0.97
SAM-KNN	**0**	*0.58*	0.93	*0.76*	*0.59*	*0.77*	-
VFDR	0.02	**0.90**	**0.97**	**0.93**	0.91	**0.92**	0.97
RSLVQ	**0**	0.76	0.95	0.86	0.79	0.85	-
OBC	**0**	0.83	**0.97**	0.90	0.85	0.90	**0.98**
ARF	**0**	0.82	0.96	0.89	0.84	0.88	0.97
NN-UN	0.02	*0.47*	*0.90*	*0.69*	*0.55*	*0.70*	*0.86*

Algorithm	W-Precision	W-Recall	W-F1	W-AUC	Kappa	Matthews
CDDRL-Original	0.83	0.85	0.81	0.96	0.80	0.82
CDDRL-Ensemble-Methods	0.82	0.84	0.80	0.89	0.79	0.81
CDDRL-Ensemble-BNs	0.79	0.84	0.80	0.90	0.79	0.82
KNN-ADWIN	0.81	0.81	0.79	0.92	0.75	0.76
AWE	0.68	0.74	0.69	-	0.66	0.68
DWM	0.82	0.81	0.77	-	0.74	0.77
LNSE	0.81	0.87	0.83	0.96	0.82	0.84
SRP	0.82	0.88	0.84	0.93	0.84	0.86
SAM-KNN	0.76	0.77	0.74	-	0.69	0.71
VFDR	*0.43*	*0.54*	*0.45*	*0.72*	*0.39*	*0.43*
RSLVQ	0.59	0.57	0.50	-	0.43	0.46
OBC	0.81	0.80	0.78	0.92	0.74	0.75
ARF	**0.91**	**0.90**	**0.87**	0.96	**0.86**	**0.88**
NN-UN	0.89	0.89	**0.87**	**0.98**	0.85	0.86

Algorithm	FA	F1-Pos	F1-Neg	F1-Avg	G-mean	B-Accuracy	AUC
CDDRL-Original	0.17	0.49	0.84	0.66	0.62	**0.69**	0.74
CDDRL-Ensemble-Methods	0.17	0.48	0.83	0.65	0.61	0.68	0.75
CDDRL-Ensemble-BNs	0.17	**0.5**	0.84	**0.67**	**0.63**	**0.69**	**0.76**
KNN-ADWIN	0.12	0.45	0.85	0.65	0.57	0.65	0.7
AWE	0.11	0.38	0.84	0.61	0.47	0.63	-
DWM	0.23	0.49	0.79	0.64	0.61	0.66	-
LNSE	0.19	0.43	0.81	0.62	0.57	0.63	0.72
SRP	0.19	0.49	0.79	0.64	0.57	0.66	**0.76**
SAM-KNN	0.06	0.4	0.86	0.63	0.48	0.64	-
VFDR	0.2	0.47	0.81	0.64	0.58	0.67	0.71
RSLVQ	*0.46*	0.48	*0.58*	*0.53*	0.53	0.63	-
OBC	0.39	0.48	0.66	0.57	0.56	0.64	0.71
ARF	0.31	0.47	0.69	0.58	0.53	0.64	0.72
NN-UN	**0.02**	*0.23*	**0.87**	0.55	*0.32*	*0.57*	*0.65*

Table 12: Results of precision agriculture experiments; (top-left) Fertilizer Stress, (top-right) Fertilizer + Water Stress and (bottom-middle) Water Stress

Generally speaking, it can be observed that in most of the experiments all three versions of the CDDRL are at the top half of the algorithms in terms of general performance

(they are colored in green shades), and there is at least one version that is the best performing algorithm. This is true for the *AWS Spot Price* experiment where the original version and the *Ensemble BNs* version are superior to all other competitors, the *Ensemble Methods* version shows the best performance in the *NYC Subway Traffic* experiment, the original version and the *Ensemble Methods* are among the best performers in the *Fertilizer Stress* experiment together with the *LNSE, AWE* and *VFDR* algorithms, and the *Ensemble BNs* version is superior to all others in the *Water Stress* experiment. Regarding the *Electricity* experiment, all algorithms present similar results where the original and the *Ensemble Methods* versions have the worst results, however the *Ensemble BNs* version shows the best results. Regarding the *Usenet* and the *Fertilizer + Water Stress* experiments, all three CDDRL versions show average results compared to all competitors - *AWE, DWM, VFDR* and *NN-UN* are superior in the *Usenet* experiment and *ARF, NN-UN, LNSE* and *SRP* are superior in the *Fertilizer + Water Stress* experiment.

The *Usenet* experiment is the only one we know the behavior of the CD in it (*real CD*). The better performance of the *Ensemble Methods* over the *Ensemble BNs* in it, might be another proof to our earlier statement that the former is more adequate to *real CD* scenarios than the latter.

4.4.5 Statistical Analysis

Since we have seven experiments with six or seven performance metrics in each, it is hard to conclude on a reliable overall ranking between the algorithms. To deal with this problem, we performed the non-parametric Friedman test (M. Friedman, 1937, 1940) and the post-hoc Nemenyi test (Nemenyi, 1963) as suggested by (Demšar, 2006). The tests are based on ranks of the compared elements (algorithms) for each experiment's metric separately (where the best performance is ranked as 1) and searched for distances between the average ranks of the elements that are larger than a critical distance (***CD***) determined by the significance level α (set to 0.05), the number of datasets (*N*), and the number of compared algorithms (*K*). The original Friedman test compares a single performance metric of different algorithms over several datasets and attaches an overall rank to each algorithm.

Having no single metric that is shared between the binary tasks and the multiclass tasks and willing to compare the algorithms' performances in several aspects simultaneously i.e., false alarm rate as well as precision capabilities, we altered the Friedman test. Instead

of using a single metric for each dataset, we used and ranked the algorithms with all measured metrics in every dataset. We excluded the AUC metric since it could not be calculated for several algorithms we tested. To the best of our knowledge, the Friedman test was never used in this kind of manner, i.e., comparing different metrics of several datasets.

In cases where the Friedman tests were significant, we continued with Nemenyi post-hoc tests comparing all algorithms to each other, allowing us to reveal their hierarchy. As suggested by Demšar, the results of the tests are presented visually in Figure 19Figure 19: Friedman-Nemenyi test with significance level of 0.05. The critical distance (CD) in this case is 2.991.. The diagram presents the average rank obtained from the Friedman test for all 14 algorithms. A horizontal line that connects algorithms indicates that they are statistically equivalent. Note that for $\alpha = 0.05,\ N = 44,\ K = 14$, the ***CD*** is 2.991—that is, a gap of more than 2.991 in the average ranks is statistically significant.

Reviewing the diagram, we can observe that the top three ranked algorithms are the three versions of the CDDRL. The *Ensemble BNs* CDDRL version is the highest ranked algorithm and is significantly superior to the *NN-UN, DWM, OBC, VFDR, SAM-KNN* and *RSLVQ* algorithms. The *Ensemble Methods* (ranked second overall) and the original (ranked third overall) CDDRL versions show significant superiority over the same group of algorithms except for the *NN-UN* algorithm. The *SRP* algorithm (highest ranked algorithm after the CDDRL versions) is significantly better than the *VFDR, SAM-KNN* and *RSLVQ* algorithms. The middle-ranked algorithms i.e., *ARF, LNSE, KNN-ADWIN, AWE* and *NN-UN* algorithms performed significantly better only compared to the *RSLVQ* algorithm. Finally, the bottom ranked algorithms i.e., *DWM, OBC, VFDR, SAM-KNN* and *RSLVQ* are not significantly superior to any other algorithm.

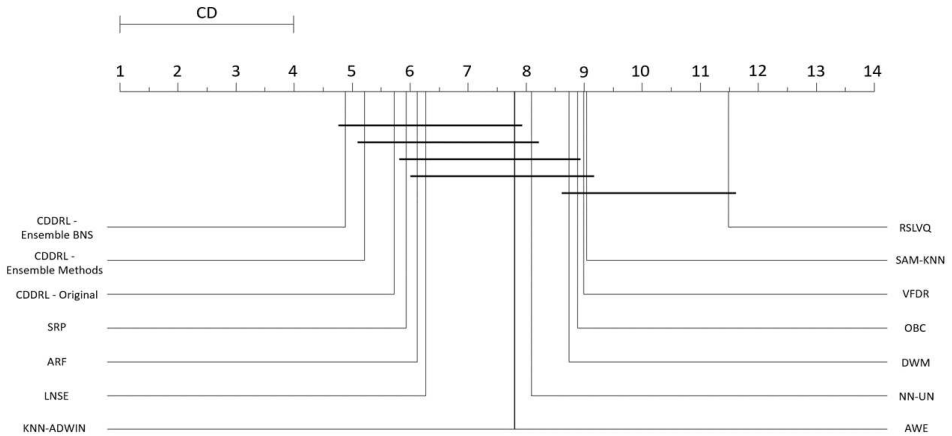


Figure 19: Friedman-Nemenyi test with significance level of 0.05. The critical distance (CD) in this case is 2.991. The bold horizontal lines indicate statistical equivalence between the algorithms covered by it.

4.5 Knowledge Representation - Water Stress

The CDDRL possess an ability no other algorithm discussed in this work does. That ability is to graphically visualize a process allowing us to better understand and explain the development of processes and drifts through time. This is done by exploring the BNs that the CDDRL has learned at different time steps during the data stream - making the CDDRL an adequate knowledge representation algorithm as well. We show an example of this ability in Figure 20. The figure shows four learned BNs by the CDDRL during the Water Stress experiment. The blue 'Treatment' node is our target variable, and the green colored nodes are those that are included in the target variable's Markov blanket (MB) - having an effect on inference. Looking at these BNs, we can explain and perhaps provide better treatment for a plant undergoing water stress. BN_1 shows that the treatment the plants received doesn't affect any other plants' characteristic (no green nodes), this is logical since BN_1 was learned at the first time step - during the stable period - when all plants received the same treatment. However, BN_{15} shows that the treatment (water stress) affects the average third bottom temperature of the plants' leaves - again, logical, since water stressed plants close their stomata causing them to warm up. Looking at BN_{26}, we can observe that the temperature of the plants is no longer affected (the stressed plants partially adapt to the stress) and now the average plants' perimeter as well as the average Hue (color measure) and the number of leaves it has are affected - the leaves of stressed plants curl, shrink and change their color and in general, the stressed plant has a difficult time growing new leaves. Finally, BN_{38} shows that at the end of the process, the same categories of plants' characteristics are affected (size, color, and number of leaves), but some of them are shown via different measures i.e., maximum parameter and area of the plants' leaves (different size measures) and the average VARI which is a different color measure. A summary of plants biology and response to water stress can be found in (Osakabe et al., 2014).

The same thought process can be applied to different processes in different domains. For example, in a medical setting, observing BNs learned over time by the CDDRL, we can better understand and monitor changes in a patient's physical condition when transforming from a healthy state to illness.

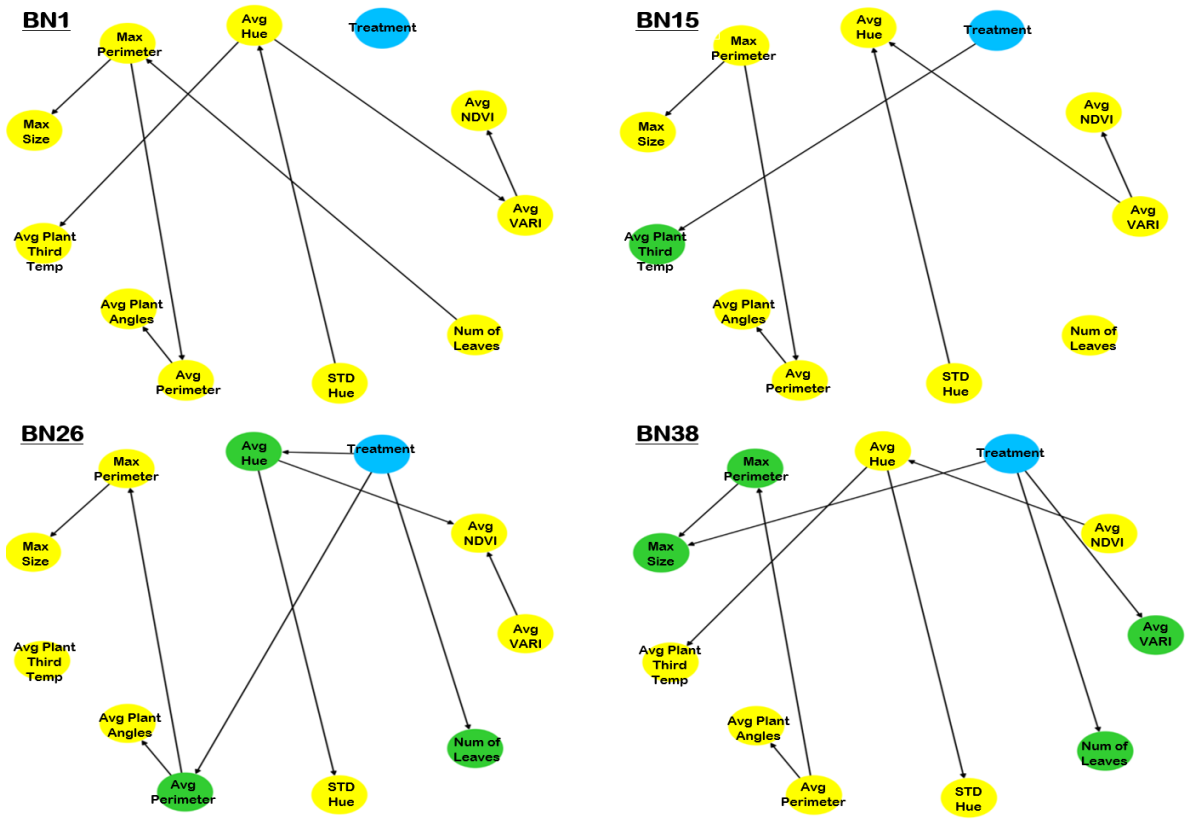


Figure 20: BNs learned by the CDDRL's ensemble methods version during the water stress experiment. The blue 'Treatment' node is the target variable, colored in green are nodes included in the target variable's markov blanket.

4.6 Summary

To sum up this section, we conducted 7 CD experiments using real-world datasets and tested all three CDDRL versions against 11 competing CD algorithms, having several evaluation metrics for each dataset. We obtained an overall rank for each algorithm using the altered Friedman-Nemenyi test and found that the three CDDRL versions are ranked at the very top, significantly superior over around half of the competing algorithms.

The CDDRL's ensemble versions were successful in elevating the predictive power of the algorithm, as proven by the results of the synthetic and real-world experiments. We note again that the current results point out that the *Ensemble Methods* version is more successful under *real CD* settings, whereas the *Ensemble BNs* version is more successful under *virtual CD* settings. Runtime computations of these experiments will be discussed in the following section.

Finally, we showed a unique capability of the CDDRL - a graphic visualization (BNs) of developing relationships between variables in the domain, explaining in our case, the process of a plant going into water stress.

5 Complexity and Runtime Analysis

For the computational complexity analysis, we will denote the following: *N* - is the number of nodes. NS_{max} - is the maximal cardinality (number of categories a variable has). C_{nodes} - are the nodes that were detected as drifted by the CDDRL in the current time step. P_{node_i} - is the number of parents node i has. P_{max} - is the maximal number of parents a node has. $Iter$ - is the number of iterations done by the simulated annealing during the search process.

The first stage of the CDDRL is to compute the distance of each node's current conditional probabilities from the corresponding ones in the previous time step. To do that, we need to compute the distance of each row in each node's CPT. The number of such distance computations per node is equal to the number of the node's parents' configurations. In the worst case, all parents of a node have NS_{max} categories - giving us a computational complexity of $O((NS_{max})^{P_{node\_i}})$ for a single node and $O(N * (NS_{max})^{P_{max}})$ for all nodes. We note that at the worst case, a single node may have all other nodes as its parents, however we cannot have another node such as this because then cycles will be present, and we therefore keep P_{max} and not $N - 1$. Aggregating each node's CPD distances to one distance is negligible compared to the distance computations themselves.

At the second part of the CDDRL, we build and score possible graphs around the previous graph using C_{nodes}; we do this $Iter$ times. Each of C_{nodes} can have at maximum two graphs generated by its interaction with each of the other nodes. If a node was not a parent of the specific C_{nodes}, then it can be added as its parent, and if that node was previously a parent, it can either be removed as a parent or the edge can be reversed in case that the node is also in C_{nodes}. Therefore, the computational complexity of this part in the worst case is $O(Iter * |C_{nodes}| * 2 * (N - 1))$. In the worst case – where all nodes are in C_{nodes} – we get $O(Iter * N * 2 * (N - 1))$ which is $O(Iter * N^2)$. To the best of our knowledge and as stated in the original simulated annealing paper (Url et al., 2007), there is no accurate complexity analysis of the simulated annealing process, but rather most related works concentrate on finding an optimal cooling schedule (Jung et al., 1998; Tsitsiklis & Bertsimas, 1993). Therefore, we do not have an upper bound on $Iter$ and since it can be manually limited (though not yet implemented in the current CDDRL versions), we disregard the complexity of $Iter$ and state that the complexity of the first CDDRL stage will most likely

overshadow the complexity of the second stage, giving us a CDDRL complexity of $O(N * (NS_{max})^{P_{max}})$.

Algorithm/Experiment	Sin	Stagger 500	Stagger 5000	Electricity	AWS Price	Usenet	NYC Subway	Fertilizer	Fertilizer + Water	Water
CDDRL-Original	6.3	7.6	8.3	18.0	583.2	1399.5	256.2	83.7	6.4	8.2
CDDRL-Ensemble-Methods	21.3	78.2	87.4	327.9	6353.8	*7958.6*	713.6	432.0	38.4	53.6
CDDRL-Ensemble-BNs	6.5	8.8	9.1	19.8	5549.5	5308.1	594.5	285.4	35.2	8.3
KNN-ADWIN	30.8	14.3	14.1	73.5	1727.7	36.9	1410.3	53.1	4.4	12.4
AWE	20.8	9.6	8.9	66.6	819.1	9.4	235.0	15.7	1.7	8.6
DWM	10.4	2.0	3.1	23.4	330.2	3.3	115.7	**2.0**	0.8	2.2
LNSE	34.6	23.8	20.0	61.6	1446.1	3.7	738.6	25.0	8.5	6.8
SRP	27.9	13.9	13.4	196.0	2646.4	35.9	503.4	45.0	10.7	32.2
SAM-KNN	10.3	3.1	2.8	27.5	450.1	**1.0**	58.4	9.8	2.2	4.4
VFDR	2.8	**1.4**	**1.3**	**4.7**	**85.1**	3.7	57.0	2.5	**0.7**	**0.7**
RSLVQ	**1.1**	7.6	7.4	31.7	305.7	1.4	**3.6**	7.0	3.9	5.2
OBC	*96.7*	42.7	48.9	*761.7*	*10184.8*	307.1	*4794.8*	*691.9*	*61.6*	*430.6*
ARF	10.8	20.5	19.8	248.1	6561.2	12.0	137.1	46.6	15.9	32.2
NN-UN	22.6	*98.2*	*95.1*	10.4	2539.5	37.0	832.7	83.1	42.3	4.7

Table 13: Run times of the 3 synthetic and 7 real-world experiments. Best algorithm in each experiment is colored in the darkest green and is in bold, whereas the worst is colored in darkest red and is italic + underscored

Experiment	Num of Nodes	Avg. Parents	Max Parents	Ratio Avg. Parents	Ratio Max Parents
Electricity	7	1.06	1.94	0.15	0.28
AWS Spot Price	6	0.77	2.00	0.13	0.33
Usenet	99	1.06	9.93	0.01	0.10
NYC Subway Traffic	17	2.35	7.68	0.14	0.45
Fertilizer Stress	10	0.96	1.89	0.10	0.19
Fertilizer + Water Stress	4	0.44	1.19	0.11	0.30
Water Stress	10	0.73	1.51	0.07	0.15
Avg. Experiments	**21.86**	**1.05**	**3.73**	**0.05**	**0.17**

Table 14: Aggregated number of parents in each experiment. The ratio of average number of nodes out of total number of nodes, as well as the ratio of maximum number of nodes out of the total number of nodes are given.

In case some nodes have many parents, then the CDDRL might have a long run time (see CDDRL's run time in the *Usenet* experiment shown in Table 13). However, the user can drastically reduce complexity by limiting the maximal number of parents a node can get (as done in the very well-known PC algorithm (Spirtes et al., 2000)), or reduce variable cardinality by, e.g., joining categories especially rare. Since P_{max} and NS_{max} can be controlled by the user, in a rare case, where the number of nodes (N) is extremely large (something the user cannot control), the complexity of the second CDDRL stage will dominant. Therefore, we keep the complexity of the second stage and state that in most realistic cases the first stage will dominant, though through user's restrictions, the complexity of the second stage might dominate. The overall complexity of the CDDRL is then:

$$O(N^2 + N * (NS_{max})^{P_{max}})$$

Table 14 shows for each experiment the number of nodes, the average number of parents a node has (extracted from the BNs the original CDDRL has learned), the average (over all datasets) maximal number of parents, and the ratios of the average and maximal (over all datasets) number of parents out of the total number of nodes. It's clear to see that the relatively high number of maximum parents in the *Usenet* and *NYC Subway Traffic* experiments is the cause for the CDDRL's long run time. Looking at the averages over all experiments, we can observe that the average number of parents is slightly above 1, where the average of the maximal number of parents (perhaps most important) is 3.73. In addition, we can see that the average parents ratio is just 5% and the maximum ratio is 17%, meaning that empirically – based on a number of CD-representing datasets – we can state that the complexity of the CDDRL in the average case is $O(N^2 + N * (NS_{max})^{0.05N})$ and in the maximal \ worst case is $O(N^2 + N * (NS_{max})^{0.17N})$ and we can get a better understanding of how limiting the maximal number of parents a node has will affect the CDDRL's run time.

In general, it can be observed in Table 13 that the original version of the CDDRL is not in the bottom half of the algorithms in terms of run time (no red cells), except for the *Usenet* experiment, where we had nodes with many parents, causing the CDDRL to run for a long time. The ensemble methods as expected have more predictive power, but they also come with a longer run time because they deal with several methods/BNs at every time step. Developing quicker ensemble versions is left for future work.

6 Discussion & Conclusions

Stream learning is not a trivial task, especially in the presence of CD. CD can be either *real* or *virtual*, and both can appear in several scenarios (e.g., *sudden* and *gradual*). To address CD in all its forms, the CDDRL (Mendelson, 2020) - a concept drift detection and re-learning algorithm - continually tests each node's CPT by an SPC test, detects which nodes have experienced a statistical change, and uses a local structure learning updating algorithm to adjust the BN for these changes in an efficient manner. The correctness of the CDDRL is mathematically proven.

In earlier works (Cohen, 2018; Mendelson, 2020; Nahum, 2015), the CDDRL's detection performance was found robust for all CD types and for different data characteristics, e.g., dimensionality and skewness - mainly using synthetic data.

In this thesis, we introduced several developments to the original CDDRL, aiming to improve the process of the CDDRL and its accuracy. Specifically, an alternative to the S&S process was proposed. The new proposed constraint-based method was not able to outperform the original version in structure learning experiments but has proven to be a quicker version of it.

In addition, four other developments were introduced, each modifying a minor process of the CDDRL, i.e., learning dynamic (instead of constant) UCLs, applying a tabu list to the S&S process, expanding the S&S process and giving weights to past CPTs whilst learning the current CPTs (*Params*). The minor developments did not significantly improve the CDDRL, however did not harm it, and in general, it was observed that the *Params* version was the most successful, outperforming the original version in most cases.

Then, two ensemble-based versions of the CDDRL were introduced; the first has the minor developments versions as its base estimators, and the second holds past learned BNs as its base estimators. The ensemble-based versions – which are known for their prediction abilities via reduction of variance – were successful in elevating the CDDRL's accuracy. This statement is supported with a series of experiments with three synthetic and seven real-life datasets (some are publicly available, and some are self-gathered from the agriculture field), in which the original CDDRL and its ensemble versions are assessed against 11 other CD competing algorithms. In general, all CDDRL versions showed very promising results in these experiments, where a new proposed modification of the Friedman-Nemenyi test was used

to obtain overall rankings and statistical significance between the tested algorithms. The three CDDRL versions were ranked at the very top; the *Ensemble BNs* version came in first place, significantly superior to six of the competitors, whereas the *Ensemble Methods* and the original versions that came in second and third place respectively, were significantly superior to five of the competitors. Following some of the presented results (mainly the synthetic experiments), it is carefully stated that the *Ensemble Methods* version better adapts to *real CD*, in contrast to the *Ensemble BNs* version that better adapts to *virtual CD*.

The computational complexity of the CDDRL was analyzed, and optional methods to reduce it were proposed (this was never discussed in previous CDDRL works). Run times of the different CDDRL versions were also evaluated and showed that the original CDDRL has an average performance compared to the competing CD algorithms, and that the ensemble versions have a longer run time, due to the additional actions they perform.

Finally, besides the shown ability of the CDDRL regarding prediction, we showed - using the water stress experiment - that the CDDRL can explain a process, manifesting, naturally, explainable AI.

7 Future Research

I offer future research to focus on the following directions:

1. Constraint-based (CB): The offered method was not successful enough in improving the CDDRL's performance, but it was found quicker than the original CDDRL version, and it holds the potential for becoming a strong causal method for learning a BN structure. V-structure contradiction settlement techniques as offered in (Simchon, 2011) and the usage of a different scoring function (such as MLD) in the last step of the method, can be implemented in order to reduce the wrong direction (WD) error of edges. A changing CI test threshold for different steps of the method – lower for the first steps (see Section 3.1.4.1.5) – should also help in balancing out the missing edge (ME) and the extra edge (EE) errors. Experimentations with the maximum ordering allowed in the network should also be done to find a better suiting method that will improve the general performance of the algorithm.

2. Computational Complexity of the CDDRL: Following the analysis done in this thesis, the most contributing part to the computational and run time complexities of the CDDRL is the number of maximum parents a node in the network has, and the second most contributing part is the dimensionality of the nodes. Therefore, I offer to implement methods that limit the number of parents and categories (dimensionality) a node can have and to evaluate their effect on run time and on general performance of the CDDRL.

3. Ensemble Versions: The two offered ensemble versions in this thesis use a certain approach to update the weights of their base estimators at each time step. This approach multiplies the AUC performance of each base estimator on the current time step with its previous weights and then normalizes all weights with each other (dividing each by the sum of all weights). Different updating techniques using different evaluation metrics could be implemented and assessed. In addition, the number of base estimators (past learned BNs) in the *Ensemble BNs* version should also be explored, since in this work we used a constant number of four networks in all experiments. Other removal and addition techniques of BNs from/to the ensemble in the *Ensemble BNs* version could also be proposed and evaluated.

4. Coding Platform: As of today, the CDDRL is implemented using *Matlab*. The current CDDRL requires old versions of *Matlab* for it to function properly (mainly learning BN_0 with the *MMHC* algorithm is not possible in newer versions of *Matlab*). In addition, *Matlab* is probably a slower platform and is not as open and popular as *Python*. For these reasons, I have started re-writing the CDDRL algorithm in *Python*. However, only the basic CDDRL functions are currently implemented, and empirical results show massive differences between results of the same tasks ran with *Matlab* and with *Python.* Therefore, a careful debugging work needs to be done as well as other code implementations (such as the ensemble versions of the CDDRL).

5. Latent Variables: The CDDRL's performance and abilities can be drastically improved with the combination of latent variable learning methods such as that presented in (Halbersberg & Lerner, 2020), having the potential to elevate the explainability and knowledge representation abilities of the CDDRL.

8 References

Aggarwal, C. C. (2007). An Introduction to Data Streams. *Data Streams*, 1–8. https://doi.org/10.1007/978-0-387-47534-9_1

Amazon Web Services. (n.d.). https://aws.amazon.com/

Baena-García, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavaldà, R., & Morales-Bueno, R. (2006). Early Drift Detection Method. *4th ECML PKDD International Workshop on Knowledge Discovery from Data Streams*, *6*, 77–86. https://doi.org/10.1.1.61.6101

Baier, L., Schlör, T., Schöffer, J., & Kühl, N. (2021). *Detecting Concept Drift With Neural Network Model Uncertainty*. 1–19. http://arxiv.org/abs/2107.01873

Bifet, A., & Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing. *Proceedings of the 7th SIAM International Conference on Data Mining*, 443–448. https://doi.org/10.1137/1.9781611972771.42

Borchani, H., Larrañaga, P., Gama, J., & Bielza, C. (2016). *Mining multi-dimensional concept-drifting data streams using Bayesian network classifiers*. *20*, 257–280. https://doi.org/10.3233/IDA-160804

Buntine, W. (1989). *Theory Refinement on Bayesian Networks*. 52–60.

Chickering, D. M. (2003). Optimal structure identification with greedy search. *Journal of Machine Learning Research*, *3*(3), 507–554. https://doi.org/10.1162/153244303321897717

Cohen, M. (2018). *Concept drift in machine learning - detection and relearning*.

Cooper, G. F., & Herskovits, E. (1992). A Bayesian Method for the Induction of Probabilistic Networks from Data. *Machine Learning*, *9*(4), 309–347. https://doi.org/10.1023/A:1022649401552

Darwiche, A. (2009). *Modeling and reasoning with Bayesian networks*. Cambridge university press.

De Morais, S. R., & Aussem, A. (2010). An efficient and scalable algorithm for local bayesian network structure discovery. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *6323 LNAI*(PART 3), 164–179. https://doi.org/10.1007/978-3-642-15939-8_11

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*, 1–30.

Elwell, R., & Polikar, R. (2011). Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, *22*(10), 1517–1531. https://doi.org/10.1109/TNN.2011.2160459

Espíndola, R. P., & Ebecken, N. F. F. (2005). On extending F-measure and G-mean metrics to multi-class problems. *Data Mining VI*, *1*, 25–34. https://doi.org/10.2495/data050031

Friedman, M. (1937). The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *Journal of the American Statistical Association*, 675–701. https://doi.org/10.1080/01621459.1937.10503522

Friedman, M. (1940). A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, *11*(1), 86–92.

Friedman, N., & Goldszmidt, M. (n.d.). *Sequential Update of Bayesian Network Structure*.

G, C. (1952). The X2 test of goodness of fit. *The Annals of Mathematical Statistics*, 315–345.

Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). Learning with drift detection. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3171*, 286–295. https://doi.org/10.1007/978-3-540-28645-5_29

Gerber, E. (2021). *Nyc subway traffic dataset*. https://www.kaggle.com/datasets/eddeng/nyc-subway-traffic-data-20172021

Glover, F., & Laguna, M. (1998). Tabu Search. *Handbook of Combinatorial Optimization*, 2093–2229. https://doi.org/10.1007/978-1-4613-0303-9_33

Gomes, Heitor M., Bifet, A., Read, J., Barddal, J. P., Enembreck, F., Pfharinger, B., Holmes, G., & Abdessalem, T. (2017). Adaptive random forests for evolving data stream classification. *Machine Learning*, *106*(9–10), 1469–1495. https://doi.org/10.1007/s10994-017-5642-8

Gomes, Heitor Murilo, Read, J., & Bifet, A. (2019). Streaming random patches for evolving data stream classification. *Proceedings - IEEE International Conference on Data Mining, ICDM*, *2019-Novem*(February 2020), 240–249. https://doi.org/10.1109/ICDM.2019.00034

González, A., Martín, I., & Ayerbe, L. (1999). Barley yield in water-stress conditions. *Field Crops Research*, *62*(1), 23–34. https://doi.org/10.1016/s0378-4290(99)00002-7

Grandini, M., Bagli, E., & Visani, G. (2020). *Metrics for Multi-Class Classification: an Overview*. 1–17. http://arxiv.org/abs/2008.05756

Halbersberg, D., & Lerner, B. (2020). *Local to Global Learning of a Latent Dynamic Bayesian Network*. 2600–2607. https://doi.org/10.3233/FAIA200396

Harries, M. (1999). *Splice-2 comparative evaluation: Electricity pricing*.

Heckerman, D. (2020). A Tutorial on Learning With Bayesian Networks. *ArXiv*, *1995*(November). https://doi.org/10.1007/978-94-011-5014-9_11

Heckerman, D., Geiger, D., & Chickering, D. M. (1995). Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. *Machine Learning*, *20*(3), 197–243. https://doi.org/10.1023/A:1022623210503

Heusinger, M., Raab, C., & Schleif, F. M. (2022). Passive concept drift handling via variations of learning vector quantization. *Neural Computing and Applications*, *34*(1), 89–100. https://doi.org/10.1007/s00521-020-05242-6

Huang, S., & Dong, Y. (2006). On mining time-changing data streams. *Chinese Journal of Electronics*, *15*(2), 220–224.

Jian-Kang Zhu. (2016). Abiotic stress signaling and responses in plants. *Cell*, *167*(3), 313–324. https://doi.org/10.1016/j.cell.2016.08.029.Abiotic

Jung, T., Wang, A. A., Monroe, W., Nourani, Y., & Andresen, B. (1998). *A comparison of simulated annealing cooling strategies*.

Katakis, I., Tsoumakas, G., & Vlahavas, I. (2008). An ensemble of classifiers for coping with recurring contexts in data streams. *Frontiers in Artificial Intelligence and Applications*, *178*, 763–764. https://doi.org/10.3233/978-1-58603-891-5-763

Khamassi, I., Sayed-Mouchaweh, M., Hammami, M., & Ghédira, K. (2015). Self-Adaptive Windowing Approach for Handling Complex Concept Drift. *Cognitive Computation*, *7*(6), 772–790. https://doi.org/10.1007/s12559-015-9341-0

KIFER, D., BENDAVID, S., & GEHRKE, J. (2004). Detecting Change in Data Streams. *Proceedings 2004 VLDB Conference*, 180–191. https://doi.org/10.1016/b978-012088469-8/50019-x

Kim, Y., Glenn, D. M., Park, J., Ngugi, H. K., & Lehman, B. L. (2011). Hyperspectral image analysis for water stress detection of apple trees. *Computers and Electronics in Agriculture*, *77*(2), 155–160. https://doi.org/10.1016/j.compag.2011.04.008

Koller, D. (n.d.). *Principles and Techniques*.

Kolter, J. Z., & Maloof, M. A. (2007). Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research*, *8*, 2755–2790.

Kosina, P., & Gama, J. (2015). Very fast decision rules for classification in data streams. *Data Mining and Knowledge Discovery*, *29*(1), 168–202. https://doi.org/10.1007/s10618-013-0340-z

Lam, W., & Bacchus, F. (1994). Learning Bayesian belief networks: An approach based on the MDL principle. *Computational Intelligence*, *10*(3), 269–293.

Lerner, B., Afek, M., & Bojmel, R. (2013). Adaptive thresholding in structure learning of a Bayesian network. *IJCAI International Joint Conference on Artificial Intelligence*, 1458–1464.

Losing, V., Hammer, B., & Wersing, H. (2017). KNN classifier with self adjusting memory for heterogeneous concept drift. *Proceedings - IEEE International Conference on Data Mining, ICDM*, *1*, 291–300. https://doi.org/10.1109/ICDM.2016.141

Louppe, G., & Geurts, P. (2012). *LNAI 7523 - Ensembles on Random Patches*. 346–361.

Lu, N., Zhang, G., & Lu, J. (2014). Concept drift detection via competence models. *Artificial Intelligence*, *209*(1), 11–28. https://doi.org/10.1016/j.artint.2014.01.001

Meek, C. (1995). *1302.4972*. 1–8. http://arxiv.org/pdf/1302.4972.pdf%5Cnpapers2://publication/uuid/1693DBAD-6D93-4665-BFC9-81ED0DB68B98

Mendelson, S. (2020). *Concept-drift detection and re-learning using Bayesian networks*.

Minku, L. L., White, A. P., & Yao, X. (2010). The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering*, *22*(5), 730–742. https://doi.org/10.1109/TKDE.2009.156

Nahum, L. (2015). *Concept-drift detection in Bayesian networks by parameter sequential monitoring*. *March*.

Nemenyi, P. (1963). *University Microfilms, Inc., Ann Arbor, Michigan f $*.

Nielsen, S. H., & Nielsen, T. D. (2008). Adapting Bayes network structures to non-stationary domains. *International Journal of Approximate Reasoning*, *49*(2), 379–397. https://doi.org/10.1016/j.ijar.2008.02.007

Nilsson, H. E. (1995). Remote sensing and image analysis in plant pathology. *Annual Review of Phytopathology*, *33*, 489–527. https://doi.org/10.1146/annurev.phyto.33.1.489

Operators, E., Johnsbraten, H., Miyanishi, M., Takeuchi, M., Louhivaara, I. S., Sathaye, A. M., Marsden, J. E., Dimensional, I., Transformation, L., Thomee, V., Wahlbin, L. B., Ramirez, D. E., Tolimieri, R., Analysis, A. H., Functions, T., Functions, E., & Glaser, L. C. (n.d.). *No Title*.

Osakabe, Y., Osakabe, K., Shinozaki, K., & Tran, L. S. P. (2014). Response of plants to water stress. *Frontiers in Plant Science*, *5*(MAR), 1–8. https://doi.org/10.3389/fpls.2014.00086

Pearl, J. (n.d.). *Models, Reasoning, and Inference*. 1–7.

Pesaranghader, A., Viktor, H. L., & Paquet, E. (2017). McDiarmid drift detection methods for evolving data streams. *ArXiv*, *October*.

Press, W. H., Flannery, B. P., Teukolsky, S. A., Vetterling, W. T., York, N., & Chester, P. (n.d.). *The Art of Scientific Computing*.

Raab, C., Heusinger, M., & Schleif, F. M. (2020). Reactive Soft Prototype Computing for Concept Drift Streams. *Neurocomputing*, *416*, 340–351. https://doi.org/10.1016/j.neucom.2019.11.111

Savary, S., Ficke, A., Aubertot, J. N., & Hollier, C. (2012). Crop losses due to diseases and their implications for global food production losses and food security. *Food Security*, *4*(4), 519–537. https://doi.org/10.1007/s12571-012-0200-5

SciKit Multiflow. (n.d.). https://scikit-multiflow.readthedocs.io/en/stable/api/api.html

Scutari, M. (n.d.). *Bayesian Network Repository*. https://www.bnlearn.com/bnrepository/

Sibomana, I. C., Aguyoh, J. N., & Opiyo, a M. (2013). Water stress affects growth and yield of container grown tomato (Lycopersicon esculentum Mill) plants. *Bangladesh Journal of Agricultural Research*, *2*(4), 461–466.

Simchon, N. (2011). *Selective CB Structure Learning* (Issue February).

Spirtes, P., Glymour, C., & Scheines, R. (2000). 1. Causation, Prediction, and Search. 2nd edn. Peter Spirtes, Clark Glymour and Richard Scheines, MIT Press, Cambridge, MA, 2000. No. of pages: 543. ISBN 0-262-19440-6. *Statistics in Medicine*, *22*(13), 2236–2237. https://doi.org/10.1002/sim.1415

Stanley, K. O. (2003). Learning Concept Drift with a Committee of Decision Trees. *Computer Science Department University of Texas*, *Cdc*, 1–17. http://scholar.google.com/scholar?hl=en&q=Learning+Concept+Drift+with+a+Committee+of+Decision+Trees&btnG=Search&as_sdt=2000&as_ylo=&as_vis=0#0

Street, C. (2003). by Judea Pearl. *Econometric Theory*, *19*, 675–685.

Surendar, K. K., Devi, D. D., Ravi, I., Krishnakumar, S., Kumar, S. R., & Velayudham, K. (2013). Impact of Water Deficit on Photosynthetic Pigments and Yield of Banana Cultivars and Hybrids. *Plant Gene and Trait*, *4*(4), 17–24. https://doi.org/10.5376/pgt.2013.04.0004

Tsamardinos, I., Brown, L. E., & Aliferis, C. F. (2006). The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, *65*(1), 31–78. https://doi.org/10.1007/s10994-006-6889-7

Tsitsiklis, J., & Bertsimas, D. (1993). *Institute of Mathematical Statistics is collaborating with JSTOR to digitize, preserve, and extend access to Statistical Science. ® www.jstor.org*.

Tsymbal, A. (2004). The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*.

Url, S., Archive, T. J., & Archive, T. (2007). *Optimization by simulated annealing*. *220*(4598), 671–680.

Verma, T., & Pearl, J. (1990). Causal Networks: Semantics and Expressiveness. *Machine Intelligence and Pattern Recognition*, *9*(C), 69–76. https://doi.org/10.1016/B978-0-444-88650-7.50011-1

Visser, B. (2017). *AWS Spot Pricing*. https://www.kaggle.com/datasets/noqcks/aws-spot-pricing-market

Wang, B., & Pineau, J. (2016). Online Bagging and Boosting for Imbalanced Data Streams. *IEEE Transactions on Knowledge and Data Engineering*, *28*(12), 3353–3366. https://doi.org/10.1109/TKDE.2016.2609424

Webb, G. I., Lee, L. K., Goethals, B., & Petitjean, F. (2017). Understanding concept drift. *ArXiv*, *June*.

Yasin, A. (2013). *Incremental Bayesian network structure learning from data streams*. *1*, 191.

Yehezkel, R., & Lerner, B. (2009). Bayesian network structure learning by recursive autonomy identification. *Journal of Machine Learning Research*, *10*, 1527–1570.

Žliobaitė, I. (2010). *Learning under Concept Drift: an Overview*. 1–36. http://arxiv.org/abs/1010.4784

9 Appendices

9.1 Appendix A – BNs Used in CB Structural Experiments

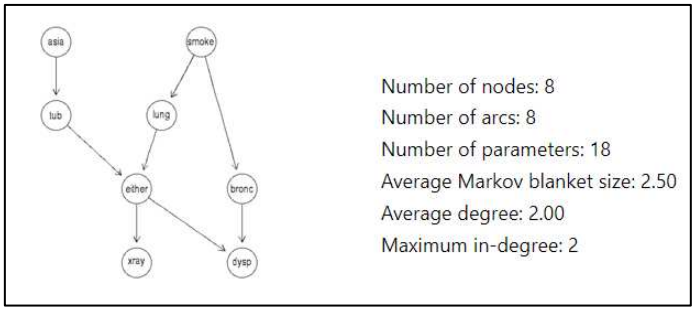


Figure 21: Asia BN and its characteristics

The Asia experiment was divided into three experiments. The first had one added edge from 'SMOKE' to 'TUB', the second experiment was an edge removal from 'BRONC' to 'DYSP' and the third was an edge reversal between 'BRONC' and 'DYSP'. Results of the three experiments were aggregated and reported as one.

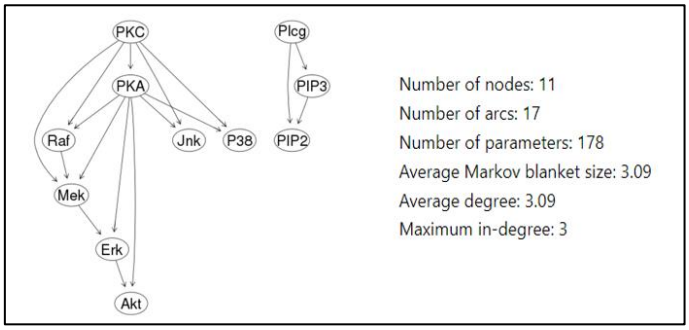


Figure 22: Sachs BN and its characteristics

During the Sachs experiment four nodes were changed; 'Jnk' was added as a parent to 'Akt', 'Raf' was removed as a parent of 'Mek' and the edge between 'PKC' and 'PKA' was reversed. The 'changed' BN (which the algorithms were trying to discover) included all changes at once.

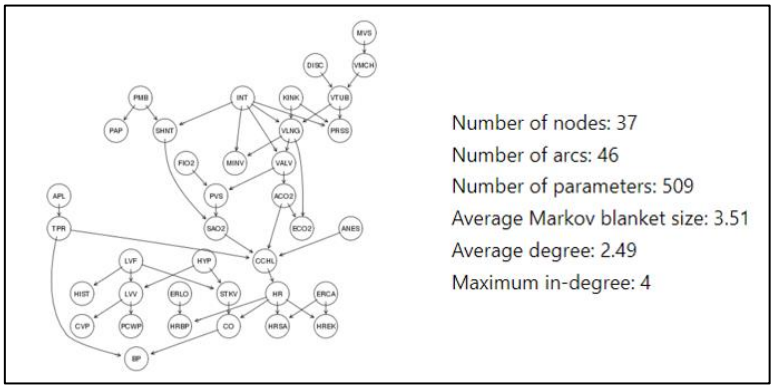


Figure 23: Alarm BN and its characteristics

During the Alarm experiment eight nodes were changed; 'PMB' was added as a parent to 'APL' and 'FIO2' was added as a parent to 'SAO2'. 'TPR' was removed as a parent of 'BP' and 'INT' was removed as a parent of 'PRESS'. The edge between 'ERLO' and 'HRBP' and the edge between 'STKV' and 'CO' were reversed. The 'changed' BN included all changes at once.

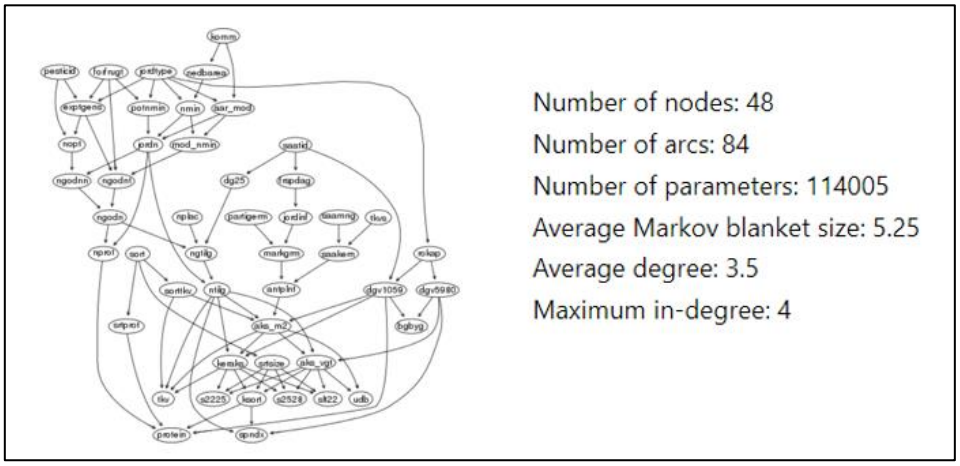


Figure 24: Barley BN and its characteristics

During the Barley experiment 12 nodes were changed; 'pesticid' was added as a parent to 'nedbarea', 'dg25' was added as a parent to 'nplac' and 'partigerm' was added as a parent to 'dgv5980'. 'jordtype' was removed as a parent of 'potmin', 'ngodn' was removed as a parent of 'ngtilg' and 'aks_vgt' was removed as a parent of 'ksort'. The edge between 'rokap' and 'dgv1059', the edge between 'frspdag' and 'jordinf' and the edge between 'srtsize' and 's2225' were reversed. The 'changed' BN included all changes at once.

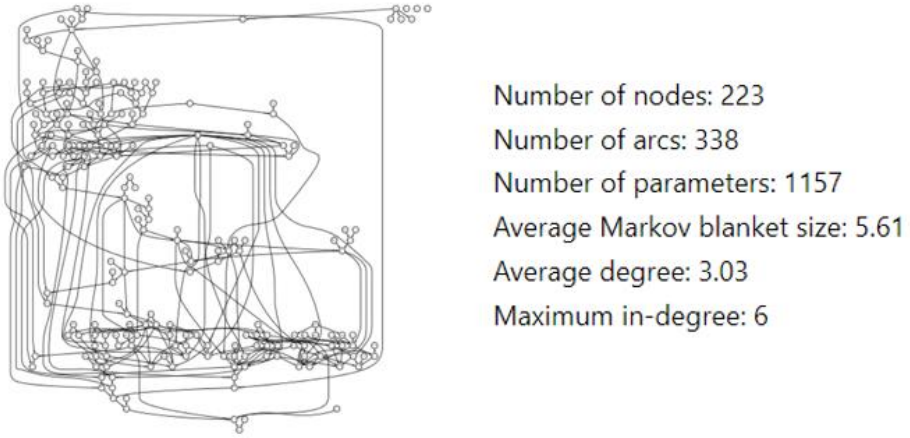


Figure 25: Andes BN and its characteristics

During the Andes experiment 16 nodes were changed; 'GIVEN_1' was added as a parent to 'Rapp1', 'VECTOR27' was added as a parent to 'GOAL_56', 'Rapp6' was added as a parent to 'GOAL_79' and 'GOAL68' was added as a parent to 'SNode_112'. 'TRY11' was removed as a parent of 'GOAL_49', 'KINE29' was removed as a parent of 'SNode_74', 'WEIGHT57' was removed as a parent of 'SNode_95' and 'SUM75' was removed as a parent of 'SNode_125'. The edge between 'TRY12' and 'TRY13', the edge between 'SNode_41' and 'SNode_42', the edge between 'NORMAL52' and 'INCLINE51' and the edge between 'GOAL_143' and 'Rapp12' were reversed. The 'changed' BN included all changes at once.

9.1.1 SHD Parts Analysis

In this section I broke down the SHD mistakes our algorithm has done into its three parts – missing edge (ME), extra edge (EE) and wrong direction (WD). The analysis was done with respect to the final SHD and also calculated after every part of the algorithm (not for all networks). The analysis was done so we could better understand what are and where are our main failures:

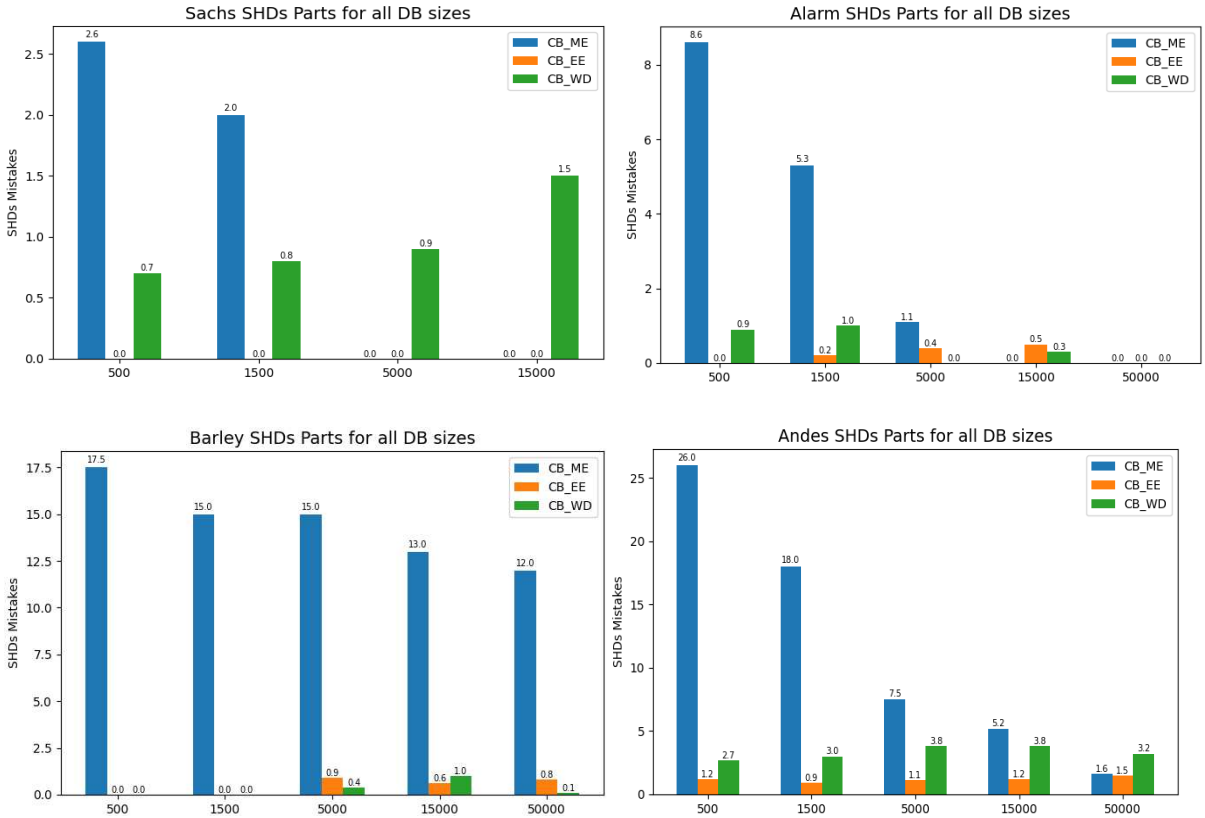
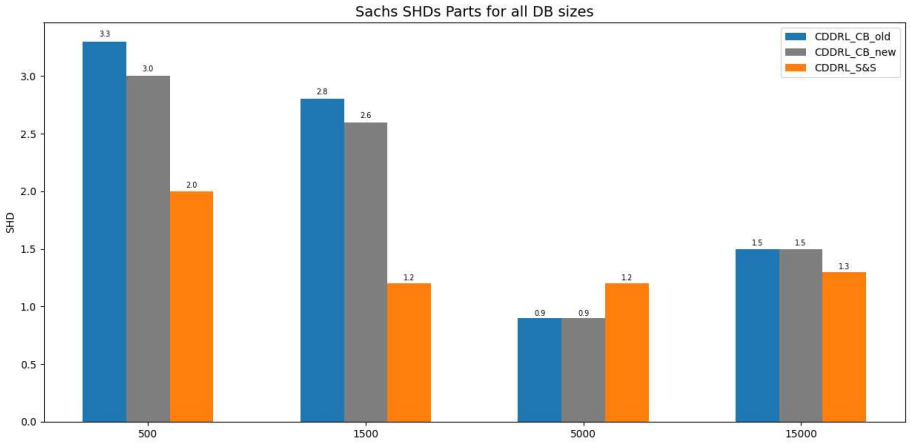
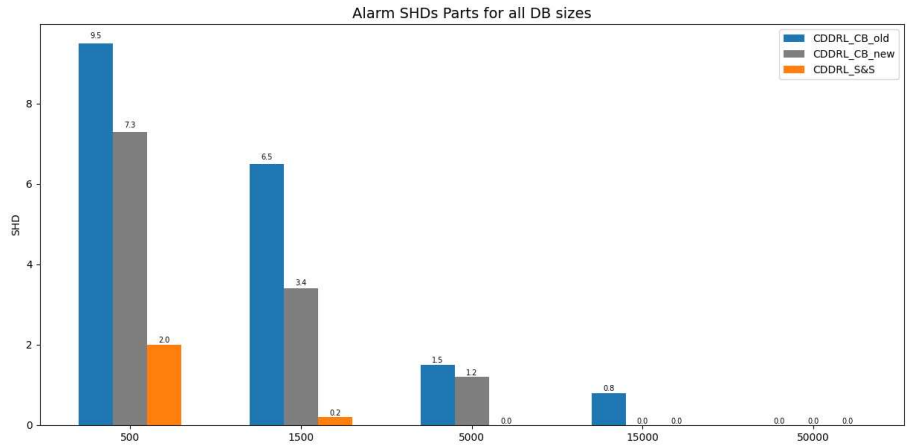


Figure 26: SHDs mistakes broken into ME, EE and WD for the Sachs, Alarm, Barley and Andes networks.

It is very easy to observe that our main contributor to the SHD measure is the missing edge (ME) error. It is also noticeable that ME decreases with the increase of the dataset's size and in some cases, there is a mild increase in the EE and ED errors. This analysis led me to the conclusion that we have a problem with our threshold. I looked for a threshold that would start smaller than our current threshold and will get bigger than our current threshold with the increase of the dataset's size. This kind of threshold will supposedly have a better balance of ME and EE errors. I came up with a change to the current threshold's constant – replacing $2 * N\ (size\ of\ the\ dataset)$ by $80 * sqrt(N)$. This was achieved through trial and error and was successful in improving some of the results, however a much more thorough job needs to be done on this matter. This threshold was tested on the 'Sachs', 'Alarm' and 'Barley' networks:





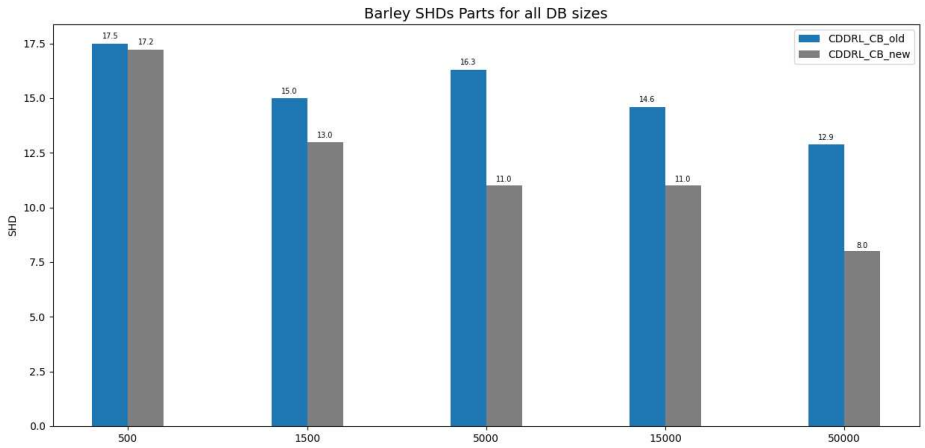


Figure 27: SHD comparison after threshold changing - old and new thresholds and the S&S version results for Sachs and Alarm are at the top and middle graph. Old and new thresholds Barley results are at the bottom graph.

9.2 Appendix B – Water Stress Experiment Process

9.2.1 Experimental Design

We took 192 pre-mature banana plants and randomly divided them into four separate groups: each consisting of 48 plants. The plants were mixed and located separately inside a greenhouse. The plants were grown in these settings for 41 days; in the first 13 days (the control period) all plants got treated with the same amount of water – 100% of the recommended amount. Starting from day 14 and forward each group of plants received a different amount of water treatment as follows:

- Group A received 100% of the recommended amount.
- Group B received 80% of the recommended amount.
- Group C received 60% of the recommended amount.
- Group D received 40% of the recommended amount and was counted for the 'stressed group'.

Each plant in each of the 41 days was photographed from above using several cameras (RGB, multispectral, thermal, depth) placed on a moving cart. At the end of the growing period, all photos were collected, and features of geometry, color, temperature and depth were extracted and used to build a time-series model that predicts for each day which plants are from group D (stressed plants) using data from the previous days.

9.2.2 Photographing

As mentioned above, the banana plants were photographed from above every day by several cameras all placed on a moving cart. Cameras of different types were used in order to evaluate different aspects of the plants e.g., color, size, radiation. The cameras thar were used:

RGB camera: the RGB camera was the most contributing camera. Each pixel in the photo created by the RGB camera consists of three numbers remarking how 'red', 'green' and 'blue' the pixel is. These numbers were later used to extract color indices for every plant. The RGB photos were also used to extract geometrical indices.

Figure 28 : An example of a photo taken by the RGB camera

Depth camera: the depth camera is similar to the RGB, only it can be used to compute the depth of the object it is photographing, meaning how far the object is from the camera. This measure helped us to compute the height of each plant, which is important since plants under water stress experience a growing defect that can effect its height.

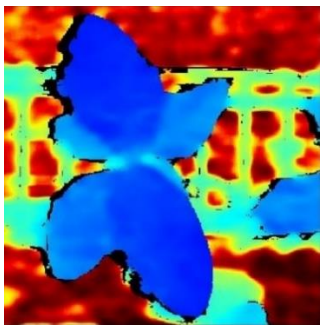


Figure 29: Photo taken by the depth camera (blue indicates a closer object)

Thermal camera: the thermal camera was used to obtain each plant's temperature value. The temperature of the plant (in particular its leaves) is changed by the amount of

water that the plant contains, thus can be a good indicator of a plant experiencing water stress.

Figure 30: Photo taken by the thermal camera

9.2.3 Segmentation

Since the plants were placed in the greenhouse relatively close to each other and the cameras used had a broad lens, neighbouring plants that are not the photographed plant entered the frame. Therefore, in the first stage we cropped each photo to center it around the wanted plant as can be seen in the following example:

Figure 31: (left) the original photo, (right) the cropped photo around the wanted plant

Since all features extracted from the different cameras mentioned above were computed by the values of the plants' leaves, we wanted to be able to know which part of the photo is a leaf and which is the background. Therefore, we developed a segmentation tool based on a CNN algorithm and trained it using a train set of RGB images of plants. The training set of images was created by pasting single cut out leaves of different size and shape on background images in a position that recreates an artificial plant. With respect to the inference part of the segmentor, the algorithm took as input a plant's RGB image and produced text files each with the contour points of every leaf in the photo. An example of the segmentor's work is presented below:

Figure 32: Marked in red are the contours around each leaf made by the segmentation tool

The leaf segmentor suffered from a few issues that needed to be dealt with; it had a hard time identifying leaves that were small/hidden/curled, these issues are planned to be dealt with in the near future by re-training the segmentor. However, a critical issue that we were able to deal with was that the segmentor wrongfully identified leaves of neighbouring plants. To deal with this issue a plant segmentor was trained and used. The plant segmentor identifies the whole plant in the image and not each of its leaves, hence making it robust to the neighbouring leaves problem. At the feature extraction stage, information was extracted from leaves that were identified by the leaves segmentor only if they were also placed inside the contours of the plant segmentor. An example of the neighbouring leaves problem and its solution is presented below:

Figure33 : Left image shows the leaves segmentor identifying neighbor leaves, right image shows the plant segmentor identifying the whole plant and ignoring the neighbor leaves

9.3 Appendix C – Mathematical Formulation of Evaluation Metrics

The following definitions will help formulating the binary class and multiclass metrics:

- True Positive (**TP**): The number of samples with the ground truth label of $'1'$ that were also predicted as class $'1'$ by the algorithm.
- False Positive (**FP**): The number of samples with the ground truth label of $'0'$ that were predicted as class $'1'$ by the algorithm.
- True Negative (**TN**): The number of samples with the ground truth label of $'0'$ that were also predicted as class $'0'$ by the algorithm.
- False Negative (**FN**): The number of samples with the ground truth label of $'0'$ that were predicted as class $'1'$ by the algorithm.

9.3.1 Binary Class Metrics

$$\text{False Alarm (}\mathbf{FA}\text{)} = \frac{FP}{FP + TN}$$

$$\text{Precision\_Pos} = \frac{TP}{TP + FP} \quad , \quad \text{Precision\_Neg} = \frac{TN}{TN + FN}$$

$$\text{Recall\_Pos} = \frac{TP}{TP + FN} \quad , \quad \text{Recall\_Neg} = \frac{TN}{TN + FP}$$

$$\mathbf{F1\text{-}Pos} = 2 * \frac{Precision\_Pos * Recall\_Pos}{Precision\_Pos + Recall\_Pos} \quad , \quad \mathbf{F1\text{-}Neg} = 2 * \frac{Precision\_Neg * Recall\_Neg}{Precision\_Neg + Recall\_Neg}$$

$$\mathbf{F1\text{-}Avg} = \frac{F1{-}Pos + F1{-}Neg}{2}$$

$$\text{Geometric Mean (}\mathbf{G\text{-}Mean}\text{)} = \sqrt{Precision\_Pos * Precision\_Neg}$$

$$\text{Balanced Accuracy (}\mathbf{B\text{-}Accuracy}\text{)} = \frac{\left(\frac{TP}{TP + FN} + \frac{TN}{FP + TN}\right)}{2}$$

9.3.2 Multiclass Metrics

$$\text{Weighted Precision (}\mathbf{W\_Precision}\text{)} = \sum_k^K \frac{|k|}{s} * \frac{TP_k}{TP_k + FP_k}$$

$$\text{Weighted Recall (}\mathbf{W\_Recall}\text{)} = \sum_k^K \frac{|k|}{s} * \frac{TP_k}{TP_k + FN_k}$$

$$\text{Weighted F1 (}\mathbf{W\_F1}\text{)} = \sum_k^K \frac{|k|}{s} * 2 * \frac{Precision\_Pos_k * Recall\_Pos_k}{Precision\_Pos_k + Recall\_Pos_k}$$

$$\text{Kappa} = \frac{c * s - \sum_k^K p_k * t_k}{s^2 - \sum_k^K p_k * t_k}$$

$$\text{Matthews} = \frac{c * s - \sum_k^K p_k * t_k}{\sqrt{\left(s^2 - \sum_k^K p_k^2\right) * \left(s^2 - \sum_k^K t_k^2\right)}}$$

- K – The total number of classes
- $c = \Sigma_k^K c_{kk}$ – the total number of elements correctly predicted
- $s = \sum_i^K \Sigma_j^K c_{ij}$ – the total number of elements
- $p_k = \sum_i^K c_{ki}$ – the number of times that class k was predicted
- $t_k = \sum_i^k c_{ik}$ – the number of times that class k truly occurred

9.4 Appendix D – CDDRL Article

Concept Drift Detection and Re-learning Using Bayesian Networks

Yotam Baron[§], Shon Mendelson[§], Liran Nahum[§], Moriya Cohen, Yoav Reisner, and Boaz Lerner
Department of Industrial Engineering and Management, Ben-Gurion University, Israel.

***Abstract*—We propose a two-stage algorithm for concept drift (CD) detection in data streams that adapts a Bayesian network (BN) to follow changes in the distribution over time. In the first stage, we sequentially monitor statistical distances between BN's conditional parameter tables through time using statistical process control charts to identify parametric and structural changes that are needed in the BN to follow the drifted distribution/concept. We prove the correctness of this stage. In the second stage, we re-learn the BN locally by searching a neighborhood of graphs derived from the graph used before the drift with structural changes only in nodes that have been detected as drifted nodes in the previous stage. We present two ensemble methods of our unsupervised algorithm and show; using five synthetic and seven real-life datasets; that it is superior to supervised algorithms in classification tasks, and it also identifies CD in the absence of a target variable—an ability that supervised algorithms lack. Finally, through visualization of the BNs learned during a data stream, we show our algorithm's unique ability to explain and better understand processes and variables' interrelationships in the domain - making it also a proper knowledge representation algorithm.**

***Index Terms*—concept drift, Bayesian network, statistical process control, statistical distance, stream learning.**

I. Introduction

MOST machine-learning methods assume examples are generated by a stable process, from a stationary distribution. But this is not true in many applications, e.g., in data streams, where the characteristics of the data might change over time, a condition known as concept drift [CD; [1]]. In general, a change can arise for many reasons, e.g, price growth due to inflation, the needs of an aging population, the effect of global warming, or equipment wear. Improvements in technology and standards result in ever-improving precision in measurements and change the systematic errors that inevitably affect recorded values to further increase the chances of CD.

CD supervised learning is either *real* or *virtual* [1]; the former is a change in the distribution of the target variable given the input [$P(y|X)$], and the latter is a change that does not affect this distribution [i.e., change in $p(y)$, $p(X)$, or $p(X|y)$]. The former is usually considered more important to detect, as it requires updating the model's decision boundary [2].

Although CD is usually discussed in supervised settings [3], [4], we argue that it should be discussed in a more general view, especially for streaming data where the target variable can change over time. For example, when creating a new brand the initial target variable is whether a customer will click the ad, but when the brand is established, the target variable may be whether the customer will buy the product (and the previous target variable can now be used as an explanatory variable). In an unsupervised setting, the separation between virtual and real CD is irrelevant and each change in the distribution should be detected and treated. Unsupervised approaches for CD detection are usually non-parametric tests [5]–[7]. While they may be an excellent scheme for general problems that require no assumptions, this advantage can be turned into a disadvantage where there are complex relationships between many variables in the domain.

A Bayesian network (BN) is a knowledge-representation graphical model that efficiently encodes conditional independences over a set of variables to promote the understanding of causal relations in the domain. Its structure comprises nodes representing the domain variables, and directed edges connecting these nodes representing probabilistic causal relations between the corresponding variables [8], [9]. This model suits our general view of CD, as in the BN, each of the variables can be the target variable.

The number of possible BN structures grows exponentially with the number of nodes [10]. Thus, in most domains, learning the structure from data by considering all possible structures is not feasible, regardless of the size of the data, meaning that BN structure learning is NP-hard [11], which requires sub-optimal procedures. This makes utilizing a BN in a streaming setting even more challenging, since the model should continually be updated.

To address these issues, we propose concept drift detection and re-learning (CDDRL). This is a two-stage algorithm that, at each time step, first detects drifts by applying statistical process control to sequentially monitor distances between current estimated conditional probability tables to those corresponding in the BN of the stable process (or that lately learnt; see Section IV). Not only does it detect a change that has occurred due to a drift, but it identifies where in the network (for which nodes) it occurred. In its second stage, a search & score procedure locally modifies the BN of the stable process (or that currently known) using only the nodes which have first been detected as changed, thus avoiding the need for globally learning the model at each time step.

We later on introduced several developments to the CDDRL algorithm; see Section VI; mainly regarding the method to obtain the threshold for alerting of a drift, an improvement to the BN structure search procedure and to the way we learn the conditional probabilities at each time step. When dealing with classification problems, these suggested improvements are the components of our first ensemble version of the CDDRL. The

[§]Equal contribution

second ensemble method we introduce, combines weighted predictions of BNs learnt in previous time steps by the CDDRL into a single prediction for the current time step. The experiments in this paper evaluate these two ensemble methods as well.

The benefits of the CDDRL algorithm are: 1) Detects and re-learns changes in the distribution of any variable, compared to most existing approaches that are supervised and thus cannot handle unsupervised tasks or scenarios in which either the target or any other variable changes over time; 2) Does not re-learn the entire BN structure at each time step, which makes it ideal for a streaming setting; 3) Enables identification of the source of the drift, as it searches for a CD separately for each variable—an ability most existing approaches lack; and 4) Produces a graphical representation of the domain (a BN) at each time step, allowing us to explain and better understand processes and relations in the data. We demonstrate these benefits using 1) synthetic data for classification tasks with various CD structures (i.e., sudden and gradual); 2) data generated from well-known BNs, where CD is introduced by changing the network structure (i.e., by adding/removing/reversing an edge); and 3) seven real-life applications; four known datasets and three experiments done by us in the agriculture field.

II. RELATED WORK

Algorithms dealing with concept drift can be roughly divided into active and passive approaches [12]. Active approaches integrate a drift-detection method to trigger a re-training of the model. Passive approaches on the other hand, do not use drift-detectors, but rather update their model every predefined number of observations or time steps.

Most drift detection methods look for significant deviations between statistics of current and past observations to alert that a drift has occurred. The most common statistic is the classification error; DDM [13] monitors the classification error rate and its standard deviation to detect drifts, the authors argue the error rate should decrease as the number of examples increases, as long as the data distribution is stationary, and, when the error rate increases, DDM implies the data distribution has changed. EDDM [14] is very similar to DDM except that it analyzes the distance between consecutive errors, rather than the error rate.

Another drift-detection method that is vastly used is known as Adaptive Windowing [ADWIN; [15]]. ADWIN maintains a sliding window of instances (W) divided in two dynamically adjusted sub-windows, which represent older and recent data, respectively. Drifts are detected when the difference of the means of these sub-windows is higher than a given threshold.

Kolmogorov-Smirnov Windowing [KSWIN; [7]] is a concept drift detection method based on the Kolmogorov-Smirnov (KS) statistical test. KS-test is a statistical test with no assumption of underlying data distribution. KSWIN can monitor data or performance distributions.

As mentioned, active approaches incorporate a drift-detection method accompanied with a base estimator to formulate a model that can deal with concept drifts and streaming data. KNN-ADWIN is such an example, where ADWIN first detects and removes drifted examples, and then a classifier is trained, e.g., the K-nearest-neighbor (KNN). The Very Fast Decision Rules [VFDR; [16]] is a rule-based classifier, each rule is a conjunction of conditions based on attribute values and the structure for keeping sufficient statistics. The sufficient statistics will determine the class predicted by the rule. Each rule is accompanied with a drift-detection method (in this paper we used EDDM), once a change is detected, the matching rule is discarded and a new one is learned. The Online Boosting Classifier [OBC; [17]] is the online version of AdaBoost, every arriving observation is sampled k times using a Binomial distribution. Since data is infinite, the Binomial distribution tends to act like a Poisson distribution, λ is computed by tracking the total weights of the correctly and misclassified examples. OBC is accompanied with the ADWIN change detector. The Adaptive Random Forest [ARF; [18]] uses Hoeffding Trees as base estimators to form a random forest classifier. The ARF uses a drift-detector (in this paper we used KSWIN) per base tree, which cause selective resets in response to drifts. Neural Network Uncertainty [NN-UN; [19]] is a recent active method that uses Monte-Carlo Dropouts during several inferences of each incoming observation to calculate the model's uncertainty (entropy is used as a measure). These uncertainties are the input to an ADWIN change detector that alerts of drifts and triggers retraining of the model. The authors of the NN-UN state that the method will not identify real concept drifts if it is not accompanied with a virtual concept drift.

The Robust Soft Learning Vector Quantization [RSLVQ; [20]] is a passive prototype-based algorithm, that uses a gradient descent method to optimize the objective function. Lately, the Adadelta momentum-based SGD was introduced to the RSLVQ algorithm as a method to deal with streaming data.

Concept drift algorithms can be further sub-categorized into an adaptive single model and to ensemble methods. Passive ensemble methods include the accuracy-weighted ensemble [AWE; [21]] that weighs classifier decisions based on their expected classification accuracy on the test data, the dynamic weighted majority [DWM; [22]] uses such weights to remove models from the ensemble and to add new models based on the global performance of the ensemble, and LNSE [[4]] dynamically weighs the classifiers based on their accuracy in the current environment compared with past environments.

The active ensemble methods include the Self Adjusting Memory coupled with the KNN classifier [SAM-KNN; [12]] that builds an ensemble with models targeting current or former concepts and is built using two memories: STM for the current concept, and the LTM to retain information about past concepts. A cleaning process is in charge of controlling the size of the STM while keeping the information in the LTM consistent with the STM. Finally, the Streaming random patches [SRP; [3]] is an ensemble Hoeffding Tree based algorithm that combines random sub-spaces and bagging to classify under CD. Trees are replaced when they are detected as drifted using an ADWIN change detector.

Although all algorithms have good results, especially for real CD, these algorithms cannot detect the source of a

change. For example, in a marketing problem, searching for CD changes in customer preferences, the ability to identify what caused changes in preferences, which can be used in future marketing campaigns, is missing.

A recent related work of a BN-based approach is locally adaptive-MB-MBC [LA-MB-MBC; [23]], applying the Page-Hinkley test [24] to the average log-likelihood in order to update locally around each node that is detected as changed.

III. Preliminaries

In this section, we provide preliminaries to the CDDRL. In Section III-A, we introduce the BN and two definitions of BN equality and BN stationarity, and in Section III-B, we describe distance functions and statistical process control (SPC) charts that are used by the CDDRL.

A. Bayesian Networks

The BN structure, G, is a directed acyclic graph (DAG) over a set of random variables $\mathbf{X} = X_1, ..., X_n$, and the BN set of parameters, Θ, holds conditional probability tables (CPTs) each associated with a node (representing a variable). Each row in a node's CPT holds a local conditional probability distribution (CPD) for this node given a certain configuration of its parents in the graph, thus, the number of node CPDs in a CPT equals the number of possible parent value configurations of this node. Assuming discrete variables, a CPD is multinomial. Learning a BN starts with learning the network structure (building a DAG), and continues with learning the parameters (CPTs). While structure learning is NP-hard [11], parameter learning, e.g., by maximum likelihood estimation (MLE), is easily solved [25].

We introduce two definitions regarding BNs:

Definition 1: $BN_1 = BN_2$ if and only if: 1) the structure, G1, of BN1 is identical to the structure, G2, of BN2, i.e., all and only the edges in G1 appear in G2 with the same orientations, and 2) the estimated parameters, $\widehat{\Theta_1}$, of BN_1, are not statistically different than the estimated parameters, $\widehat{\Theta_2}$, of BN_2 (leading to likelihood equivalence [9]).

Following this definition of BN equality, we define BN stationarity, assuming for simplicity a stationary process in which the joint distribution does not change over time [26],

Definition 2: $BN_{[t]}$ represents a stationary process if $BN_{[t]} = BN_{[t+L]} \quad \forall t$ and $L \in \mathbb{Z}$.

B. Statistical Distance and Statistical Process Control

To detect CD at a specific time by a BN, we calculate the statistical distance between each individual CPD of a CPT estimated for a BN's node in the stable process (in-control) and the corresponding estimated distribution at this time. Then we apply an SPC to decide whether a distance is statistically unlikely, i.e., it is outside the process control limits (out-of-control), reflecting a change in the distribution of this CPD that demonstrates a drift of the corresponding node. We compared both theoretically and empirically several combinations of a statistical distance function and an SPC.

*1) **Statistical Distance**:* Let P and Q be two multinomial CPDs over a set of K variable values. We introduce four statistical distances:

Total Variation (TV) Distance measures the sum of absolute differences between P and Q [27]:

$$D_{TV}(P,Q) = \frac{1}{2}\sum_{k=1}^{K}|P(k) - Q(k)|. \tag{1}$$

Pearson Chi-Square (CS) Statistic tests whether the observed distribution (Q) is consistent with a hypothesized, expected distribution (P) [28]:

$$D_{CS}(P,Q) = \sum_{k=1}^{K}\frac{(P(k) - Q(k))^2}{P(k)} \tag{2}$$

Kullback-Leibler (KL) Divergence measures the divergence of the observed distribution Q from the expected one, P [29]:

$$D_{KL}(P,Q) = \sum_{k=1}^{K}P(k)log\frac{P(k)}{Q(k)} \tag{3}$$

Hellinger (H) Distance is based on the Bhattacharyya coefficient [30] and is calculated as:

$$D_H(P,Q) = (1 - \sum_{k=1}^{K}(P(k) * Q(k))^{1/2})^{1/2} \tag{4}$$

*2) **Statistical Process Control**:* To monitor the measured statistical distances over time, we apply the SPC methodology. Since the target statistical distance value is always set at zero (i.e., no drift), the distance is bounded below by zero; hence, we discard the lower limit and calculate only an upper control limit (UCL). Let $d_{[t]}$ be the t^{th} measurement (CPD difference) that we monitor, and $\hat{\sigma}$ be its standard deviation calculated using T measurements from a stable process. We consider four SPC charts:

The Shewhart Control Chart is the most common SPC chart [31] to monitor each measurement $d_{[t]}$ separately. Its UCL is based on a distance of 3 standard deviations from the target value (center line, CL, in SPC terms), $UCL = CL + 3 * \hat{\sigma}$.

The Exponentially Weighted Moving Average (EWMA) Chart [32]) monitors a weighted average of the measurements, such that the most recent one gets the greatest weight,

$$Z_{[t]} = \lambda * d_{[t]} + (1 - \lambda) * Z_{[t-1]}, \tag{5}$$

where $Z_{[0]}$ is the target value of the process, and $\lambda \in (0, 1)$ is a smoothing parameter. The UCL is

$$CL + L * \hat{\sigma} * \sqrt{\frac{\lambda}{2 - \lambda}}, \tag{6}$$

where L is the width of the control limits, set at 3 in this work.

Both the Shewhart and EWMA charts are designed for univariate monitoring, meaning a single measure at a time. The following SPC charts are extensions of these two, designed to monitor multiple variables jointly. Let $d_{[t]}$ denote the multivariate measurement vector at time t with J values, $J = |d_{[t]}|$, and $\overline{d}$ and S are the measurements mean vector and covariance matrix, respectively.

The Hotelling T^2 Control Chart, which is the multivariate version of the Shewhart chart, measures the Mahalanobis distance of $d_{[t]}$ from $\overline{d}$. The T^2 statistic is calculated as

$$T^2{}_{[t]} = (d_{[t]} - \overline{d})^{'} S^{-1} (d_{[t]} - \overline{d}), \tag{7}$$

where

$$S = \frac{1}{T-1} \sum_{t=1}^{T} (d_{[t]} - \overline{d})(d_{[t]} - \overline{d})^{'}, \tag{8}$$

and [33]

$$UCL = \frac{J(T+1)(T-1)}{T^2 - TJ} F_{\alpha,J,T-J}, \tag{9}$$

where $F_{\alpha,J,T-J}$ is the $(1-\alpha)^{th}$ quantile of the $F_{J,T-J}$ distribution.

The multivariate EWMA (MEWMA) Chart is the multivariate version of EWMA [34]. $Z_{[t]}$ is the average of $d_{[t]}$:

$$Z_{[t]} = \lambda * d_{[t]} + (1-\lambda) * Z_{[t-1]}. \tag{10}$$

Similar to Hotelling T^2, the statistic is calculated as

$$T^2{}_{[t]} = Z^{'}{}_{[t]} S^{-1}{}_{Z(t)} Z_{[t]} \tag{11}$$

where

$$S_{Z(t)} = \frac{\lambda}{2-\lambda} [1 - (1-\lambda)^{2t}] S \tag{12}$$

and

$$S = \frac{1}{T-1} \sum_{t=1}^{T} (d_{[t]} - \overline{d})(d_{[t]} - \overline{d})^{'}. \tag{13}$$

Since there are no recommended values to UCL of MEWMA, we set it heuristically based on the maximum $T^2[t]$ value calculated using 200 known in-control measurements.

IV. CONCEPT DRIFT DETECTION AND RE-LEARNING

Drift detection by parameter monitoring (DDPM) to track a chosen distance function using a chosen SPC, and re-learning the current BN's structure and parameters using local re-learning (LRL) are the first and second stages of the CDDRL algorithm, respectively. Initially, CDDRL was implemented with its static version, where at every time step, the baseline network was considered to be the initial BN learnt at the stable process. The baseline network is the one containing the parameters we measure the distance from at any given point in time. Later, we implemented the dynamic version of the CDDRL, where the baseline network was taken to be the BN that was learned by the CDDRL at the last time step when a change was detected. This baseline network for either of the versions will be referred to as $BN_{baseline}$.

A. DDPM

*1) **Network Measurement**:* We apply parameter estimation, which is significantly inexpensive relative to structure learning, to identify changes in the network parameters or structure. A change in the network, i.e., BN inequality $BN_{[t]} \neq BN_{[t+L]}$ (Definition 1), indicates a loss of stationarity (Definition 2) i.e, CD. The initial BN, $BN_{t=0}$, learned during the stable process is assumed correct before any change occurs. Since learning the structure without an initial network is not in the scope of this work, we assume a proper initial BN—denoted as BN_0—has been learned based on historical data, and that it is faithful to the true unknown network BN^*_{pre}. Moreover, at every time step, by assuming a fixed, reference BN structure denoted as $BN_{baseline}$ and estimating only its parameters using the latest examples available, we learn at time t a new BN, $\widehat{BN_{[t]}}$, with the same structure, but with new parameter values. To test if $\widehat{BN_{[t]}} = BN_{baseline}$, the DDPM measures statistical distances between corresponding CPDs of the networks (Figure 1).

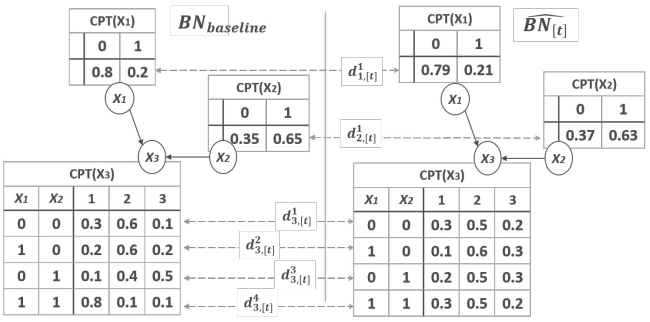


Fig. 1. Measuring statistical distances between corresponding CPDs.

In some scenarios, we would like to use BN_0 as $BN_{baseline}$, even after a drift has been detected, in order to continue monitoring the drift compared to the point of stability. In other scenarios, probably more relevant to real applications, once a drift is detected and $\widehat{BN_{[t]}}$ is learned, it becomes the new baseline model, $BN_{baseline}$, that is used as the new reference (instead of the stable model, BN_0) to detect further drifts. We refer to these approaches as "static" and "dynamic", respectively, and perform experiments with both.

For practical reasons, two parameters should be determined: a window size (W), which is how many of the latest examples should be used to learn the parameters of $\widehat{BN_{[t]}}$; and a jump size (S), which is how often measurements should be taken.

*2) **Measurement Monitoring**:* Since in practice, the parameters are estimated using limited data, and variability is an inevitable component, we cannot assume that every distance greater than zero will indicate a true drift. Hence, we monitor the BN stationarity measured by parameter distances using the SPC chart that monitors the process variation to detect such a drift [31]. We keep the limits of the charts (UCL) fixed during monitoring, and calculate them based on historical data from which BN_0 is learned, or from a preliminary time frame in which no changes occurred.

To identify which node is involved in a change (drift), DDPM monitors the statistical distance for each node separately using SPC. When a node that has multiple CPDs (e.g., X_3 in Figure 1) is monitored by a univariate SPC (Shewhart and EWMA), we aggregate CPD distances and monitor a single CPT distance. Aggregation for X_i is by a weighted mean function with weights w_{ij} that are equal to the proportions of the q_i parent configurations for X_i in the data

($\sum_{j=1}^{q_i} w_{ij} = 1 \quad \forall i$). Let $d_{i,[t]}^{j}$ be the distance measured for the j^{th} CPD of X_i at time t and $d_{i,[t]}^{A} = \sum_{j=1}^{q_i} w_{ij} * d_{i,[t]}^{j}$ be the aggregated distance of $d_{i,[t]}^{j}$ over all q_i parents configurations.

After estimating network parameters with recent examples and measuring CPD distances (which are further aggregated if the selected SPC chart is univariate), we obtain a distance measure per node, which indicates, based on this node, how far our current model is from the baseline model. All node distances are monitored using SPC, and if at least a single distance is higher than the control limit, a change is alerted. Algorithm 1 shows the process of the DDPM algorithm which is the first stage of the CDDRL.

Algorithm 1 DDPM algorithm.

input: $BN_{baseline}, D_{[t]}, UCLs, Dist_{Metric}$

$C_{nodes} \leftarrow []$
$BN_{[t]} \leftarrow$BN$_{baseline}$ with parameters learned with $D_{[t]}$
for $node = 1$ to $Nodes_{num}$ **do**
  $d_{vector} \leftarrow []$
  for $combination = 1$ to $Parents_{Combinations}$ **do**
    $d_{vector}[combination] \leftarrow$ The distance between the CPD in $BN_{baseline}$ to the matching one in $BN_{[t]}$, calculated using $Dist_{Metric}$
  end for
  $Aggregated_{Dist} \leftarrow$ Aggregate(d_{vector})
  if $Aggregated_{Dist} > UCLs[node]$ **then**
    Add $node$ to C_{nodes}
  end if
end for
$return \quad C_{nodes}$

B. LRL

Based on the information from the DDPM algorithm which nodes have been changed, the LRL algorithm locally re-learns a new network by considering for each such node a parametric change or a structural one: an addition, removal, or reversal of an edge, without the need for re-learning the complete graph from scratch. When considering a structural change, we search the space by starting from $BN_{baseline}$, performing a finite number of steps and, at each step, only consider local changes that can lead to neighboring DAGs. The LRL chooses the one DAG resulting in the greatest improvement in a score function and stops when there is no local change yielding such improvement.

The effectiveness and efficiency of our learning method are obtained by 1) considering only a neighborhood of the DAG used before the drift has been detected, and 2) using such a neighborhood of only the graphs that can possibly represent the drift because each is derived based on local changes corresponding to those nodes in the DAG which have experienced a change according to the DDPM algorithm.

Algorithm 2 shows that the input of the LRL algorithm is $D_{[t]}$, a new dataset defined over variables $\mathbf{V} = X_1, ..., X_n$; g, the DAG of $BN_{baseline}$ defined over $\mathbf{V}$ used as the starting point for the search; and C_{nodes}, the nodes which were identified as changed by the DDPM algorithm. At each iteration, a list of candidate graphs is created by either removing, adding, or reversing an edge from a node which was identified as changed (C_{nodes}) by the DDPM, avoiding possible cycles. As the LRL does not rely on a specific scoring function, we used here the Bayesian Dirichlet with likelihood equivalence and a uniform joint distribution [BDeu; [35]]. Optimization is by simulated annealing, i.e., if g^* (the graph with the maximal score in the neighborhood) improves the existing graph score, then it is always accepted. Otherwise, the algorithm makes the move with some probability less than 1 [36] that decreases exponentially as the graph score worsens, which is the amount of Δ by which the solution is worsened. In each iteration, the best score is updated ($BestScore$). We initialized two parameters: T (temperature) and α (a constant relating temperature to energy) using $T_0 = 0.025$ and $\alpha = 0.5$. At the end of each iteration, the temperature is decreased by α which causes the probability for acceptance to decrease as the iterations proceed.

Algorithm 2 LRL algorithm.

input: $D_{[t]}, \mathbf{V}, g, C_{nodes}$

$BestScore \leftarrow Score(g, D)$
while $improvement$ **do**
  $Candidates \leftarrow$ all graphs created from g by either adding, removing, or reversing an edge from a node in $\mathbf{V}$ to one in C_{nodes}
  $S^* \leftarrow max_{g \in Candidats} Score(g, D)$
  $g^* \leftarrow$ the corresponding graph to S^*
  $\Delta \leftarrow (S^* - BestScore) \,/\, BestScore$
  if $S^* > BestScore$ **then**
    $BestScore \leftarrow S^*$
    $g \leftarrow g^*$
  else
    if $exp(-\Delta \,/\, T) > rand()$ **then**
      $BestScore \leftarrow S^*$
      $g \leftarrow g^*$
    else
      $improvement \leftarrow false$
    end if
  end if
  $T \leftarrow T * \alpha$
end while
$return \quad g$

After obtaining the structure of $BN_{[t]}$ by LRL, we learn the network's parameters using MLE and the last (W) observations to get the complete learned BN for the current time step. Algorithm 1 followed by Algorithm 2 executed at every time step, formulate our CDDRL algorithm.

V. Theoretical Correctness

We mathematically prove the correctness of DDPM.

Theorem 1: *DDPM detects all types of structural and parametric changes in BN.*

We prove Theorem 1 by first proving that DDPM detects parametric changes in the BN (*Lemma 1*) as well as structural changes followed by parametric changes (*Lemma 2*).

Lemma 1: *DDPM detects a change in the BN parameters.*

Proof of Lemma 1: Since MLE is asymptotically a consistent estimator, two estimations have different values iff there is a difference in the parameter value they estimate, i.e., a change in the parameter. **Q.E.D.**

In section VII-A, we show that even with finite data, DDPM correctly identifies changes.

Let $\Theta_{X_i|Z_j[BN_k]}$ be the probability that $X = i$ given that a set of X's parents $\mathbf{Z}$ in BN_k gets its J^{th} configuration. Let $\mathbf{\Theta}_{[BN_1]}$ be the parameter set (i.e., all variable probabilities) for BN_1 and $\mathbf{\Theta}_{[BN_1]} \asymp \mathbf{\Theta}_{[BN_2]}$ stands for $\mathbf{\Theta}_{[BN_1]}$ is asymptotically equivalent to $\mathbf{\Theta}_{[BN_2]}$, i.e., $\lim_{w\to\infty} \frac{\Theta_{X_i|\mathbf{Z}_j[BN_1]}}{\Theta_{X_i|\mathbf{Z}_j[BN_2]}} \to 1$ for all i and j, where w (the window size) is the number of observations used to estimate the parameters.

By assumption, $\mathbf{\Theta_{BN_0}}$ is faithful to the real network at time 0, i.e., $\mathbf{\Theta}_{[BN_0]} \asymp \mathbf{\Theta}_{[BN^*_{pre}]}$. Before any change occurs, the learned parameter set $\mathbf{\Theta}_{[\widehat{\mathbf{BN_t}}]}$ estimated by MLE is asymptotically equivalent to $\mathbf{\Theta}_{[BN^*_{pre}]}$, i.e., $\mathbf{\Theta}_{[\widehat{\mathbf{BN_t}}]} \asymp \mathbf{\Theta}_{[BN^*_{pre}]}$. Hence, the distance between $\mathbf{\Theta_{BN_0}}$ and $\mathbf{\Theta}_{[\widehat{\mathbf{BN_t}}]}$ is zero. After the network parameters have been changed, i.e., $\mathbf{\Theta}_{[BN^*_{pre}]} \neq \mathbf{\Theta}_{[BN^*_{post}]}$, the learned parameters $\mathbf{\Theta}_{[\widehat{\mathbf{BN_t}}]}$ are asymptotically equivalent to $\mathbf{\Theta}_{[BN^*_{post}]}$; hence, the distance between $\mathbf{\Theta}_{[\widehat{\mathbf{BN_t}}]}$ and $\mathbf{\Theta_{BN_0}}$ is now greater than zero. Therefore, as long as there is no change in the parameters, the measured distances are zero, and if all distances are zero, the average distance and its variance are also zero, as is the upper control limit of any SPC procedure. Once a change occurs in the parameters, the measured distances increase, exceeding the control limit and alerting a change in the network.

Lemma 2: *DDPM detects a change in the BN structure.*

It is sufficient to show that a structural change is followed by a change in the learned parameters when the structure remains of BN^*_{pre}, but the observations are generated from BN^*_{post}. Then, according to *Lemma 1*, a change in the parameters is guaranteed to be detected by DDPM.

There are three possible structural changes: removal, addition, and reversal of an edge. Thus, we divide *Lemma 2* into three smaller lemmas, one for every possible change. For the sake of *Lemma 2*, when we deal with removal of an edge, $\mathbf{Z}$ holds parents of X, except for the one that is disconnected from X, and when we deal with addition of an edge, $\mathbf{Z}$ holds the parents of X that were connected to X before the addition. Y is the parent node that was either connected to or disconnected from the changed node. The numbers of configurations of $\mathbf{Z}$, values of Y, and values of X are $K1$, $K2$, and $K3$, respectively.

Lemma 2.1: *DDPM detects removal of an edge in BN structure.*

Let X be a node in BN with parent Y, where all other parents of X besides Y are in Z, and consider the removal of the edge $Y \to X$. Figure 2 shows edge removal between true BNs and as observed during learning prior to change detection. Note that the existence of $Y \to X$ in BN^*_{pre} is justified if and

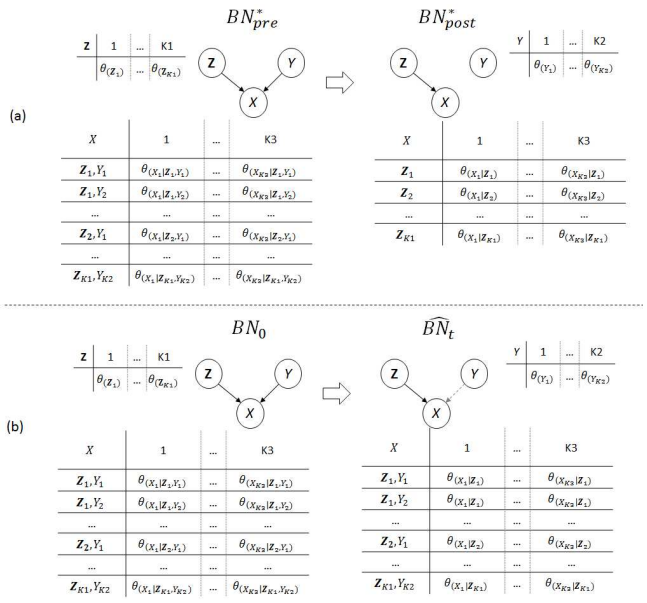


Fig. 2. Edge removal transition in (a) the real underlying BN and (b) as observed by learning the parameters. The dashed gray edge is incorrectly assumed to exist.

only if there is a difference in the probabilities of X given different values of Y, i.e., there are some h and l such that

$$\Theta_{(X_i|Z_j,Y_h)[BN^*_{pre}]} \neq \Theta_{(X_i|Z_j,Y_l)[BN^*_{pre}]}. \tag{14}$$

Therefore, removal of the edge is demonstrated by equality for h and l in (14) (for brevity, from now on, we consider equality as being asymptotic equality). Figure 2(a) shows the real network before (left) and after (right) the removal. Figure 2(b) presents the learned parameters prior to the removal (left) and after it (right). The dashed gray arrow from node Y to node X represents the edge we believe exists in the structure although it was already removed in the real underlying network BN^*_{post}. Following the removal, and prior to its detection, the parameters are learned based on the structure before the change (BN_0). Since learning assumes Y has an effect on X, the parameters for X are learned for every possible value of Y, but the observations used after the change cannot demonstrate any effect of Y on X. Therefore, the learned parameters will hold for the equality $\Theta_{(X_i|Z_j,Y_h)[\widehat{BN_t}]} = \Theta_{(X_i|Z_j,Y_l)[BN^*_{post}]} \forall i,j,h$ (i.e., for any X_i and Z_j the parameters $\Theta_{(X_i|Z_j,Y_h)} \forall h$ are the same) (See Figure 2).

Proof of Lemma 2.1: Because $Y \to X$ was removed in BN^*_{post},

$$\Theta_{(X_i|Z_j,Y_h)[\widehat{BN_{[t]}}]} = \Theta_{(X_i|Z_j,Y_l)[\widehat{BN_{[t]}}]}, \forall h,l. \tag{15}$$

Let's assume by contradiction that there is no change in the learned parameters of node X following the removal of $Y \to X$, i.e.,

$$\Theta_{(X_i|Z_j,Y_h)[\widehat{BN_{[t]}}]} = \Theta_{(X_i|Z_j,Y_h)[BN_0]}, \forall h. \tag{16}$$

which means that, based on (15),

$$\Theta_{(X_i|Z_j,Y_h)[BN_0]} = \Theta_{(X_i|Z_j,Y_l)[BN_0]}, \forall h,l. \tag{17}$$

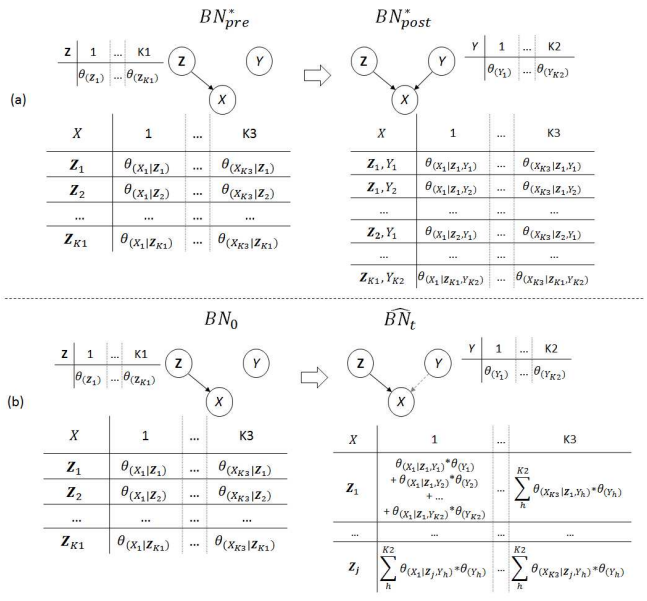


Fig. 3. Edge addition transition in (a) the real underlying BNs and (b) as observed by learning the parameters.

Since $Y \rightarrow X$ exists in BN^*_{pre} (and thus also in BN_0), (17) contradicts (14), which means that there is a change in the parameters following the removal of $Y \rightarrow X$, and based on *Lemma 1*, this change is detected by DDPM. **Q.E.D.**

Lemma 2.2: *Except for a rare case, DDPM detects the addition of an edge in BN structure.*

Let X be a node in BN and consider the addition of edge $Y \rightarrow X$ between Y and X, where $X's$ parents before the addition are in $\mathbf{Z}$. Note that the existence of $Y \rightarrow X$ in BN^*_{post} is justified if and only if, there is a difference in the probabilities of X given different values of Y, i.e., there are some h and l such that

$$\Theta_{(X_i|Z_j,Y_h)[BN^*_{post}]} \neq \Theta_{(X_i|Z_j,Y_l)[BN^*_{post}]}. \tag{18}$$

Figure 3 demonstrates edge addition. Figure 3(a) shows the real network before (left) and after (right) the addition. Figure 3(b) presents the learned parameters prior to the addition (left) and after it (right). The dashed gray arrow between node Y and X represents the edge in the real underlying network BN^*_{post} that we are not aware of in $\widehat{BN_{[t]}}$.

Following the addition, and prior to its detection, the parameters are learned based on BN_0. Since learning assumes Y has no effect on X, the parameters for X are learned without reference to Y. However, the observations generated from BN^*_{post} reflect the effect of Y on X. Therefore, the learned parameters of $\widehat{BN_{[t]}}$ [Figure 3(b)] equal the parameters of BN^*_{post} after marginalizing Y out:

$$\Theta_{(X_i|Z_j)[BN_t]} = \sum_{h}^{K2} \Theta_{(X_i|Z_j,Y_h)[BN^*_{post}]} * \Theta_{(Y_h)[BN^*_{post}]}. \tag{19}$$

Proof of Lemma 2.2: Let's first define the very rare case in which edge addition cannot create a change in the parameters. Such a case requires the parameters in $\widehat{BN_{[t]}}$ after the addition to equal the parameters in BN_0 (see Figure 3):

$$\Theta_{(X_i|Z_j)[\widehat{BN_{[t]}}]} = \Theta_{(X_i|Z_j)[BN_0]} = \sum_{h}^{K2} \Theta_{(X_i|Z_j,Y_h)[BN^*_{post}]} * \Theta_{(Y_h)[BN^*_{post}]}. \tag{20}$$

We define a change factor $A_{h(i,j)} > 0$ using the posterior $\Theta_{(X_i|Z_j,Y_h)[BN^*_{post}]}$ and prior $\Theta_{(X_i|Z_j)[BN^*_{pre}]}$ probabilities

$$\Theta_{(X_i|Z_j,Y_h)[BN^*_{post}]} = A_{h(i,j)} * \Theta_{(X_i|Z_j)[BN^*_{pre}]}. \tag{21}$$

Rewriting (20) using (21):

$$\Theta_{(X_i|Z_j)[BN_0]} = \sum_{h}^{K2} \Theta_{(X_i|Z_j,Y_h)[BN^*_{post}]} * \Theta_{(Y_h)[BN^*_{post}]} = \sum_{h}^{K2} A_{h(i,j)} \Theta_{(X_i|Z_j)[BN^*_{pre}]} * \Theta_{(Y_h)[BN^*_{post}]} = \Theta_{(X_i|Z_j)[BN^*_{pre}]} * \sum_{h}^{K2} A_{h(i,j)} * \Theta_{(Y_h)[BN^*_{post}]}. \tag{22}$$

Since $\Theta_{(X_i|Z_j)[BN_0]} = \Theta_{(X_i|Z_j)[BN^*_{pre}]}$, we get

$$\sum_{h}^{K2} A_{h(i,j)} * \Theta_{(Y_h)[BN^*_{post}]} = 1. \tag{23}$$

Since $\sum_h^{K2} \Theta_{(Y_h)[BN^*_{post}]} = 1$, $\sum_h^{K2} A_{h(i,j)} \Theta_{(Y_h)[BN^*_{post}]}$ is a weighted by $\Theta_{(Y_h)}$ average of the change factors $A_{h(i,j)}$ for every i and j. Therefore, for (23) to hold, $A_{h(i,j)}$ is required to be one for any i and j. But to hold the inequality in (18), there must be at least a single $A_{h(i,j)} \neq 1$ for some i, j, h. Then, there must also be an additional $A_{h(i,j)} \neq 1$ such that if $A_{h(i,j)} > 1$, then $A_{l(i,j)} < 1$ and vice versa. Meaning that to generate a case where we observe no change in the learned parameters (21), there must be factors that are both proportional to $\Theta_{(Y_h)}$ and also coordinately and oppositely directed for any i and j. This is a singular case that can hardly be expected in real-life scenarios.

In any other case, where there is no equality to one in (23) for every i and j, i.e., the change factors are not expected to be coordinated, any addition of an edge is followed by a change in the learned parameters. Based on *Lemma 1*, this change is detected by DDPM. **Q.E.D.**

Lemma 2.3: *DDPM detects reversal of an edge in BN structure.*

Proof of Lemma 2.3: Since DDPM monitors each node separately with respect to its parents, reversal of an edge can be decomposed into two sub-changes: the removal of an edge and the addition of an edge. When $X \rightarrow Y$ is reversed to $Y \rightarrow X$, node X experiences the addition of an edge, and node Y experiences the removal of an edge. Based on *Lemma 2.1* and *Lemma 2.2*, both changes are detected by DDPM; hence, DDPM also detects reversal of an edge. **Q.E.D.**

This completes the proof of Theorem 1.

Theorem 2: *If DDPM detects a change, the change exists in the real underlying network.*

Proof of Theorem 2: If the learned parameters at time t, $\mathbf{\Theta}_{[\widehat{BN_t}]}$, are found to be different than those of the initial

network, $\mathbf{\Theta}_{[BN_0]}$, then, since learning is by the consistent MLE, this difference exists also between the parameters of the underlying networks BN^*_{post} and BN^*_{pre}. **Q.E.D.**

VI. CDDRL-BASED ENSEMBLES

We will shortly describe several developments we introduced to the CDDRL algorithm; Three of them relate to the way in which we obtain our UCLs, search for the current BN structure and learn the parameters of the current BN. Two other developments change the way in which we make predictions in classification tasks; these two are in fact ensemble methods.

*1) **Dynamic UCLs**:* In the original version of the CDDRL, the UCLs (which are the thresholds for alerting of drifted nodes) are calculated once in the initial stable period and are kept unchanged in the future. In the $Dynamic\ UCLs$ version of the CDDRL, the UCLs change through out the data stream. The UCL for a specific node is re-calculated whenever the node is not alerted of being drifted for a predefined number of successive time steps. We consider these unchanged time steps to be a stable period for the node, thus allowing us to safely compute a new UCL for the node in the same manner calculated in the original CDDRL version. The new UCL is kept unchanged until another stable period occurs.

*2) **Tabu Search**:* Tabu search [37] is a well-known method used in optimization problems with a large search space, such as ours. In short, a tabu list of recently visited solutions is kept in memory, not allowing the search algorithm to revisit those solutions and by that, increasing the chance to escape from a local minima/maxima. Algorithm 2 shows the search and score process of the CDDRL after the best suiting current BN structure. In the $Tabu\ Search$ version of the CDDRL, we insert into Algorithm 2 a tabu list of predefined size (10 is used in this paper), that keeps in memory the chosen graph in each iteration and does not allow that graph to be revisited in the next tabu list size (10) number of iterations.

*3) **Parameter Learning**:* In the original CDDRL version, parameters (conditional probabilities) of the chosen BN structure are fitted using MLE. The issue with MLE, is that it may overfit the data [38]. In our case, it might happen given a small window size, or a specifically noisy jump. Therefore, we introduce the $Params$ version of the CDDRL. Parameters in this version are fitted using a weighted average between those learned in the current and in the previous time steps. That is:

$$CPT_{new} = CPT_{current} * weight + CPT_{previous} * (1 - weight) \tag{24}$$

For each node in the network. It is important to note that both current and previous CPTs are learnt using MLE. The weight is automatically adjusted by an estimation of the amount of change the node has experienced from the last time step (number of changes to its markov blanket) relative to the other nodes. Higher weights will be given to the current CPTs for nodes that experienced more change. The motivation is that if a node changed drastically, this means the past is less relevant.

*4) **Ensemble Methods**:* We introduce the first ensemble method of the CDDRL. Each one of the three CDDRL versions explained above is a base estimator in the ensemble. At every time step we run all three versions separately and save their individual predicted class probabilities. We then aggregate their predicted class probabilities using a weighted average to obtain single predicted class probabilities and use those to predict the observations in the current time step. At first the weights are equally distributed between the versions, but at every time step transition, each weight is multiplied by the normalized AUC performance of the corresponding base estimator on the previous time step, therefore, increasing the influence of better performing versions on the ensemble in future time steps. Algorithm 3 shows a pseudo-code of the $Ensemble\ Methods$ process.

Algorithm 3 Ensemble-Methods algorithm.

input: $Weights, D_{[t]}, Base_{estimators}$

$ProbabilitiesArray \leftarrow []$
$AUCsArray \leftarrow []$
for $estimator = 1$ to length($Base_{estimators}$) **do**
  $ProbabilitiesArray[estimator] \leftarrow$ Predict class probabilities using $estimator$
end for
$Probabilities_{Agg} \leftarrow []$
for $i = 1$ to length($Weights$) **do**
  $Probabilities_{Agg} \leftarrow Probabilities_{Agg} + Weights[i] * ProbabilitiesArray[i]$
end for
$Predictions \leftarrow$ Predict $D_{[t]}$ using $Probabilities_{Agg}$
$AUCsArray \leftarrow$ Calculate AUC for each estimator on $D_{[t]}$ using its predicted class probabilities
for $j = 1$ to length($Weights$) **do**
  $Weights[j] \leftarrow Weights[j] * AUCsArray[j]$
end for
$Weights \leftarrow$ Normalize $Weights$ by dividing each weight by the sum of $Weights$
$return \quad Predictions, Weights$

*5) **Ensemble BNs**:* The second ensemble version of the CDDRL works in a similar manner to the first ensemble version. Here, the base estimators used in the ensemble are the previous BNs learned by the original CDDRL version. The number of BNs in the ensemble is a predefined number. At every time step the currently learned BN contributes 50% to the final prediction of the ensemble, where the existing BNs in the ensemble divide among themselves the other 50% by their weights representing their past performance. The new learned BN enters the ensemble if its AUC performance in the current time step is higher than the worst performing BN in the ensemble, in that case the worst performing BN is removed from the ensemble. Weights updating and prediction aggregation is done in the same manner as in the first ensemble method. The motivation for this ensemble is that when dealing with data streams and concept drifts, past knowledge might be influential e.g., in case of recurring drifts.

In almost all experiments presented in this paper, we show and analyze results for the original CDDRL version as well as for both ensemble methods.

VII. Experiments

In Section VII-A, in three preliminary experiments, we evaluate the detection sensitivity of the original CDDRL to the SPC chart, distance function, and BN properties. In Section VII-B, we test the CDDRL's performance with respect to classification accuracy and correct structure learning under several types of CD, i.e., sudden or gradual and real or virtual. In Section VII-C, we test and compare the CDDRL to other CD algorithms on seven real world datasets.

A. Preliminary Experiments

When using the non-symmetric CS (2) and KL (3) distance functions, there is a need to specify "roles" of the distributions. For CS, we set the CPDs taken from BN_0 as "expected", and those from $\widehat{BN_{[t]}}$ as "observed". Regarding KL, the distributions are defined as "true" (P) and "used" (Q). But since in our case the choice of which is which is inconclusive and both directions are conceivable, i.e., setting BN_0 as P and $\widehat{BN_{[t]}}$ as Q and vice versa are both possible, we test both directions. We denote the divergence as KL when BN_0 is considered as Q and as KL2 when BN_0 is considered as P.

We use three key performance indicators (KPIs): 1) the detection rate, i.e., true positive (TP) rate, which is the proportion of successfully detected changes of the total number of changes that occurred; 2) false alarm (FA), which is the proportion of incorrect change signals, given there was no change; and 3) average number of observations to signal (AOS), which is how quickly a change is detected.

1) ***Change Effect In Different Nodes****:* In this experiment, we evaluated the impact of structural changes (transitions) on a node and its children, as measured by the distance functions. We avoided using SPC because we only wanted to analyze DDPM behavior and not performance. We examined structural transitions with three-node networks, and tested the 600 transitions from each of the 25 possible DAGs to any other. For each transition, 10,000 observations were sampled from BN^*_{pre} and 5,000 from BN^*_{post}. We used a window size of 1,000 observations and a jump size of 100. We divided the nodes into three groups: (1) Nodes that have experienced a change in the transition (e.g., lost or gained a parent), denoted as C (changed) nodes; (2) Nodes that have not experienced a change, but their parents, if existing, experienced a change (e.g., their parent lost or gained a parent), denoted as CoC (child of changed) nodes; and (3) Nodes where neither the node nor its parents, if existing, have experienced any change in the transition, denoted as NC (not changed) nodes.

Figure 4 shows distance measurements for all 600 transitions for (a) C, (b) NC, and (c) CoC nodes. Each of the 3×600 lines in each of the figures represents a node in a particular transition. The solid bold vertical line marks the change point at 10,000 observations, whereas the dashed vertical line marks the point from which all observations in the window are generated from the changed graph, BN^*_{post}. Figure 4(a) shows that for C nodes, the distance before the change is stable and close to zero. Once the window starts to include observations from BN^*_{post}, the measured distance of the C nodes increases and indicates a change, even when the window

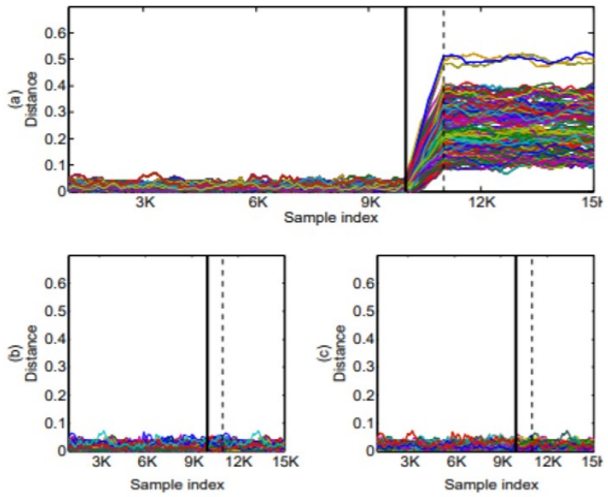


Fig. 4. Distance measures of (a) C, (b) NC, and (c) CoC nodes.

contains a mixed proportion of observations from BN^*_{pre} and BN^*_{post}. After the dashed line, the distance is stable and significantly higher relative to that before the change. Figures 4(b) and 4(c) show that NC and CoC nodes behave the same, with both having small, close to zero and stable distance measurements before and after the change. Figure 4 demonstrates that the change is identified locally in all and only the nodes that have experienced the change.

2) ***Parameter Drift****:* In this experiment, we evaluated the performance of DDPM following a parameter drift, i.e., the structure remains unchanged but the parameters are changed for different node properties, e.g., cardinality and skewness.

We constructed BN^*_{pre} and BN^*_{post} with the same structure, where node X has no parents. We assigned X in BN^*_{pre} with 50 different distributions, divided into five levels of cardinality (dimension) and ten levels of skewness. In BN^*_{post}, each of these 50 distributions was changed with ten levels of change, between 0.01 to 0.30, by subtracting the change level from the highest probability in the distribution and dividing it among the probabilities of the remaining states. Each of the 500 transitions was sampled ten times, producing 5,000 cases for evaluation. Each case was tested with all combinations of SPC charts and distance functions and with five window sizes (100, 250, 500, 1,000, and 2,000) and four jump sizes (25, 100, 250, and 500). For the EWMA (5) and MEWMA (10) SPC charts, a preliminary step was taken to set the best smoothing parameter (λ). Generally, lower values are preferred, and the range [0.1, 0.3] is recommended. For these experiments, we used $\lambda = 0.1$.

a) Parameter-Drift Effect on Performance: Figure 5 shows TP and FA values for all combinations of SPC chart and distance function, showing: 1) CS (2) and TV (1) have the best performance, followed by KL and KL2, where H is the poorest; and 2) MEWMA has the smallest FA rates, but with the poorest TP rates.

b) Dimension Effect: Figure 6 shows the TP and FA rates as a function of variable dimensionality. H is the distance that is most sensitive to node dimension with a dramatic decrease in the TP rate for increasing dimensions. TP for CS and TV

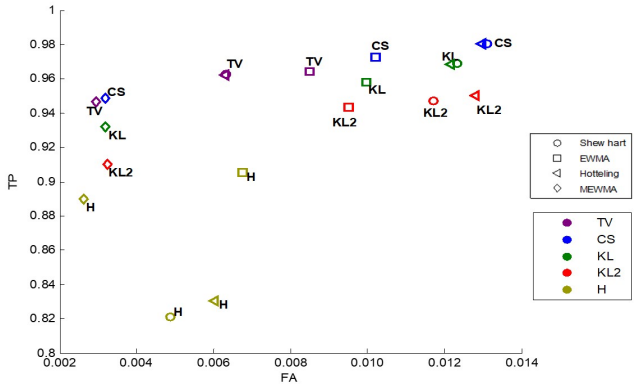


Fig. 5. TP vs FA for every SPC-distance function combination.

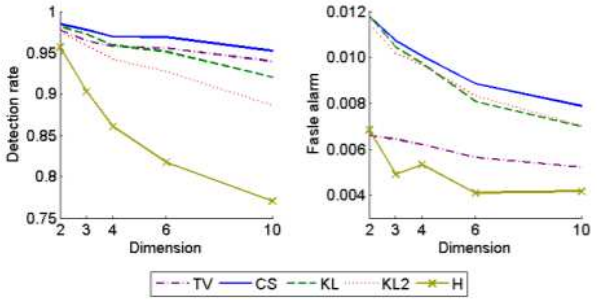


Fig. 6. Dimension effect on TP and FA rates for different distance functions.

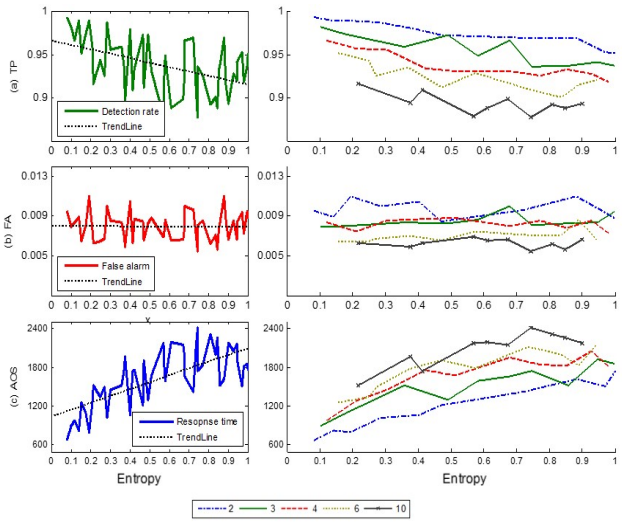


Fig. 7. Entropy effect on performance. (Left) Average values and trends, and (right) values per dimension.

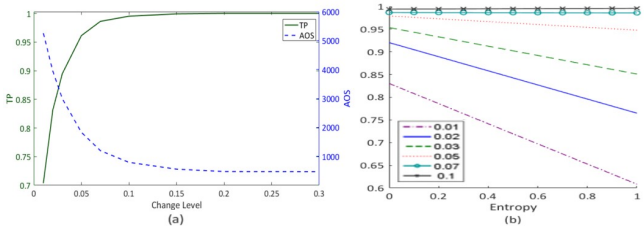


Fig. 8. TP and AOS vs. the change level (left) and entropy (right).

is almost insensitive to the dimension, but not FA.

c) Entropy Effect: Figure 7 shows the entropy effect (achieved by changing the skewness level) for different variable dimensions (right) and on average over the dimensions, together with a trend line (left). While high entropy decreases the TP rate and increases the AOS, it does not affect the FA rate much. Note that the entropy affects all dimensions similarly, meaning that there is no interaction between them. Of course, detection worsens with dimensionality.

d) Change Level Effect: Figure 8(a) shows detection performance as a function of the change level. The TP rate reaches 100% at a change level of 0.15 and from there, AOS is relatively small. Figure 8(b) focuses on the impact of the interaction between the change level and entropy on TP for the levels of change that are smaller than 0.15. Figure 8(b) shows that the TP rates decrease as the entropy increases (as in Figure 6), but the decrease is higher for smaller change levels. That means that there is an interaction between entropy and the change level, and small changes are better detected with skewed distribution, where their significance increases.

*3) **Structural Changes**:* In this experiment, we evaluated DDPM in detecting structural changes: edge removal and addition. Starting from a structure with a single node X having no parents, and gradually adding parents to X up to four, we generated four transitions of edge addition. Then, edges were gradually removed back to the initial network to generate four transitions of edge removal. For BN^*_{pre}, CPDs of X were sampled five times from a Dirichlet distribution for each of four different sets of priors: [99,1], [95,5], [80,20], and [60,40], which represent four levels of skewness. Each added/removed parent was assigned two different distributions: skewed (0.99, 0.01) and not skewed (0.6, 0.4). In BN^*_{post}, CPDs of X were also sampled from a Dirichlet distribution. To generate cases that represent small changes (i.e., more difficult to detect), we used priors based on the data sampled from BN^*_{pre}.

a) General Performance: Figure 9 shows the TP vs. FA rates for each combination of the SPC chart and distance function. CS, KL, and KL2 have the highest TP rates, followed by TV. H (4) continues to have the lowest TP rates, but also with low FA rates. Figure 9 also clearly shows that the performance of the univariate SPC charts outperforms those of the multivariate ones.

b) Parents Distribution Effect: The parents distribution should not affect the node's conditional distribution parameters, but it does affect the number of observations used to estimate these parameters, which in turn affects their variability (and reliability). Figure 10 shows the chart shown in Figure 9; once for skewed parents (on the right side) and once for non-skewed parents (on the left side). Note that "skewed parents" means that only one of X's parents is skewed distributed (0.99, 0.01), and the rest, if exist, are non-skewed distributed (0.6, 0.4). We can see that MEWMA is highly sensitive to the presence of skewed parents, the Hotelling chart has poor performance in both cases, and the univariate procedures are insensitive to skewness.

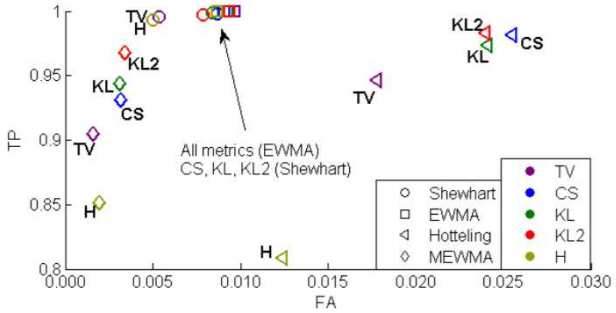


Fig. 9. TP vs. FA for every SPC-distance function combination.

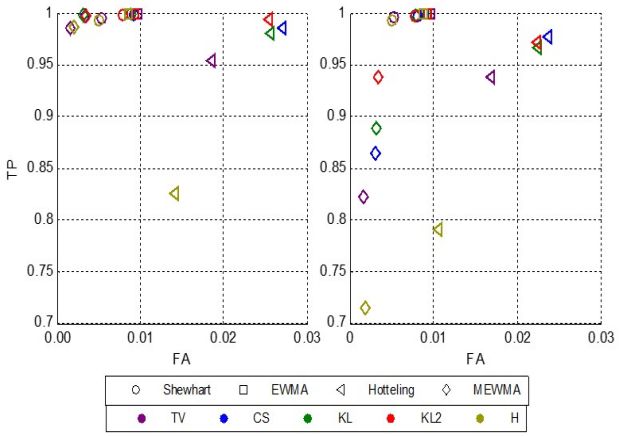


Fig. 10. Performance with non-skewed (left) and skewed (right) parents.

TABLE I
TP RATES FOR EDGE REMOVAL

SPC	TV	CS	KL	KL2	H
Shewhart	0.999	0.998	0.999	1.000	0.998
EWMA	1.000	0.999	1.000	0.999	1.000
Hotelling	0.917	0.971	0.963	0.970	0.793
MEWMA	0.829	0.867	0.890	0.941	0.724

TABLE II
TP RATES FOR EDGE ADDITION

SPC	TV	CS	KL	KL2	H
Shewhart	0.993	0.998	0.998	0.996	0.989
EWMA	1.000	1.000	1.000	1.000	0.999
Hotelling	0.974	0.992	0.984	0.996	0.823
MEWMA	0.980	0.996	0.997	0.994	0.979

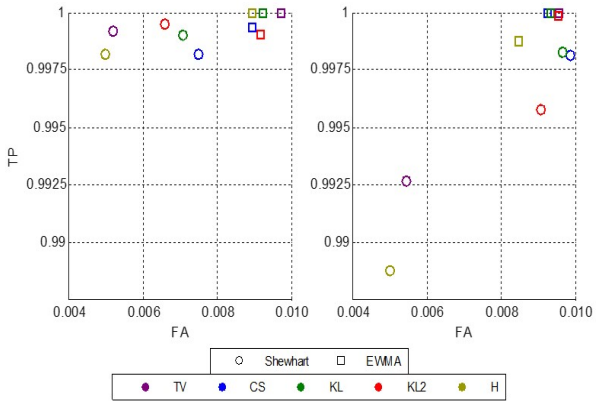


Fig. 11. TP vs. FA for removal (left) and addition (right) of edges for the Shewhart and EWMA procedures.

c) Type of Change: Another issue we wish to test is the type of change, whether it is an addition or removal of an edge. Tables I and II show the TP rates for the case of edge removal and edge addition, respectively. We can see that the multivariate SPC procedures have a lower detection rate for edges that are removed (Table I) than edges that are added (Table II), while the univariate procedures perform similarly in both cases. The MEWMA SPC is highly sensitive to the task, and loses 13.9% on average (over all metrics) of its TP rate when needed to detect removed edges, instead of detecting edge addition. While moving from detecting added edges to detecting removed edges, Hotelling TP rates decrease on average 3%, whereas its FA rates increase on average 0.25%. The Shewhart procedure has slightly better TP (up to 1% on average) and better FA (up to 0.26%) for removing edges than for adding edges. The EWMA procedure is similar, but with a lesser effect. For both learning tasks the EWMA and Shewhart procedures are superior to Hotelling with respect to TP and FA rates. MEWMA is also inferior to the univariate procedures with regards to TP rate, but shows the best FA rates among all procedures. To better analyze the univariate procedures that are superior to the multivariate procedures in almost all criteria, i.e., TP and FA rates and sensitivity to task, the Hotelling and MEWMA procedures are temporarily discarded from the analysis. Figure 11 shows the data in Tables I-II (including FA rates that are not shown in the tables) for the two univariate procedures: Shewhart and EWMA.

The comparison between Shewhart and EWMA is task dependent. Regarding the task of detecting addition of edges, EWMA has better TP rates with similar FA rates for CS, KL, and KL2 metrics, and higher FA rates for H and TV. Regarding the task of detecting removal of edges, EWMA has slightly better TP rates, but also higher FA rates.

To sum up, it can be concluded that the type of change has a different effect on different SPC procedures. The multivariate procedures are more affected by this type and have more difficulties detecting removed edges. The Shewhart procedure, on the other hand, detects removed edges slightly more easily than detecting edge additions. The EWMA procedure is the least sensitive to the type of change.

d) Number of Parents Effect: In this section, we test whether there is a difference between adding (or removing) the first parent and the 4^{th} parent to X. More precisely, we test if there is a meaning (in terms of change detection) to a change as a function of the number of parents a node already has. Following the conclusions from previous sections, we split the analysis and test the cases of adding and removing edges separately.

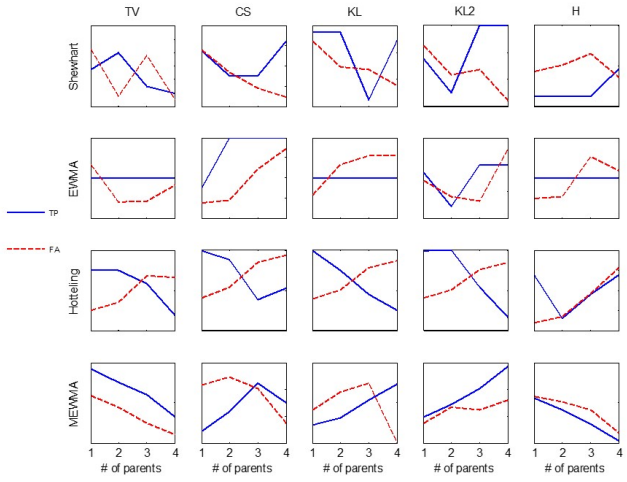


Fig. 12. TP and FA rates in detecting removed edges as a function of the number of parents X already has.

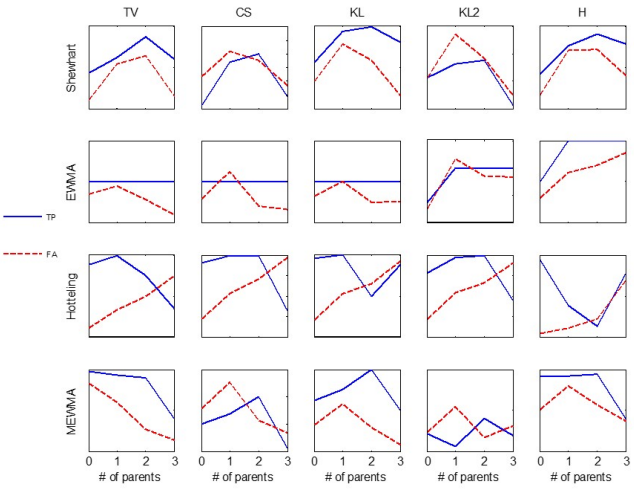


Fig. 13. TP and FA rates in detecting added edges as a function of the number of parents X already has.

Figures 12 and 13 show the TP and FA rates as a function of the number of parents a node has for the cases of removing and adding a single edge, respectively. Each figure presents 20 charts, one for each combination of SPC chart and distance function. Note that for clarity, the charts show no values, but only trends. It can be seen in Figure 12 (removing edges) that there is no clear pattern or trend regarding the number of parents a node has. There are different and opposite trends with different sub-charts, even within a particular metric or procedure.

Figure 13 (adding edges) looks similar. While one can find local patterns regarding some particular property (e.g., when using a Hotelling chart, FA increases with the number of parents), it is difficult to point at a general trend. Figures 12 and 13 suggest that the detection and false alarm rates do not depend on the number of parents a node has.

To sum this section up, we empirically evaluated the CDDRL with the different distance functions and SPC charts and with different properties of the data, i.e., entropy, nodes' dimensions, and types and magnitude of change. We conclude that the CDDRL successfully discovers both parameter and structural changes in a BN for all and only those nodes that have truly experienced change. Furthermore, we conclude that the univariate procedures are superior to the multivariate ones, where EWMA is the least sensitive to task and gains the highest TP rates. The CS and TV distance measures show the most promising results compared to the other measures regarding sensitivity to parents' dimensionality and TP rates, where CS achieves higher TP rates than TV, and TV achieves lower FA rates than CS. Therefore, for the following experiments, we use the CS distance function with the EWMA SPC chart.

B. Synthetic CD Experiments

First, we demonstrate using synthetic data that the CDDRL - both the original and the two ensemble versions, although not designed for a supervised setting, is comparable to other streaming classification methods. Second, we exhibit using data sampled from popular BNs the CDDRL's ability to detect and adapt to changes in the distribution in the more general case of unsupervised CD, an ability most existing approaches do not have. The ensemble versions are not well fitted for the unsupervised experiment's settings - learning a single BN at every time step - thus, only the original version was tested here. Since data are sampled from given distributions, we repeated each experiment 10 times and report the averages. For the CDDRL, we used a jump $S = 1$ time step, a window $W = 2$ time steps (Section IV-A1), and the first four time steps in the stream to calculate CL and $\hat{\sigma}$ for the SPC test. We note again the formula used by the EWMA SPC to compute the upper control limit for change (UCL):

$$UCL = CL + L * \hat{\sigma} * \sqrt{\lambda/(2-\lambda)} \tag{25}$$

In the following experiments, we set $\lambda = 0.2$ and $L = 3$.

1) ***Supervised CD****:* Since both are important, we present experiments for stream learning for classification under real and virtual CD. For comparison, we used the state-of-the-art streaming classification algorithms: KNN-ADWIN [15] (a combination of ADWIN and KNN), AWE [21], DWM [22], LNSE [4], SRP [3], SAM-KNN [12], VFDR [16], RSLVQ [20], OBC [17] and ARF [18]. All are widely used for streaming classification and described in Section II. We used the Scikit-multiflow [39] package and kept most of the algorithms with their default hyper-parameters. Since we wanted to compare the CDDRL to different drift detection methods, we set the drift detector of the VFDR algorithm to EDDM [14] and that of the ARF algorithm to KSWIN [7]. In addition, the gradient descent method of the RSLVQ algorithm was changed to Adadelta with momentum of 0.9 to make it adequate for streaming data. We used 10 trees instead of the default value of 100 for the SRP algorithm since 100 trees caused a very long running time. A recent less known algorithm named NN-UN [19] was also tested, NN-UN uses a deep neural network coupled with an ADWIN drift detector that monitors predictions' uncertainties to deal with CD. We assume data arrive in batches of one time step, and update the models after each batch arrives. For example, at the beginning

TABLE III
ACCURACY IN SUDDEN CD ($\eta = 500$)

Algorithm	Average	Std	Min	RT
CDDRL-Original	0.9955	0.034	**0.683**	**4**
CDDRL-Ensemble$_{Methods}$	**0.9960**	**0.032**	**0.683**	**4**
CDDRL-Ensemble$_{BNs}$	0.9912	0.057	0.510	5
KNN-ADWIN	0.9890	0.060	0.579	6
AWE	0.9892	0.063	0.579	**4**
DWM	0.9915	0.053	0.646	**4**
LNSE	0.9667	0.120	0.445	13
SRP	0.9936	0.045	**0.683**	**4**
SAM-KNN	0.9916	0.053	0.595	**4**
VFDR	0.9893	0.052	0.660	8
RSLVQ	0.8772	0.125	0.622	*100*
OBC	0.9919	0.049	0.678	5
ARF	0.9919	0.047	**0.683**	5
NN-UN	*0.7376*	*0.264*	*0.421*	*100*

TABLE IV
ACCURACY IN GRADUAL CD ($\eta = 5000$)

Algorithm	Average	Std	Min	RT
CDDRL-Original	0.9721	0.064	0.702	**21**
CDDRL-Ensemble$_{Methods}$	**0.9747**	**0.057**	**0.712**	**21**
CDDRL-Ensemble$_{BNs}$	0.9681	0.076	0.693	24
KNN-ADWIN	0.9676	0.073	0.697	26
AWE	0.9599	0.091	0.690	26
DWM	0.9654	0.082	0.691	23
LNSE	0.9472	0.119	0.566	25
SRP	0.9707	0.066	**0.712**	24
SAM-KNN	0.9699	0.069	0.700	23
VFDR	0.9466	0.100	0.614	35
RSLVQ	0.8575	0.151	0.536	*100*
OBC	0.9610	0.081	0.629	27
ARF	0.9669	0.068	**0.712**	28
NN-UN	*0.7919*	*0.214*	*0.468*	*100*

of the process, we predict the class in the first time step, then update the model for the next time step.

For the following experiments, there is no known BN_0 that represents BN^*_{pre} of the stable process. Thus, we learned it directly from the data using the first four time steps (which are considered stable for all of the experiments in this section). The structure was learned using min-max hill climbing [MMHC; [40]] and the parameters (CPTs) using MLE.

a) Real CD: To test the performance of the CDDRL (the original version as well as both ensemble versions) as a streaming-classification algorithm in real CD, we used the STAGGER benchmark [41]. STAGGER consists of three categorical features: size (small/medium/large), shape (circle/square/triangle), and color (red/blue/green). There are three concepts/classification functions (CFs). For the following experiments, we used two CFs, one returns 1 if the size is small and the color is red, and the second returns 1 if the color is green or the shape is circular. CD was introduced by changing the classification function, and it is real CD since $P(y|X; function1) \neq P(y|X; function2)$.

We sampled 20,000 examples using Scikit-multiflow [39], where the probability that the t^{th} example belongs to the second concept (CF_2) is,

$$P(CF_2) = \begin{cases} \frac{1}{1+\exp^{-4*(t-p)/\eta}}, & \text{if } t \geq p \\ 0, & \text{else} \end{cases}$$

where p is the drift position, and η is the drift width (how sharply the drift rises). We consider each 100 examples as a time step, resulting in 200 time steps (100 before and 100 after the change).

We used a drift width $\eta = 500$ and $\eta = 5000$ of examples for sudden and gradual CD, respectively. We calculated the accuracy at each time step (we sampled balance classes; hence, accuracy is a proper evaluation metric): the average accuracy, the standard deviation (Std) of the accuracy, the minimum accuracy, and the recovery time (RT), which we defined as how many time steps it takes the algorithm to recover and reach 95% accuracy after the drift occurred.

Table IV shows that the RSLVQ and NN-UN algorithms were drastically inferior to all other algorithms, where RT of 100 represents that these algorithms never reached back an accuracy over 95% after the drift has occurred. A possible explanation for their bad performance is that as the authors of the NN-UN algorithm stated, NN-UN will not detect well real CD when it appears without a virtual CD as well. The RSLVQ has no active drift detection mechanism, but rather uses a momentum based SGD to deal with the streaming data. The rest of the algorithms achieved similar very good results in terms of the average accuracy (besides the LNSE, which was significantly inferior), the ensemble methods version of the CDDRL was slightly better than the other algorithms. These results are not trivial, since the others were designed for streaming-classification, whereas the CDDRL is for general (unsupervised) CD detection. In addition, the original CDDRL as well as the ensemble method CDDRL versions achieved the least noisy results over the stream. These two versions also had the best performance in terms of RT together with the AWE, DWM, SRP and SAM-KNN algorithms with only four time steps to return to high accuracy. However, at the change point, where all of the algorithms dropped their performance, AWE, DWM and SAM-KNN dropped to a lower (minimum) accuracy compared to both CDDRL's versions and the SRP. The ensemble BNs version of the CDDRL was slightly worse than the other two CDDRL's versions in this task and had an average performance compared to the other competitors.

Table III shows similar results to those achieved for the sudden CD. The original CDDRL and the ensemble methods CDDRL outperformed the other algorithms in terms of average accuracy, Std and RT. The ensemble BNs version of the CDDRL once again was inferior compared to the other two CDDRL versions and had an average performance compared to the other competitors.

b) Virtual CD: Since virtual CDs are not as common as real CDs, we designed an experiment to test the algorithms' performance in virtual CD. In this experiment, we sampled 30 time steps, each consisting of 1,000 samples. We sampled three input variables according to the following distributions:

$$X_1 \sim \begin{cases} B(p = 0.2, n = 4), & \text{if } t < d \\ B(p = min(1, 0.2 + 0.01 * t), n = 4), & \text{else} \end{cases}$$

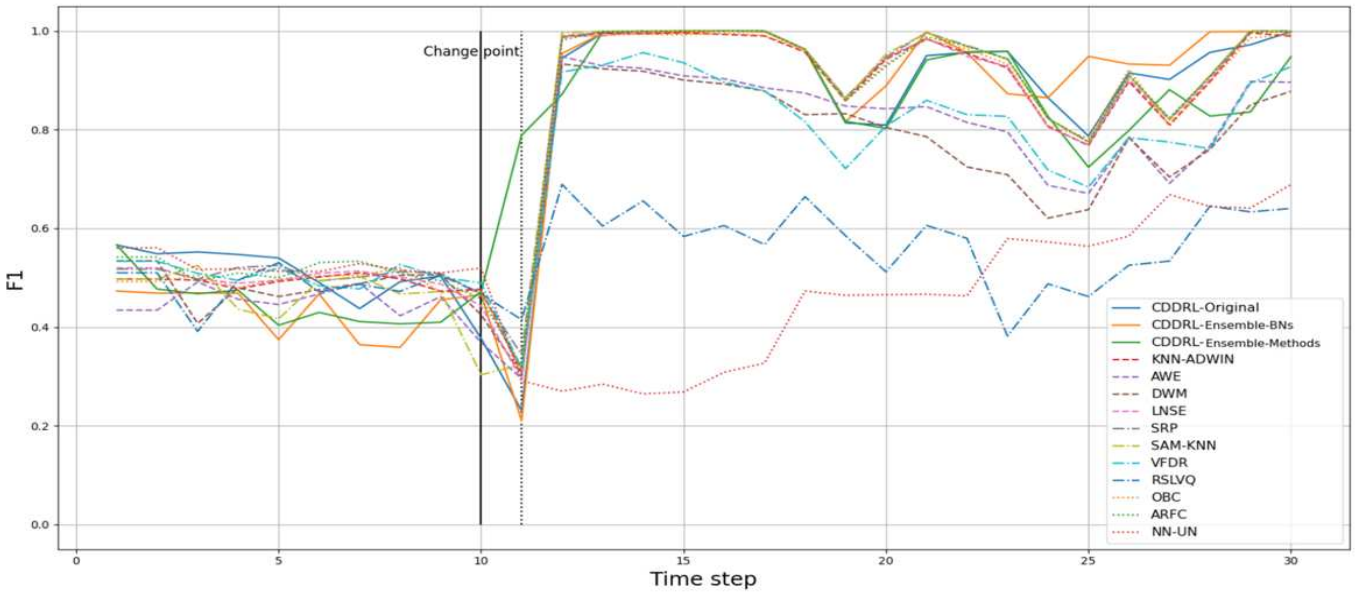


Fig. 14. Virtual CD F1 results for CDDRL and all competing algorithms.

$$X_2 \sim \begin{cases} BOL(\lambda = 1, N = 3), & \text{if } t < d \\ BOL(\lambda = \frac{1}{1+0.1*(t-d+1)}, N = 3), & \text{else} \end{cases}$$

$$X_3 \sim \begin{cases} BB(a = 2, b = 1, n = 4), & \text{if } t < d \\ BB(a = 2 + 0.1 * (t - d + 1), b = 1, n = 4), & \text{else} \end{cases}$$

where d is the change point, p and n are the parameters of the binomial (B) distribution, λ and N are the parameters of the Boltzmann (BOL) distribution, and a, b, and n are the parameters of the beta-binomial (BB) distribution.

With all three variables, the change in distribution caused them to gain higher values with a higher probability. This is reasonable if we think of them, for example, in terms of size, which naturally grows with time. Thus, for X_1, we increased p (the probability of success in one trial for a binomial variable), since the expected value for the number of successes is $n * p$ (where n is the number of trials), and as p grows, $n * p$ also grows for fixed n. Similar ideas have been applied for X_2 and X_3, with their own distributions.

The target variable was uniformly sampled in $[0, 1]$ until the change point (i.e., in the first 10 time steps). In the tenth time step, a real CD has been applied, and the target variable is 1 if one of the following conditions is fulfilled:

$$\{X_2 = min(X_2)\} \ \& \ \{X_3 \leq Q_1[X_3]\} \text{ or}$$

$$\{X_2 = min(X_2) + 1\} \ \& \ \{(X_1 > \overline{X_1} + 1 \ || \ X_1 < \overline{X_1} - 1)\}$$

and 0 otherwise, where $Q1[X]$ is the 0.25th quantile of X. The indicators for all of the variables are computed at each time step (e.g., $min(X_2)$ and $Q_1[X_3]$). The first change is real CD since $p(y|x_1, x_2, x_3)$ changes from uniform (which is independent in x_1, x_2, x_3) to dependent in x_1, x_2, x_3, with the rules defined above. However, from time step 11 on, the distribution of x_1, x_2, x_3 keeps changing, whereas $p(y|x_1, x_2, x_3)$ remains the same. Hence, from time step 11 on, virtual CD is introduced. The experiment design resulted in an imbalance classification task; hence, we report here the F1 measure.

Figure 14 shows that in the first 10 time steps, all of the algorithms achieved F1 of about 0.5, resulting from random guessing of the target variable as either 1 or 0 (it was randomly assigned in this period). Since window = 2, at time step 11, the data used to train the algorithms is mixed (half of it is drifted and the other is not), therefore all algorithms drop their performance except for the ensemble methods version of the CDDRL that managed to significantly improve its performance at time step 11, but is too still adjusting to the changed data. From time step 12 on, all of the algorithms improve their performance, whereas the ensemble BNs CDDRL version is superior compared to all other algorithms and achieves a higher or equal F1 in almost all time steps. The original CDDRL is the second best algorithm and is inferior to the ensemble BNs version starting from day 23. The ensemble methods version although showing a strong performance at time step 11, falls in performance compared to the other two CDDRL versions starting from day 23 but is still at least as good as most of the other competing algorithms.

*2) **Unsupervised CD**:* Here, we measure the ability of the CDDRL to detect a change in an unsupervised setting by adapting the BN structure to the change. We used the popular Asia and ALARM BNs to create this setting, where the Asia BN is relatively small (7 nodes) and Alarm is bigger (37 nodes).

For the following experiments, assuming a single change point, we denote again BN^*_{pre} as the stable process generating model and BN^*_{post} as the generating model following the change. BN_0 is the network learned using historical data representing the stable process, i.e., the true underlying network

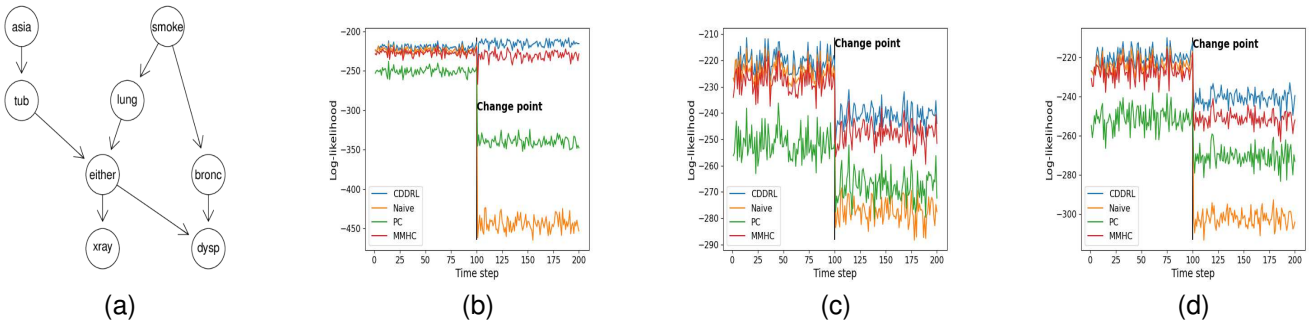


Fig. 15. Asia BN Experiments: (a) shows the Asia BN before any change was made, (b), (c) and (d) show all algorithms' log-likelihood performances on edge addition, removal, and reversal respectively.

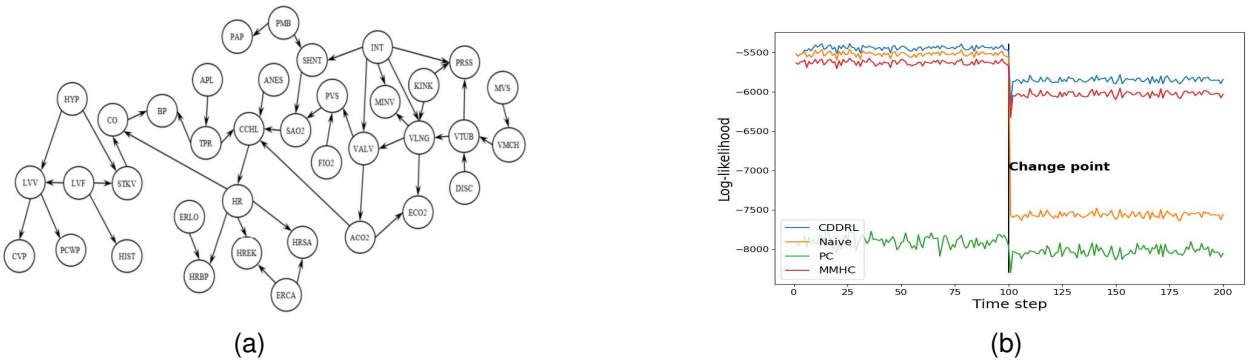


Fig. 16. Alarm Experiment: (a) shows the Alarm BN before any change was made and (b) shows all algorithms' log-likelihood performances on edge addition, deletion, and reversal simultaneously.

BN^*_{pre}.

The CDDRL is an approach for locally updating the current model at each time step. For comparison, we use two alternative approaches: 1) at each time step, learn a new model from scratch, and 2) the initial model is always true; therefore, use it at each time step. As a representative of the first approach, we use the two structure learning algorithms, PC [42] and MMHC [40]. For these two approaches, at each time step, we use the latest window examples to learn the model structure (this what static models such as PC and MMHC would normally do), and the MLE to learn its parameters. The main difference is that the CDDRL learns guided local changes, and those approaches learn the entire network from scratch at every time step. For the second naive approach, we use BN_0 as the generative model along all time steps.

For evaluation, we use the log-likelihood, which measures how well the model describes the data and is commonly used for evaluating unsupervised paradigms and particularly BNs. As mentioned before, in this experiment, only the original version of the CDDRL was tested.

Our implementation was based on Matlab 6.5 and used the Causal Explorer package [43] for the PC and MMHC.

a) Asia BN: This network was built to describe the relationships between lung diseases (tuberculosis (tub), lung cancer, or bronchitis ($bronc$)), visits to Asia, smoking ($smoke$), and dyspnea ($dysp$) [44]. It consists of 7 binary variables and 8 edges (Figure 15a).

For the following experiment, 10,000 examples were sampled from BN^*_{pre} (the original BN; Figure 15a), and the next 10,000 from BN^*_{post} (the network after a sudden CD). We consider each 100 examples as a time step, resulting in 200 time steps. We examine several scenarios of distribution changes (i.e., add/remove/reverse edge).

Add Edge: In this experiment, BN^*_{post} is the original BN with an additional edge from $smoke$ to tub.

Remove Edge: In this experiment, BN^*_{post} is the original BN where the edge from $bronc$ to $dysp$ is missing.

Reverse Edge: In this experiment, BN^*_{post} is the original BN where the edge from $bronc$ to $dysp$ is reversed.

As we can see, in all three cases, until the change point, all three—the CDDRL, MMHC, and the Naive approach—achieved similar results, whereas the PC is inferior to these approaches, and CDDRL is slightly superior to the MMHC and Naive algorithms. In the scenario of adding an edge (Figure 15b), while the performance of both the PC and the Naive approach worsen, the drop in performance of the Naive is more significant. The CDDRL and the MMHC are stable after the change point, where the CDDRL outperforms the MMHC. In both scenarios of edge removal (Figure 15c) and edge reversal (Figure 15d), the performance of all of the approaches dropped after the change; the Naive approach drop was the most significant, followed by PC, and MMHC, whereas the CDDRL achieved the best results.

b) Alarm BN: The network used in the evaluation is the Alarm network [45] designed to provide an alarm system for patient monitoring, and it consists of 37 discrete variables and 46 edges (Figure 16a).

Since Alarm is a bigger network than Asia, we sampled 50,000 examples from BN^*_{pre}, and the next 50,000 from BN^*_{post}. We consider each 500 examples as a time step, resulting in 200 time steps.

In this experiment, BN^*_{post} is the original BN with several structural changes: 1) an edge from PMB to APL added, 2) an edge from TPR to BP removed, and 3) an edge from ERLO to HRBP reversed.

As we can see (Figure 16b), until the change the CDDRL and the Naive approach achieved similar results, although the CDDRL was slightly better. The MMHC was inferior compared to those two, but it also achieved reasonable results. The PC was significantly inferior compared to the other three. After the change point, the Naive approach significantly decreased performance but was still better than the PC algorithm, which was inaccurate along the stream, but stable. The performance of both the MMHC and the CDDRL dropped, although the latter outperformed the former.

C. Real Life Experiments

In these experiments we test and evaluate the CDDRL on seven real world datasets. Here, the ensemble versions were evaluated as well as the original version.

Methodology The evaluation process of all seven experiments was that at every time step, all algorithms predict the current data (prediction of class or fertilizer/water treatment), and then the predicted data is used to update all algorithms for prediction of the next time step.

Datasets: Four of them are known datasets that include natural or manual induced CD:

1) ***Electricity****:* a widely used dataset described by [46]. The data is collected from the Australian New South Wales Electricity Market. In this market, prices are not fixed and are affected by demand and supply of the market. They are set every five minutes. Electricity transfers to/from the neighboring state of Victoria were done to alleviate fluctuations. The class label identifies the change of the price (UP or DOWN) in New South Wales relative to a moving average of the last 24 hours.

2) ***AWS Spot Price****:* Amazon Web Services [AWS; [47]] provides virtual computing environments via their EC2 service that can be configured with up to 16 GPUs. You can launch instances with your favourite operating system, select preconfigured instance images or create your own. However, you can request Spot Instance Pricing. Which basically charges you for the spot price that is in effect for the duration of your instance running time. They are adjusted based on long-term trends in supply and demand for Spot instance capacity. Our goal here was to predict Spot pricing for two regions in Central California between March and May 2017 [48].

3) ***Usenet****:* this dataset is based on the 20 newsgroups collection. They simulate a stream of messages from different newsgroups that are sequentially presented to a user, who then labels them as interesting or junk, according to his/her personal interests. The user's interest changes among the batches (we have five batches each contains 300 instances), there are three newsgroups articles; *Medicine* is the user's interest in the first, third and fifth batches, whereas *Space* and *Baseball* are the interesting topics in the second and fourth batches [49].

TABLE V
REAL WORLD DATASETS OVERVIEW

Dataset	# Insts.	# Attrs.	# Time Steps	# Classes
Electricity	45312	8	83	2
AWS Spot Price	199800	6	54	3
Usenet	1500	99	15	2
NYC Subway Traffic	105000	17	70	3

4) ***NYC Subway Traffic****:* this dataset holds data about hourly passenger traffic within the NYC subway system with each station having its regular seasonal, daily and weekly fluctuations. The dataset includes the number of subway station entries, at 4 hour intervals, for 469 subway stations from Feb. 4th 2017 to Aug. 13th 2021. In addition to 5 time and stations' attributes, each of the 469 stations in the dataset was referenced to one of 51 neighborhoods, each associated with 12 aggregated financial and demographic variables. Our task here was to predict for every time step (month) whether the number of entries will be low (entries in the 0-1/3 percentile), medium (entries in the 1/3-2/3 percentile) or high (entries above the 2/3 percentile) using previous time steps' data [50].

Table V gives an overview of the four datasets.

5) ***Agriculture Experiments****:* In addition we performed three real-life scenarios experiments in the agriculture field. The goal of the experiments was to detect manually induced stress in banana plants that were grown in a greenhouse. In each of the three experiments we grew a different number of banana plants for a different number of days. We divided the plants into four equally sized groups and treated each with a different fertilizer/water regime, intentionally causing plants to get fertilizer/water stressed in different levels. Every experiment started with a few days where all plants received the same treatment with the recommended amount of fertilizer/water the plants need. We relate to these days as the stable period and use them to learn BN_0. In each experiment, we collected different data about the plants from cameras and sensors. Table VI sums up the settings and objectives of each experiment.

Evaluation Metrics: Since several experiments had the task of binary class predictions (Electricity, Usenet, Fertilizer Stress, Water Stress) and the rest had the task of multiclass predictions (AWS Spot Price, NYC Subway Traffic, Fertilizer+Water Stress), we used different evaluation metrics for each task. For the binary class predictions task we used the following metrics: False Alarm Rate (FA) defined as the number of false positives out of the total ground truth negatives, F1-Pos (when class 1 is treated as the positive class), F1-Neg (when class 0 is treated as the positive class), F1-Avg (the average of F1-Pos and F1-Neg), Geometric Mean (G-Mean), Balanced Accuracy (B-Accuracy), Area Under the ROC Curve (AUC) (except for AWE, DWM, SAM-KNN and RSLVQ that cannot produce class probabilities, thus cannot be evaluated using AUC). For the multiclass predictions task we used: Weighted Precision, Recall, F1 and AUC, Kappa and

TABLE VI
A SUMMARY OF THE THREE STRESS EXPERIMENTS, NUMBER OF PLANTS, STABLE DAYS AND TOTAL GROWING DAYS ARE PRESENTED AS WELL AS THE DATA COLLECTED AND THE OBJECTIVE FOR EACH EXPERIMENT.

Criterion	Fertilizer Stress	Fertilizer+Water	Water Stress
# of Plants	201	120	188
Stable Days	7	4	13
Total Days	28	17	41
Treatment Regime	A - 200%, B - 100%, C - 67%, D - 0%	A-100%, B-80%, C-60%, D-40% (of water and fertilizer)	A - 100%, B - 80%, C - 60%, D - 40%
Data Collected	RGB images - Features of size and color (10 features)	Sensory data - SPAD (color), Temperature and Water left in the plant's tray (4 features)	RGB, Thermal, Depth images - Features of size, color, temp and height (10 features)
Objective	D Vs. All	Multiclass - A, B, C, D	D Vs. All

Algorithm	FA	F1-Pos	F1-Neg	F1-Avg	G-mean	B-Accuracy	AUC
CDDRL-Original	0.19	0.59	0.75	*0.67*	0.64	*0.71*	*0.77*
CDDRL-Ensemble-Methods	0.14	*0.56*	0.78	*0.67*	*0.62*	*0.71*	0.79
CDDRL-Ensemble-BNs	0.14	**0.69**	**0.82**	**0.75**	**0.73**	**0.76**	**0.86**
KNN-ADWIN	0.19	0.65	0.78	0.72	0.70	0.73	0.80
AWE	0.16	0.63	0.79	0.71	0.68	0.73	-
DWM	0.12	0.60	0.80	0.70	0.66	0.72	-
LNSE	0.21	0.68	0.78	0.73	0.72	0.74	0.85
SRP	0.17	0.67	0.80	0.73	0.72	0.74	0.83
SAM-KNN	0.15	0.60	0.79	0.69	0.66	*0.71*	-
VFDR	0.19	0.62	0.76	0.69	0.66	*0.71*	0.81
RSLVQ	*0.29*	0.65	*0.72*	0.69	0.67	*0.71*	-
OBC	0.21	0.64	0.77	0.71	0.70	0.72	0.80
ARF	0.17	0.68	0.80	0.74	0.72	0.75	0.85
NN-UN	**0.11**	0.58	0.80	0.69	0.65	*0.71*	0.81

Fig. 17. Electricity experiment results

Algorithm	W-Precision	W-Recall	W-F1	W-AUC	Kappa	Matthews
CDDRL-Original	**0.95**	0.94	0.94	**0.96**	**0.94**	**0.94**
CDDRL-Ensemble-Methods	0.94	0.94	0.94	0.95	0.92	0.92
CDDRL-Ensemble-BNs	**0.95**	**0.95**	**0.95**	**0.96**	**0.94**	**0.94**
KNN-ADWIN	0.74	0.74	0.73	0.87	0.61	0.62
AWE	*0.47*	0.62	0.51	-	0.42	0.50
DWM	*0.47*	0.60	0.50	-	0.39	0.47
LNSE	0.94	0.94	0.94	**0.96**	0.93	0.93
SRP	0.90	0.89	0.89	0.93	0.85	0.85
SAM-KNN	0.70	0.70	0.68	-	0.54	0.55
VFDR	0.66	0.55	*0.49*	*0.71*	0.32	0.37
RSLVQ	0.53	*0.54*	0.52	-	*0.30*	*0.31*
OBC	0.79	0.79	0.79	0.90	0.68	0.69
ARF	0.92	0.91	0.91	0.95	0.89	0.89
NN-UN	0.88	0.85	0.84	0.91	0.78	0.79

Fig. 18. AWS Spot Price experiment results

Algorithm	FA	F1-Pos	F1-Neg	F1-Avg	G-mean	B-Accuracy	AUC
CDDRL-Original	0.37	0.52	0.57	0.54	0.46	0.59	0.61
CDDRL-Ensemble-Methods	0.39	0.57	0.55	0.56	0.52	0.60	0.63
CDDRL-Ensemble-BNs	0.29	0.45	0.57	0.51	0.46	0.57	0.57
KNN-ADWIN	0.48	0.49	0.49	*0.49*	0.49	*0.54*	0.55
AWE	0.27	0.55	**0.66**	**0.60**	0.57	0.62	-
DWM	0.47	0.58	0.55	0.56	0.55	0.59	-
LNSE	0.36	*0.43*	0.55	*0.49*	0.46	*0.54*	*0.54*
SRP	0.49	0.58	0.48	0.53	0.46	0.58	0.61
SAM-KNN	0.42	0.49	0.53	0.51	0.41	0.57	-
VFDR	*0.59*	**0.67**	0.50	0.59	**0.58**	**0.63**	**0.66**
RSLVQ	0.38	0.44	0.54	0.50	0.45	0.55	-
OBC	0.50	0.53	0.51	0.52	0.54	0.55	0.55
ARF	0.49	0.54	*0.47*	0.50	*0.40*	0.56	0.55
NN-UN	**0.22**	0.46	0.65	0.55	0.53	0.60	0.61

Fig. 19. Usenet experiment results

Mathews coefficients. All evaluation metrics are built to deal with imbalanced data, which is the case in most of our datasets. A detailed explanation of the different evaluation metrics we used can be found in [[51], [52]].

Results: Figures [17 - 23] show the experiments' average results (over all time steps) of the three CDDRL versions (original and two ensemble versions) and all competing algorithms. In bold is the best performing algorithm for the specific metric and in italic + underscore is the worst performing one. Cells are colored in a green-red color range, where the darkest green and darkest red fill the best and worst performing algorithms respectively, separately for each metric.

Algorithm	W-Precision	W-Recall	W-F1	W-AUC	Kappa	Matthews
CDDRL-Original	0.83	0.82	0.82	0.94	0.71	0.71
CDDRL-Ensemble-Methods	**0.85**	**0.84**	**0.84**	**0.95**	**0.75**	**0.76**
CDDRL-Ensemble-BNs	0.82	0.79	0.80	0.92	0.68	0.68
KNN-ADWIN	0.81	0.81	0.80	0.93	0.69	0.69
AWE	0.60	0.64	0.58	-	0.42	0.45
DWM	*0.57*	*0.60*	*0.55*	-	*0.38*	0.41
LNSE	0.81	0.81	0.81	0.94	0.70	0.70
SRP	0.75	0.75	0.74	0.89	0.60	0.61
SAM-KNN	0.82	0.82	0.81	-	0.71	0.71
VFDR	0.60	0.61	0.56	*0.78*	*0.38*	*0.40*
RSLVQ	0.63	*0.60*	0.59	-	0.39	0.41
OBC	0.80	0.79	0.79	0.92	0.67	0.67
ARF	**0.85**	**0.84**	**0.84**	**0.95**	**0.75**	0.75
NN-UN	0.78	0.74	0.72	0.90	0.58	0.59

Fig. 20. NYC Subway Traffic experiment results

Algorithm	FA	F1-Pos	F1-Neg	F1-Avg	G-mean	B-Accuracy	AUC
CDDRL-Original	*0.06*	0.88	0.95	0.92	**0.92**	**0.92**	0.95
CDDRL-Ensemble-Methods	**0**	0.88	**0.97**	**0.93**	0.90	0.91	0.95
CDDRL-Ensemble-BNs	**0**	0.84	0.96	0.90	0.86	0.88	0.95
KNN-ADWIN	0.02	0.87	0.96	0.92	0.89	0.90	0.96
AWE	0.02	0.88	**0.97**	**0.93**	0.91	**0.92**	-
DWM	0.01	0.77	0.95	0.86	0.79	0.85	-
LNSE	0.02	**0.90**	**0.97**	**0.93**	**0.92**	**0.92**	0.96
SRP	**0**	0.86	**0.97**	0.91	0.88	0.90	0.97
SAM-KNN	**0**	0.58	0.93	0.76	0.59	0.77	-
VFDR	0.02	**0.90**	**0.97**	**0.93**	0.91	**0.92**	0.97
RSLVQ	**0**	0.76	0.95	0.86	0.79	0.85	-
OBC	**0**	0.83	**0.97**	0.90	0.85	0.90	**0.98**
ARF	**0**	0.82	0.96	0.89	0.84	0.88	0.97
NN-UN	0.02	*0.47*	*0.90*	*0.69*	*0.55*	*0.70*	*0.86*

Fig. 21. Fertilizer Stress experiment results

Algorithm	W-Precision	W-Recall	W-F1	W-AUC	Kappa	Matthews
CDDRL-Original	0.83	0.85	0.81	0.96	0.80	0.82
CDDRL-Ensemble-Methods	0.82	0.84	0.80	0.89	0.79	0.81
CDDRL-Ensemble-BNs	0.79	0.84	0.80	0.90	0.79	0.82
KNN-ADWIN	0.81	0.81	0.79	0.92	0.75	0.76
AWE	0.68	0.74	0.69	-	0.66	0.68
DWM	0.82	0.81	0.77	-	0.74	0.77
LNSE	0.81	0.87	0.83	0.96	0.82	0.84
SRP	0.82	0.88	0.84	0.93	0.84	0.86
SAM-KNN	0.76	0.77	0.74	-	0.69	0.71
VFDR	*0.43*	*0.54*	*0.45*	*0.72*	*0.39*	*0.43*
RSLVQ	0.59	0.57	0.50	-	0.43	0.46
OBC	0.81	0.80	0.78	0.92	0.74	0.75
ARF	**0.91**	**0.90**	**0.87**	0.96	**0.86**	**0.88**
NN-UN	0.89	0.89	**0.87**	**0.98**	0.85	0.86

Fig. 22. Fertilizer + Water Stress experiment results

Algorithm	FA	F1-Pos	F1-Neg	F1-Avg	G-mean	B-Accuracy	AUC
CDDRL-Original	0.17	0.49	0.84	0.66	0.62	**0.69**	0.74
CDDRL-Ensemble-Methods	0.17	0.48	0.83	0.65	0.61	0.68	0.75
CDDRL-Ensemble-BNs	0.17	**0.5**	0.84	**0.67**	**0.63**	**0.69**	**0.76**
KNN-ADWIN	0.12	0.45	0.85	0.65	0.57	0.65	0.7
AWE	0.11	0.38	0.84	0.61	0.47	0.63	-
DWM	0.23	0.49	0.79	0.64	0.61	0.66	-
LNSE	0.19	0.43	0.81	0.62	0.57	0.63	0.72
SRP	0.19	0.49	0.79	0.64	0.57	0.66	**0.76**
SAM-KNN	0.06	0.4	0.86	0.63	0.48	0.64	-
VFDR	0.2	0.47	0.81	0.64	0.58	0.67	0.71
RSLVQ	*0.46*	0.48	*0.58*	*0.53*	0.53	0.63	-
OBC	0.39	0.48	0.66	0.57	0.56	0.64	0.71
ARF	0.31	0.47	0.69	0.58	0.53	0.64	0.72
NN-UN	**0.02**	*0.23*	**0.87**	0.55	*0.32*	*0.57*	*0.65*

Fig. 23. Water Stress experiment results

Generally speaking, it can be observed that in most of the experiments all three versions of the CDDRL are at the top half of the algorithms in terms of general performance (they are colored in green shades), and there is at least one version that is the best performing algorithm. This is true for the AWS Spot Price experiment where the original version and the ensemble BNs version are superior to all other competitors, the ensemble methods version shows the best performance in the NYC Subway Traffic experiment, the original version and the ensemble methods are among the best performers in the Fertilizer Stress experiment together with the LNSE, AWE and VFDR algorithms, and the ensemble BNs version is superior to all others in the Water Stress experiment. Regarding the Electricity experiment, all algorithms present similar results where the original and the ensemble methods versions have the

worst results, however the ensemble BNs version shows the best results. Regarding the Usenet and the Fertilizer + Water Stress experiments, all three CDDRL versions show average results compared to all competitors - AWE, DWM, VFDR and NN-UN are superior in the Usenet experiment and ARF, NN-UN, LNSE and SRP are superior in the Fertilizer + Water Stress experiment.

Statistical Analysis: Since we have seven experiments with six or seven performance metrics in each, it is hard to conclude on a reliable overall ranking between the algorithms. To deal with this problem, we performed the non-parametric Friedman test [53] [54] and the post-hoc Nemenyi test [55] as suggested by Demšar [56]. The tests are based on ranks of the compared elements (algorithms) for each experiment's metric separately (where the best performance is ranked as 1) and searched for distances between the average ranks of the elements that are larger than a critical distance (CD) determined by the significance level alpha (set to 0.05), the number of datasets (N), and the number of compared algorithms (K). The original Friedman test compares a single performance metric of different algorithms over several datasets and attaches an overall rank to each algorithm. Having no single metric that is shared between the binary class and the multiclass prediction tasks and willing to compare the algorithms' performances in several aspects simultaneously i.e., false alarm rate as well as precision capabilities, we altered the Friedman test. Instead of using a single metric for each dataset, we used and ranked the algorithms with all measured metrics in every dataset. We excluded the AUC metric since it could not be calculated for several algorithms we tested.

In cases where the Friedman tests were significant, we continued with Nemenyi post-hoc tests comparing all algorithms to each other, allowing us to reveal their hierarchy. As suggested by Demšar [56], the results of the tests are presented visually in Figure 24. The diagram presents the average rank obtained from the Friedman test for all 14 algorithms. A horizontal line that connects algorithms indicates that they are statistically equivalent. Note that for $\alpha = 0.05, N = 44, K = 14$, the CD is 2.991—that is, a gap of more than 2.991 in the average ranks is statistically significant.

Reviewing the diagram, we can observe that the top three ranked algorithms are the three versions of the CDDRL. The ensemble BNs CDDRL version is the highest ranked algorithm and is significantly superior over the NN-UN, DWM, OBC, VFDR, SAM-KNN and RSLVQ algorithms. The ensemble methods (ranked second overall) and the original (ranked third overall) CDDRL versions show significant superiority over the same group of algorithms except for the NN-UN algorithm. The SRP algorithm (highest ranked algorithm after the CDDRL versions), is significantly better than the VFDR, SAM-KNN and RSLVQ algorithms. The middle ranked algorithms i.e., ARF, LNSE, KNN-ADWIN, AWE and NN-UN algorithms performed significantly better only compared to the RSLVQ algorithm. Finally, the bottom ranked algorithms i.e., DWM, OBC, VFDR, SAM-KNN and RSLVQ are not significantly superior to any other algorithm.

Knowledge Representation: We mentioned earlier that the CDDRL possess an ability no other algorithm discussed in this work does. That ability is to graphically visualize a process allowing us to better understand and explain the development of processes and drifts through time. This is done by exploring the BNs that the CDDRL has learned at different time steps during the data stream - making the CDDRL an adequate knowledge representation algorithm as well. We show an example of this ability in Figure 25. The figure shows four learnt BNs by the CDDRL during the Water Stress experiment. The blue 'Treatment' node is our target variable, and the green colored nodes are those that are included in the target variable's Markov blanket (MB) - having an effect on inference. Looking at these BNs, we can explain and perhaps provide better treatment for a plant undergoing water stress. BN_1 shows that the treatment the plants received doesn't affect any other plants' characteristic (no green nodes), this is logical since BN_1 was learned at the first time step - during the stable period - when all plants received the same treatment. However, BN_{15} shows that the treatment (water stress) affects the average third bottom temperature of the plants' leaves - again, logical, since water stressed plants close their stomata causing them to warm up. Looking at BN_{26} we can observe that the temperature of the plants is no longer affected (the stressed plants partially adapt to the stress) and now the average plants' perimeter as well as the average Hue (color measure) and the number of leaves it has are affected - the leaves of stressed plants curl, shrink and change their color and in general, the stressed plant has a difficult time growing new leaves. Finally, BN_{38} shows that at the end of the process, the same categories of plants' characteristics are affected (size, color and number of leaves), but some of them are shown via different measures i.e., maximum parameter and area of the plants' leaves (different size measures) and the average VARI which is a different color measure. A summary of plants biology and response to water stress can be found in [57].

The same thought process can be applied to different processes in different domains. For example, in a medical setting, observing BNs learned over time by the CDDRL, we can better understand and monitor changes in a patient's physical condition when transforming from a healthy state to illness.

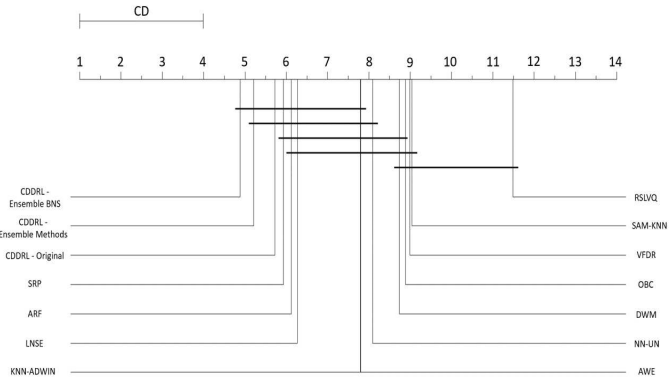


Fig. 24. Friedman-Nemenyi test with significance level of 0.05. The critical distance (CD) in this case is 2.991. The bold horizontal lines indicate statistical equivalence between the algorithms covered by it.

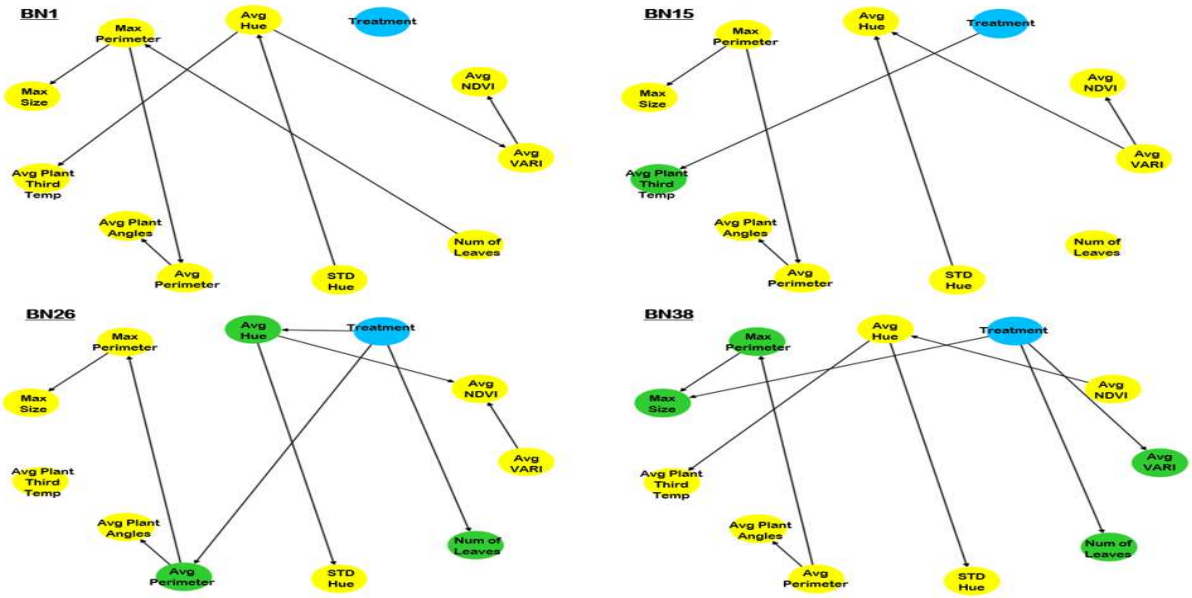


Fig. 25. BNs learned by the CDDRL's ensemble methods version during the water stress experiment. The blue 'Treatment' node is the target variable, colored in green are nodes included in the target variable's markov blanket.

Algorithm/Experiment	Sin	Stagger 500	Stagger 5000	Electricity	AWS Price	Usenet	NYC Subway	Fertilizer	Fertilizer + Water	Water
CDDRL-Original	6.3	7.6	8.3	18.0	583.2	1399.5	256.2	83.7	6.4	8.2
CDDRL-Ensemble-Methods	21.3	78.2	87.4	327.9	6353.8	*7958.6*	713.6	432.0	38.4	53.6
CDDRL-Ensemble-BNs	6.5	8.8	9.1	19.8	5549.5	5308.1	594.5	285.4	35.2	8.3
KNN-ADWIN	30.8	14.3	14.1	73.5	1727.7	36.9	1410.3	53.1	4.4	12.4
AWE	20.8	9.6	8.9	66.6	819.1	9.4	235.0	15.7	1.7	8.6
DWM	10.4	2.0	3.1	23.4	330.2	3.3	115.7	**2.0**	0.8	2.2
LNSE	34.6	23.8	20.0	61.6	1446.1	3.7	738.6	25.0	8.5	6.8
SRP	27.9	13.9	13.4	196.0	2646.4	35.9	503.4	45.0	10.7	32.2
SAM-KNN	10.3	3.1	2.8	27.5	450.1	**1.0**	58.4	9.8	2.2	4.4
VFDR	2.8	**1.4**	**1.3**	**4.7**	**85.1**	3.7	57.0	2.5	**0.7**	**0.7**
RSLVQ	**1.1**	7.6	7.4	31.7	305.7	1.4	**3.6**	7.0	3.9	5.2
OBC	*96.7*	42.7	48.9	*761.7*	*10184.8*	307.1	*4794.8*	*691.9*	*61.6*	*430.6*
ARF	10.8	20.5	19.8	248.1	6561.2	12.0	137.1	46.6	15.9	32.2
NN-UN	22.6	*98.2*	*95.1*	10.4	2539.5	37.0	832.7	83.1	42.3	4.7

Fig. 26. Run time of all algorithms in all datasets tested. Cells are colored in a green-red color template - the darker the green/red the better/worse the algorithm. Best algorithm in each dataset is in bold and the worst is italic and underlined.

Experiment	Num of Nodes	Avg. Parents	Max Parents	Ratio Avg. Parents	Ratio Max Parents
Electricity	7	1.06	1.94	0.15	0.28
AWS Spot Price	6	0.77	2.00	0.13	0.33
Usenet	99	1.06	9.93	0.01	0.10
NYC Subway Traffic	17	2.35	7.68	0.14	0.45
Fertilizer Stress	10	0.96	1.89	0.10	0.19
Fertilizer + Water Stress	4	0.44	1.19	0.11	0.30
Water Stress	10	0.73	1.51	0.07	0.15
Avg. Experiments	**21.86**	**1.05**	**3.73**	**0.05**	**0.17**

Fig. 27. Aggregated number of parents in each experiment. The ratio of average number of nodes out of total number of nodes, as well as the ratio of maximum number of nodes out of the total number of nodes are given.

To sum up this section, we conducted 7 CD experiments using real world datasets and tested all three CDDRL versions against 11 competing CD algorithms, having several evaluation metrics for each dataset. We obtained an overall rank for each algorithm using the altered Friedman-Nemenyi test and found that the three CDDRL versions are ranked at the very top, significantly superior over around half of the competing algorithms. In addition, we showed a unique capability of the CDDRL - a graphic visualization (BN) of developing relationships between variables in the domain, explaining in our case, the process of a plant going into water stress.

VIII. Complexity Analysis

For the computational complexity analysis we will denote the following: N - is the number of nodes. NS_{max} - is the maximum number of categories any variable has. C_{nodes} - are the nodes that were detected as drifted by the CDDRL in the current time step. P_{node_i} - is the number of parents node i has. P_{max} - is the maximum number of parents any node has. $Iter$ - is the number of iterations done by the simulated annealing during the search process.

The first stage of the CDDRL is to compute the distance of each node's current conditional probabilities from the corresponding ones in the previous time step. To do that, we need to compute the distance of each row in each node's CPT. The number of such distance computations per node is equal to the number of the node's parents' configurations. In the worst case, all parents of a node have NS_{max} categories - giving us a computational complexity of $O((NS_{max})^{P_{node_i}})$ for a single node and $O(N * (NS_{max})^{P_{max}})$ for all nodes. We note that at the worst case, a single node may have all other nodes as its parents, however we cannot have another node such as this because then cycles will be present and we therefore keep P_{max} and not $N-1$. Aggregating each node's CPD distances to one distance is negligible compared to the distance computations themselves.

At the second part of the CDDRL, we build and score possible graphs around the previous graph using C_{nodes}, we do this $Iter$ times. Each C_{node} can have at maximum two graphs generated by its interaction with each of the other

nodes - if a node was not a parent of the specific C_{node}, then it can be added as its parent and if that node was previously a parent, it can once be removed as a parent and second the edge can be reversed in case that node is also in C_{nodes}. Therefore the computational complexity of this part is $O(Iter * |C_{nodes}| * 2 * (N-1))$. In the worst case - where all nodes are in C_{nodes} - we get $O(Iter * N * 2 * (N-1))$ which is in fact $O(Iter * N^2)$. To the best of our knowledge and as stated in the original simulated annealing paper [58], there is no accurate complexity analysis of the simulated annealing process, but rather most related works concentrate on finding an optimal cooling schedule [59] [60]. Therefore, we do not have an upper bound on $Iter$ and since it can be manually limited (though not yet implemented in the current CDDRL versions), we disregard the complexity of $Iter$ and state that the complexity of the first CDDRL stage overshadows the complexity of the second stage, giving us a final CDDRL complexity of $O(N*(NS_{max})^{P_{max}})$. In case some nodes have many parents, then the CDDRL might have a long run time (see CDDRL's run time in the Usenet experiment shown in Figure 26). However, we can drastically reduce complexity by limiting the maximum number of parents a node can get (as done in the very well known PC algorithm [42]), or by creating variables with a limited number of categories.

Figure 27 shows the number of nodes in each experiment as well as the average number of parents (extracted from the BNs the original CDDRL has learned), the average maximum number of parents and the ratios of the average and max number of parents out of the total number of nodes. It's clear to see that the relatively high number of maximum parents in the Usenet and NYC Subway Traffic experiments is the cause for the CDDRL's long run time. Looking at the averages of all experiments, we can observe that the average number of parents is slightly above 1 where the maximum number of parents (perhaps most important) is 3.73, in addition we can see that the average parents ratio is just 5% and the maximum ratio is 17%, meaning that empirically we can state that the complexity of the CDDRL is $O(N*(NS_{max})^{0.05N})$ and we can get a better understanding of how limiting the maximum number of parents a node has will affect the CDDRL's run time.

In general, it can be observed in Figure 26 that the original version of the CDDRL is not in the bottom half of the algorithms in terms of run time (no red cells), except for the Usenet experiment where we had nodes with many parents, causing the CDDRL to run for a long time. The ensemble methods as expected have more predictive power but they also come with a longer run time due to the fact that they deal with several methods / BNs at every time step.

IX. Conclusion

Stream learning is not a trivial task, especially in the presence of concept drift. CD can be either real or virtual, and both can appear in several scenarios (e.g., sudden and gradual). To address CD in all its forms, we present the CDDRL—a concept drift detection and re-learning algorithm—that, by continually testing each node's CPT by an SPC test, detects which nodes have experienced a statistical change, and by using a local structure learning updating algorithm adjusts the BN for these changes in an efficient manner. We proved the correctness of the CDDRL mathematically and demonstrated its process empirically. We found the CDDRL's detection performance robust for all CD types and for different data characteristics, e.g., dimensionality and skewness.

Additionally, we introduced two ensemble versions of the CDDRL in an attempt to improve the CDDRL's predictive capabilities. Both ensemble versions were evaluated along side the original version and 11 other CD competing algorithms in both supervised and unsupervised settings.

In the supervised settings created through Stagger and Sin, all three CDDRL versions - and in particular the original and the ensemble methods versions - were not inferior, and even slightly (though not significantly) better for both sudden and gradual CD structures in both real and virtual CD, even though the CDDRL was not developed for classification as were the other competitors. In the seven real world datasets we tested, the CDDRL's versions were ranked at the very top, significantly better than half of the competitors (this time the ensemble BNs version showed the best results). The ensemble versions proved efficient in improving the CDDRL's predictive power.

In unsupervised settings, which the forementioned competitors cannot handle, the CDDRL was significantly superior compared to the other static approaches in three cases of adding/removing/reversing an edge of the Asia BN. It was also better for the Alarm BN in a scenario of simultaneous change that included all three structural changes at once.

Finally, besides the shown ability of the CDDRL regarding prediction, we showed - using the water stress experiment - that the CDDRL has the ability to explain a process, manifesting, naturally, explainable AI.

References

[1] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM computing surveys (CSUR)*, vol. 46, no. 4, pp. 1–37, 2014.

[2] I. Khamassi, M. Sayed-Mouchaweh, M. Hammami, and K. Ghédira, "Discussion and review on evolving data streams and concept drift adapting," *Evolving Systems*, vol. 9, no. 1, pp. 1–23, 2018.

[3] H. M. Gomes, J. Read, and A. Bifet, "Streaming random patches for evolving data stream classification," in *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2019, pp. 240–249.

[4] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Transactions on Neural Networks*, vol. 22, no. 10, pp. 1517–1531, 2011.

[5] D. Kifer, S. Ben-David, and J. Gehrke, "Detecting change in data streams," in *VLDB*, vol. 4. Toronto, Canada, 2004, pp. 180–191.

[6] I. Frías-Blanco, J. del Campo-Ávila, G. Ramos-Jimenez, R. Morales-Bueno, A. Ortiz-Díaz, and Y. Caballero-Mota, "Online and non-parametric drift detection methods based on Hoeffding's bounds," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 810–823, 2014.

[7] C. Raab, M. Heusinger, and F.-M. Schleif, "Reactive soft prototype computing for concept drift streams," *Neurocomputing*, vol. 416, pp. 340–351, 2020.

[8] L. G. Neuberg, "Causality: models, reasoning, and inference, by Judea Pearl, Cambridge University press, 2000," *Econometric Theory*, vol. 19, no. 4, pp. 675–685, 2003.

[9] D. Heckerman, D. Geiger, and D. M. Chickering, "Learning Bayesian networks: The combination of knowledge and statistical data," *Machine Learning*, vol. 20, no. 3, pp. 197–243, 1995.

[10] G. F. Cooper and E. Herskovits, "A Bayesian method for the induction of probabilistic networks from data," *Machine Learning*, vol. 9, no. 4, pp. 309–347, 1992.

[11] D. M. Chickering, "Learning equivalence classes of Bayesian-network structures," *Journal of Machine Learning Research*, vol. 2, no. March, p. 445–498, 2002.

[12] V. Losing, B. Hammer, and H. Wersing, "Knn classifier with self adjusting memory for heterogeneous concept drift," in *2016 IEEE 16th international conference on data mining (ICDM)*. IEEE, 2016, pp. 291–300.

[13] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Brazilian Symposium on Artificial Intelligence*. Springer, 2004, pp. 286–295.

[14] M. Baena-Garcıa, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-Bueno, "Early drift detection method," in *Fourth International Workshop on Knowledge Discovery from Data Streams*, vol. 6, 2006, pp. 77–86.

[15] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *Proceedings of the 2007 SIAM international conference on data mining*. SIAM, 2007, pp. 443–448.

[16] P. Kosina and J. Gama, "Very fast decision rules for classification in data streams," *Data Mining and Knowledge Discovery*, vol. 29, no. 1, pp. 168–202, 2015.

[17] B. Wang and J. Pineau, "Online bagging and boosting for imbalanced data streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 12, pp. 3353–3366, 2016.

[18] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem, "Adaptive random forests for evolving data stream classification," *Machine Learning*, vol. 106, no. 9, pp. 1469–1495, 2017.

[19] L. Baier, T. Schlör, J. Schöffer, and N. Kühl, "Detecting concept drift with neural network model uncertainty," *arXiv preprint arXiv:2107.01873*, 2021.

[20] M. Heusinger, C. Raab, and F.-M. Schleif, "Passive concept drift handling via momentum based robust soft learning vector quantization," in *International Workshop on Self-Organizing Maps*. Springer, 2019, pp. 200–209.

[21] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003, pp. 226–235.

[22] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts," *The Journal of Machine Learning Research*, vol. 8, pp. 2755–2790, 2007.

[23] H. Borchani, P. Larranaga, J. Gama, and C. Bielza, "Mining multi-dimensional concept-drifting data streams using Bayesian network classifiers," *Intelligent Data Analysis*, vol. 20, no. 2, pp. 257–280, 2016.

[24] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, no. 1/2, pp. 100–115, 1954.

[25] D. Heckerman, "A tutorial on learning with Bayesian networks," in *Innovations in Bayesian Networks*. Springer, 2008, pp. 33–82.

[26] A. Papoulis and S. U. Pillai, *Probability, random variables, and stochastic processes*. Tata McGraw-Hill Education, 2002.

[27] D. A. Levin and Y. Peres, *Markov chains and mixing times*. American Mathematical Soc., 2017, vol. 107.

[28] W. G. Cochran, "The χ^2 test of goodness of fit," *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 315 – 345, 1952. [Online]. Available: https://doi.org/10.1214/aoms/1177729380

[29] S. Kullback and R. A. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[30] A. Bhattacharyya, "On a measure of divergence between two statistical populations defined by their probability distributions," *Bull. Calcutta Math. Soc.*, vol. 35, pp. 99–109, 1943.

[31] D. C. Montgomery, *Introduction to statistical quality control*. John Wiley & Sons, 2007.

[32] S. Roberts, "Control chart tests based on geometric moving averages," *Technometrics*, vol. 42, no. 1, pp. 97–101, 2000.

[33] S. Bersimis, S. Psarakis, and J. Panaretos, "Multivariate statistical process control charts: an overview," *Quality and Reliability Engineering International*, vol. 23, no. 5, pp. 517–543, 2007.

[34] C. A. Lowry, W. H. Woodall, C. W. Champ, and S. E. Rigdon, "A multivariate exponentially weighted moving average control chart," *Technometrics*, vol. 34, no. 1, pp. 46–53, 1992.

[35] W. Buntine, "Theory refinement on Bayesian networks," in *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 1991, pp. 52–60.

[36] S. A. Teukolsky, B. P. Flannery, W. Press, and W. Vetterling, "Numerical recipes in C," *SMR*, vol. 693, no. 1, pp. 59–70, 1992.

[37] F. Glover and M. Laguna, "Tabu search," in *Handbook of combinatorial optimization*. Springer, 1998, pp. 2093–2229.

[38] J. Guiver and E. Snelson, "Bayesian inference for plackett-luce ranking models," in *proceedings of the 26th annual international conference on machine learning*, 2009, pp. 377–384.

[39] J. Montiel, J. Read, A. Bifet, and T. Abdessalem, "Scikit-multiflow: A multi-output streaming framework," *The Journal of Machine Learning Research*, vol. 19, no. 1, pp. 2915–2914, 2018.

[40] I. Tsamardinos, L. E. Brown, and C. F. Aliferis, "The max-min hill-climbing Bayesian network structure learning algorithm," *Machine Learning*, vol. 65, no. 1, pp. 31–78, 2006.

[41] J. C. Schlimmer and R. H. Granger, "Incremental learning from noisy data," *Machine Learning*, vol. 1, no. 3, pp. 317–354, 1986.

[42] P. Spirtes, C. N. Glymour, and R. Scheines, *Causation, prediction, and search*. MIT press, 2000.

[43] C. F. Aliferis, I. Tsamardinos, A. R. Statnikov, and L. E. Brown, "Causal explorer: A causal probabilistic network learning toolkit for biomedical discovery." in *Proceedings of Mathematics and Engineering Techniques in Medicine and Biological Scienes (METMBS)*, vol. 3, 2003, pp. 371–376.

[44] S. L. Lauritzen and D. J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 50, no. 2, pp. 157–194, 1988.

[45] 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*. Springer, 1989, pp. 247–256.

[46] M. Harries and N. S. Wales, "Splice-2 comparative evaluation: Electricity pricing," 1999.

[47] Amazon web services. [Online]. Available: https://aws.amazon.com/

[48] Aws spot pricing. [Online]. Available: https://www.kaggle.com/datasets/noqcks/aws-spot-pricing-market

[49] I. Katakis, G. Tsoumakas, and I. Vlahavas, "An ensemble of classifiers for coping with recurring contexts in data streams," in *ECAI 2008*. IOS Press, 2008, pp. 763–764.

[50] E. Gerber. Nyc subway traffic dataset. [Online]. Available: https://www.kaggle.com/datasets/eddeng/nyc-subway-traffic-data-20172021

[51] M. Grandini, E. Bagli, and G. Visani, "Metrics for multi-class classification: an overview," *arXiv preprint arXiv:2008.05756*, 2020.

[52] R. P. Espíndola and N. F. Ebecken, "On extending f-measure and g-mean metrics to multi-class problems," *WIT Transactions on Information and Communication Technologies*, vol. 35, 2005.

[53] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the american statistical association*, vol. 32, no. 200, pp. 675–701, 1937.

[54] ——, "A comparison of alternative tests of significance for the problem of m rankings," *The Annals of Mathematical Statistics*, vol. 11, no. 1, pp. 86–92, 1940.

[55] P. B. Nemenyi, *Distribution-free multiple comparisons*. Princeton University, 1963.

[56] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine learning research*, vol. 7, pp. 1–30, 2006.

[57] Y. Osakabe, K. Osakabe, K. Shinozaki, and L.-S. Tran, "Response of plants to water stress," *Frontiers in Plant Science*, vol. 5, 2014. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fpls.2014.00086

[58] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[59] D. Bertsimas and J. Tsitsiklis, "Simulated annealing," *Statistical science*, vol. 8, no. 1, pp. 10–15, 1993.

[60] Y. Nourani and B. Andresen, "A comparison of simulated annealing cooling strategies," *Journal of Physics A: Mathematical and General*, vol. 31, no. 41, p. 8373, 1998.

Yotam Baron received his B.Sc. degree in IEM from BGU. Currently, he is pursuing an M.Sc. degree in IEM at the same university, specializing in data science and machine learning. His research focuses on algorithms of concept drift detection to agriculture applications.

Shon Mendelson received a B.A in Statistics & Economics in 2018 and an M.Sc in Industrial Engineering and Management (IEM) in 2020, both degrees are from Ben-Gurion University (BGU), Israel. For the last year, he has been working as a data scientist in JFrog.

Liran Nahum received the B.Sc. and M.Sc. degrees in IEM from BGU in 2015 and 2016, respectively. For the last two years, he has been a data scientist in Superwise.

Moriya Cohen received the B.Sc. and M.Sc. degrees in IEM from BGU in 2018 and 2019, respectively. For the last year, she has been a freelancer in deep learning.

Yoav Reisner received his B.Sc. in Management & Sociology from BGU in 2019. Currently, he is pursuing an M.Sc. degree in IEM at BGU, specializing in data science and machine learning, while working as a data scientist in a high-tech startup. His research focuses on algorithms of concept drift detection to digital health.

Boaz Lerner received a B.A. degree in Physics & Mathematics from the Hebrew University in 1982 and a Ph.D. degree in Computer Engineering from BGU in 1996. He was a researcher at the Neural Computing Research Group at Aston University and the Computer Laboratory of Cambridge University, UK, and now is an Associate Professor at BGU. His current interests include structure learning of Bayesian networks and latent variable models and machine learning applications in digital health and precision agriculture.

תקציר

העולם מייצר כמויות אדירות של נתונים בכל דקה בכל יום, והמידע ממשיך לגדול בקצב מסחרר. חברות בכל תעשייה עוברות במהירות מעיבוד אצוות לעיבוד זרמי נתונים בזמן אמת כדי לעמוד בקצב הדרישות העסקיות המודרניות. זרמי נתונים נוטים להכיל בתוכם תופעה הנקראת סחיפה, שבה מאפיינים של הנתונים המגיעים משתנים בצורה כלשהי לאורך הזמן.

באופן כללי, שינוי יכול להיווצר מסיבות רבות, למשל, עליית מחירים עקב אינפלציה, צרכי אוכלוסייה מזדקנת, השפעת ההתחממות הגלובלית או בלאי של ציוד. התמודדות עם סחיפה ושמירה על מודל יעיל, מדויק ועדכני היא משימה מכרעת בעולמנו הדינמי.

בהתמודדות עם משימה זו, הוצג אלגוריתם המבוסס על רשתות בייסיאניות שמתמודד ביעילות עם סחיפה. האלגוריתם שלנו – המכונה זיהוי סחיפה ולמידה מחדש – יכול לשמש גם כאלגוריתם לייצוג ידע באמצעות ניתוח של רשתות בייסיאניות אשר נלמדו על ידו לאורך הזמן. האלגוריתם הינו אלגוריתם דו-שלבי. בשלב הראשון, הוא עוקב אחר מרחקים סטטיסטיים בין טבלאות ההסתברויות המותנות של הרשת באמצעות תרשימי בקרת תהליכים סטטיסטיים, על מנת לזהות שינויים פרמטריים ומבניים הדרושים ברשת כדי לעקוב אחר התפלגות הנתונים שעברה סחיפה. בשלב השני, האלגוריתם לומד מחדש את הרשת באופן מקומי דרך תהליך של חיפוש וניקוד תוך התבוננות בשכונה של גרפים אשר נוצרו מהגרף שהיה בשימוש לפני הסחיפה עם שינויים מבניים רק בקודקודים אשר זוהו כקודקודים שעברו סחיפה על ידי השלב הקודם של האלגוריתם.

בעבודה זו, אנו מציגים מספר פיתוחים לאלגוריתם זיהוי הסחיפה ולמידה מחדש – שיטה חדשה המבוססת על מבחני אי-תלות מותנים מוצעת כתחליף לחלק השני של האלגוריתם, כלומר, תחליף לתהליך החיפוש והניקוד. בנוסף, בניסיון להגדיל את יכולת הניבוי של האלגוריתם, הוצעו שתי גרסאות מבוססות אינסמבל. הגרסה הראשונה מבצעת ניבויים בכל נקודת זמן בעזרת רשתות אשר נלמדו בעבר, והגרסה השניה מבצעת את ניבוייה על ידי איסוף ניבויים ממספר שיפורים של האלגוריתם, שכל אחד מהם מכיל שינוי בודד לתהליך ספציפי באלגוריתם המקורי – שינוי בדרך בה נלמדות טבלאות ההסתברויות המותנות של הרשת, שינוי באופן בו משיגים ספי סחיפה, ותוספת של רשימת טאבו לתהליך החיפוש והניקוד של האלגוריתם.

השיטות החדשות יושמו ונבדקו יחד עם הגרסה המקורית של האלגוריתם. שלושה בסיסי נתונים סינתטיים אשר מכילים סוגים ורמות שונות של סחיפה שימשו להערכה, ביחד עם שבעה בסיסי נתונים מהעולם האמיתי, כאשר שלושה מאלו היו ניסויים של זיהוי עקת דשן ומים בצמחי בננה אשר נערכו איתנו. תוצאות הניסויים מעודדות מאוד, כאשר שתי שיטות האינסמבל כמו גם השיטה המקורית דורגו בראש רשימת הביצועים הכוללים על כל בסיסי הנתונים כפי שנמדד על ידי מבחן פרידמן-נמאני. ביצועי השיטות היו טובים בצורה מובהקת מלפחות חמישה מתוך אחד עשר האלגוריתמים המתחרים שנבחנו. שיטות האינסמבל שיפרו את יכולת החיזוי של האלגוריתם אבל הראו זמני ריצה ארוכים יותר, בעוד שהשיטה מבוססות מבחני האי תלות המותנים – שנבחנה בניסויים אחרים – הייתה נחותה יחסית לגרסה המקורית של האלגוריתם אבל הציגה זמני ריצה קצרים יותר.

מילות מפתח: רשתות בייסיאניות, סחיפה, חיפוש וניקוד, מבחני אי-תלות מותנים, שיטות אינסמבל, עקות בצמחים, מבחן פרידמן-נמאני

אוניברסיטת בן-גוריון בנגב

הפקולטה למדעי ההנדסה

המחלקה להנדסת תעשייה וניהול

זיהוי סחיפה מבוסס אינסמבל ורשתות בייסיאניות

חיבור זה מהווה חלק מהדרישות לקבלת תואר מגיסטר בהנדסה

מאת

יותם ברון

בהנחיית: פרופ' בעז לרנר

חתימת המחבר: ..... yotam ..... תאריך: 26.12.22

אישור המנחה: ..... תאריך: 26.12.22

אישור יו"ר ועדת תואר שני מחלקתית: ..... תאריך: 26.12.22

אלול תשפ"ב ספטמבר 2022

אוניברסיטת בן-גוריון בנגב

הפקולטה למדעי ההנדסה

המחלקה להנדסת תעשייה וניהול

זיהוי סחיפה מבוסס אינסמבל ורשתות בייסיאניות

חיבור זה מהווה חלק מהדרישות לקבלת תואר מגיסטר בהנדסה

מאת

יותם ברון

בהנחיית: פרופ' בעז לרנר

אלול תשפ"ב ספטמבר 2022