École Centrale de Nantes
École Polytechnique de l'Université de Nantes
Département Informatique - LS2N : Laboratoire des Sciences du Numérique de Nantes

PILGRIM: Multi-task learning for Bayesian Networks.

Abdellah LAASSAIRI & Zihao GUO
Supervised by : Phillippe LERAY

Submitted in part fulfilment of the requirements for the R&D course
at École Centrale de Nantes, May 25, 2023

Abstract

Bayesian networks are probabilistic graphical models that rely on the properties of a set of random variables and their conditional probability distributions. In our current work, we use a c++ library dedicated to Bayesian networks proposed by the University of Nantes called PILGRIM in order to validate GS and MMHC algorithms in both single-task and multi-task learning contexts for Bayesian network structures. In addition, we will also validate the implementation of a caching system that is used to speed up the computation of those algorithms.

Keywords: Bayesian Networks · Structure learning · Multi-task learning · Transfer learning.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Pilgrim

PILGRIM [7] is a software developed by the DUKe team at LS2N dealing with bayesian network learning. There are three distinct versions of the library :

1. PILGRIM-General: Manipulation, evaluation, and learning of static or dynamic Bayesian networks by GreedySearch (GS) and Max-MinHillClimbing (MMHC)
2. PILGRIM-Relational: Definition of probabilistic relational models and their extensions and their transformation into a Bayesian network.
3. PILGRIM-Evential: Creation of graphical event models, learning of such models with event log files.

1.2 Problem Statement

This project objective is to consolidate the experimental study developed in the article [1] which compares two MT algorithms (Multi-Task) to two ST "Single Task" solutions on different benchmarks. This consolidation involves the step-by-step validation of the different implementations used, which are based on the PILGRIM-General library developed by the DUKe team.

1.3 The Proposed Methodology

To ensure an impartial validation of the previously mentioned results, we propose the following methodology that we'll be conducting for this study :

1. Setting up the work environment for reproducible results.
2. Verification of the results and behavior of algorithms introduced in [1] on the Alarm and Asia datasets.
3. Implementation of those results on additional datasets for further verification.
4. Study of optimization paths (such as caching) in order to accelerate the introduced algorithms.

Chapter 2

Background

In this section we'll be introducing the base concepts and theories required for this project.

2.1 Bayesian Network definition

A Bayesian Network or Belief Network (BN) [5] is a Probabilistic Graphical Model (PGM) that represents conditional dependencies between random variables through a Directed Acyclic Graph (DAG).

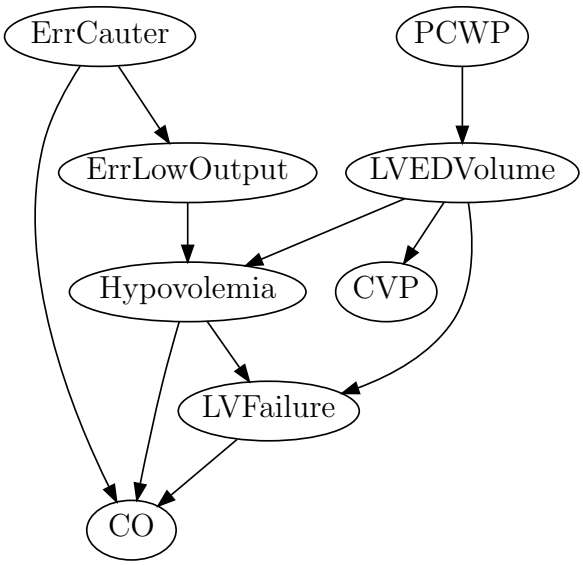


Figure 2.1: A Bayesian Network representation example

2.1.1 Principle Introduction

Definition Let $G = (I, E)$ denote a directed acyclic graph (DAG), where I represents the set of all nodes in the graph, and E represents the set of directed connected line segments, and let $X = (X_i)_{i \in I}$ be the random variable represented by a node i in its directed acyclic graph, if the joint probability of a node X can be expressed as

$$p(x) = \prod_{i \in I} p(x_i | x_{pa(i)}) \tag{2.1}$$

Then, X is said to be a Bayesian network relative to a directed acyclic graph G, where $pa(i)$ denotes the "cause" of node i, or pa(i) is the parents (parents) of i. Furthermore, for any random variable, the joint probabilities can be derived by multiplying the respective local conditional probability distributions.

$$p(x_1, ..., x_K) = p(x_K | x_1, ..., x_K) ... p(x_2 | x_1) \cdot p(x_1) \tag{2.2}$$

A simple Bayesian network is shown in the [Fig. 2.3(a)].

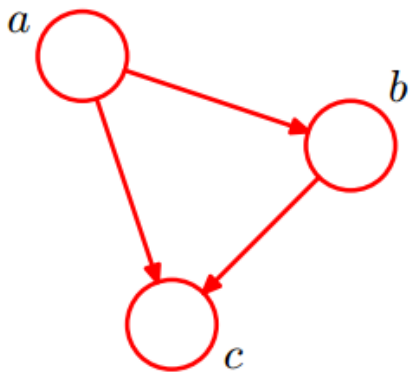
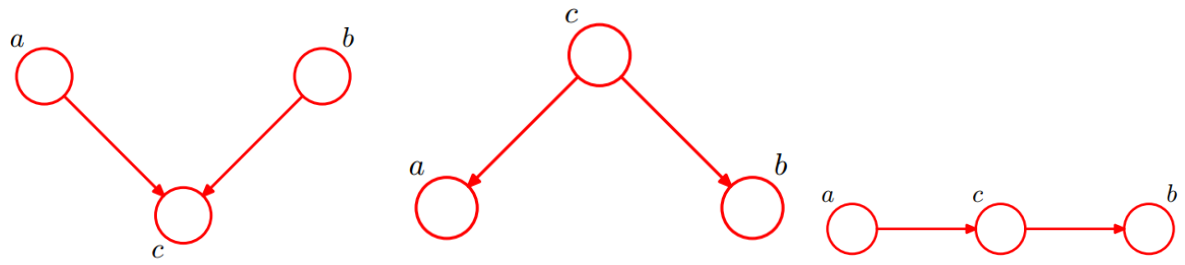


Figure 2.2: Simple bayesian network illustration

Since a leads to b and a and b lead to c, there is $p(a, b, c) = p(c|a, b) \cdot p(b|a) \cdot p(a)$

2.1.2 Structure form

Bayesian networks have three structural forms.



(a) head-to-head (b) tail-to-tail (c) head-to-tail

Figure 2.3: Bayesian network structure form

1. The first structure is shown in the [Fig. 2.3(a)], so we have: P(a,b,c) = P(a)*P(b)*P(c—a,b) holds and after simplification we get: $\underline{\sum_c P(a,b,c) = \sum_c P(a) \cdot P(b) \cdot P(c|a,b)} \Rightarrow P(a,b) = P(a) \cdot P(b)$

2. The second structure is shown in the [Fig. 2.3(b)], we have $P(a,b,c) = P(c) * P(a|c) * P(b|c)$, then: $P(a,b|c) = P(a,b,c)/P(c)$, then we bring $P(a,b,c) = P(c) * P(a|c) * P(b|c)$ into the above equation to get: $\underline{P(a,b|c) = P(a|c) * P(b|c)}$

3. The third structure is shown in the [Fig. 2.3(c)], we have: $P(a,b,c) = P(a) * P(c|a) * P(b|c)$, after simplification, we get: $\underline{P(a,b|c) = P(a|c) * P(b|c)}$

The main objective of the Bayesian networks is to model the posterior conditional probability distribution of outcome (often causal) variable(s) after observing new evidence. Bayesian networks may be constructed either manually with knowledge of the underlying domain, or automatically from a dataset by different approaches which we'll be discussing in the following sections.

2.2 Bayesian network structure learning

2.2.1 Introduction to BN structure learning methods

There are three methods for learning the structure of Bayesian networks from data.[12]
The first approach views learning as a problem of constraint satisfaction[3]. To obtain the optimal network structure, the dependencies between variables are typically analyzed using statistical or information theoretic methods. This is usually accomplished through statistical hypothesis testing, such as the t-test. Then we construct a network that demonstrates the observed dependencies and independence.

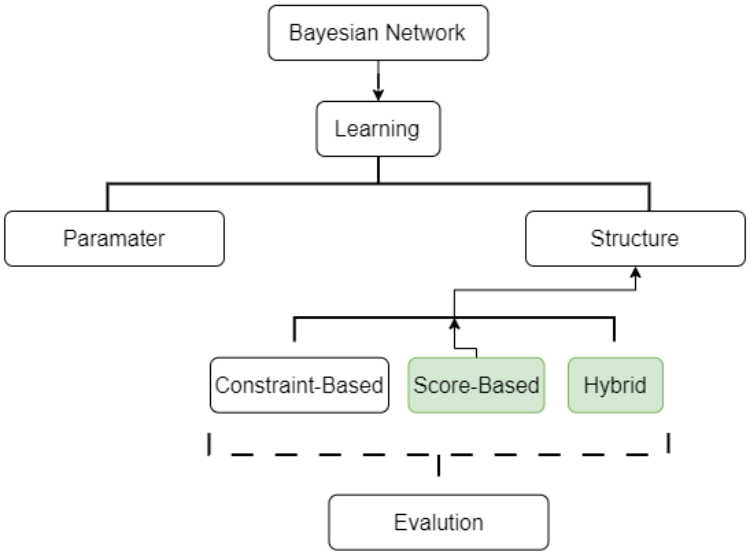


Figure 2.4: Different concepts of Bayesian network learning

The second method is to think of learning as an optimization problem. We begin by defining a statistically motivated score indicating how well the learned network matches the real network. The learner's task is then to find a structure that maximizes the score.

As a result, we frequently rely on heuristics, as demonstrated by Larranaga et al. via genetic algorithms[6], and while the constraint satisfaction approach is effective, it is sensitive to failure of independence tests. As a result, most people believe that optimization methods are a better tool for learning structure from small amounts of data.

By integrating the first two strategies, a third strategy known as hybrid methods can overcome this problem[8][10]. Local search methods use local information as constraints to identify local structures and optimize global models. These approaches are capable of dealing with distributions with thousands of variables.

In general, learning BNs from training data in order to apply BNs to real-world applications is a difficulty. There are two ways of learning BNs: learning BN parameters when the BN structure is known and learning both the BN structure and the parameters. We highlight that both complete and incomplete data can be used to learn BN parameters[4]. This is not as difficult to master as the BN structure. We will concentrate on structure learning with complete data in this task.As shown in [Fig. 2.4], we have tried the latter two methods.

2.2.2 Strategies for structural learning

1. Single Task Learning

Bayesian Networks Single task learning purpose is to determine the structure of the network G and estimating the parameters Θ of the model from one dataset D.

2. Multi-Task Learning

In Multi-Task learning [11], we consider k tasks corresponding to k datasets & graphs such as:

$$G = (G_1, G_2, \ldots, G_k) \qquad D = (D_1, D_2, \ldots, D_k)$$

The objective is to learn all the models simultaneously as a multi-task problem while leveraging information between the tasks.

2.2.3 Evaluation of Bayesian Network structure learning algorithms

1. Evaluation metrics

Confusion matrix is a table that summarizes the performance of a classification algorithm by comparing the predicted and actual class labels of a set of data. It is often used in machine learning and pattern recognition to evaluate the accuracy and quality of a model's predictions. The confusion matrix displays the number of true positives, false positives, true negatives, and false negatives, which can be used to calculate various performance metrics such as precision, recall, and F1 score. The concept of confusion matrix was introduced by L. A. Goodman in 1984 [2] and has since become a widely used tool for evaluating the performance of classification algorithms.

Sensitivity and specificity are statistical measures of the performance of a binary classification test, often known as the classification function in statistics.

The percentage of true positives that are accurately identified is referred to as sensitivity. Specificity is the percentage of accurately identified negatives.

The sensitivity-specificity-based technique begins by computing the following indices given the theoretical network graph G_0 = (V, E_0) and the learned network graph G = (V, E):

- TP (true positive) = number of edges present in both G_0 and G
- TN (true negative) = number of edges absent in both G_0 and G
- FP (false positive) = number of edges present in G, but not in G_0
- FN (false negative) = number of edges absent in G, but not in G_0

Then, the sensitivity and specificity can be calculated as follows:

$$Sensitivity = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{TN + FP}$$

These measures are easy to calculate. They are often used in the literature. However, the differences in orientation between the two graphs in the same equivalence class are counted as errors.

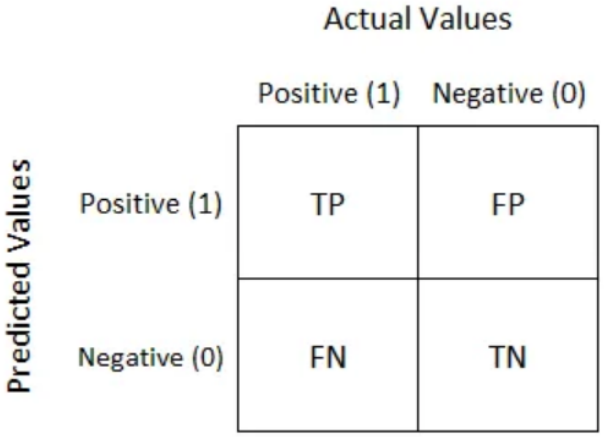


Figure 2.5: Confusion Matrix

2. Structural hamming distance

In the context of Bayesian network structure learning, SHD can be used to compare the learned structure with the true underlying structure. If the learned structure is close to the true structure, then the SHD between them will be small. On the other hand, if the learned structure is far from the true structure, then the SHD will be large. Therefore, SHD can be used as a measure of accuracy for Bayesian network structure learning algorithms.

In our experiments, We assess performance using run time and the structural Hamming distance (SHD) between the true structure (from which we generate sampled data) and the learned structure, or, more accurately, the distance between the underlying graphs. We show the mean and standard deviation of the SHD for 10 runs x 5 tasks, as well as the mean and standard deviation of the execution time for 10 runs, for each experiment.

Chapter 3

Literature Review

In this chapter, a variety of methods that deal with the Bayesian Networks learning.

3.1 Search and score approaches

Search and score techniques are widely used for learning Bayesian network structures. These techniques involve performing an exact or heuristic search within the network structure space and evaluating the best structure to fit the data based on a given scoring metric. To accomplish this, a search space of allowable states of the problem is required, along with a scoring function to evaluate a state's fit with the data and a mechanism for exploring the search space in an exact or heuristic way. For example, Greedy Search (GS)[13] is a popular ST structure learning algorithm that explores the DAG space by selecting the neighbor graph with the highest score at each iteration. A neighborhood of a structure is commonly defined as all the DAGs obtained by removing, reversing, or adding an edge in the structure.

To evaluate states, various scores have been used in the literature, such as AIC, BIC, or BDeu, and more recent ones such as qNML. Niculescu-Mizil and Caruana[9] proposed the MT-GS algorithm, which extends the score-based search to a MT context by considering a configuration of k structures and following a greedy search procedure to find the k graphs that fit the k respective datasets best. They define a neighborhood of a configuration as the set of all

configurations obtained by considering all possible subsets of the graphs and applying an add, remove, or reverse operation to the same edge in each graph of each subset.
The scoring function maximizes the posterior probability of a configuration given the data. Two different priors are proposed: an edit prior and a paired prior. The scoring function takes into account data from all tasks and leverages information between them. To optimize the computational complexity of this method, a Branch-and-Bound strategy is used to find the best configuration in the neighborhood of the current one. The exploration goes through a search tree of depth k, and each sub-tree rooted at a partial configuration whose score is lower than the current best score can be pruned.

Use of search and score approaches is one of the most well-known ways to learn Bayesian network architectures. For instance, the ST structure learning method Greedy Search (GS)[13] traverses the DAG space by picking the neighbor graph with the highest score after each iteration. The search may be repeated numerous times, starting from different initial states to avoid local maxima, and finishes when the current structure outperforms all of its neighbors in terms of score. In an MT setting, Niculescu-Mizil and Caruana[9] enhance this score-based search. This algorithm is called MT-GS, for short. They take into account a configuration of k structures $(G_1, G_2, ..., G_k)$ and then use a greedy search method to locate the k graphs that best match the k different datasets. For states evaluation, several scores have been settled in the literature such as AIC, BIC or more recent ones such as qNML.

3.2 Constraint-based approaches

In constraint-based approaches for learning Bayesian network structures in multi-task transfer learning settings, the algorithms test conditional independencies between variables to identify the graph that describes the discovered dependencies and independencies. The key difference in these approaches is the evaluation of independence tests, which involves a linear combination of independence measures from the target task and the closest auxiliary task. The combined similarity measure S_c is calculated using global similarity S_g and local similarity S_l as follows:

$$S_{gab} = \sum_{X<Y} 1(I_a(X,Y) - I_b(X,Y)) \tag{3.1}$$

$$S_{lab}(X,Y|S) = \begin{cases} 1, & \text{if } I_a(X,Y|S) = I_b(X,Y|S) \\ 0.5, & \text{otherwise} \end{cases} \tag{3.2}$$

$$S_{cab}(X,Y|S) = S_{gab} \times S_{lab}(X,Y|S) \tag{3.3}$$

The confidence measure α_a and the combined independence function I_{ca} are given by:

$$\alpha_a(X,Y|S) = 1 - \frac{\log N_a}{2N_a \times T} \tag{3.4}$$

$$I_{ca}(X,Y|S) = \alpha_a(X,Y|S) \times \text{sgn}(I_a(X,Y|S)) + \alpha_{b^*}(X,Y|S) \times S_{cab^*}(X,Y|S) \times \text{sgn}(I_{b^*}(X,Y|S)) \tag{3.5}$$

3.3 Hybrid approaches

Max-Min Hill-Climbing(MMHC)[14] is one of the more successful current hybrid Bayesian structure learning algorithms, whose main idea is to build a Bayesian network skeleton first, and then determine the final structure by the greedy algorithm.

For the established Bayesian network framework, the main process is done in two steps.

1. Forward, populate the CPC of the parent-child node table of the target variable T, using the maximum-minimum heuristic. Maximum-minimum means that the CPC table is built in such a way that the node to be selected maximizes the minimum connection with the target variable at a given CPC table, and so the preliminary CPC table is obtained.

2. Backward, in the CPC, given the variables S in it, if the target variable T is independent of the element X in the CPC table, X is removed (proving non-parent-child relationship).

After building the CPC table, then the algorithm is using a hill-climbing search algorithm to search in a narrowed search space. But the disadvantage is that there is still the possibility of falling into local optimum. For this, simulated annealing, random restart and forbidden search are used in the paper[14] for improvement, and the results are found to be improved.

3.4 Applied methods

3.4.1 Greedy search (GS)

Greedy search (GS)[13] algorithm, in [Alg. 1] employs heuristic problem solving in which the locally optimal choice is made at each stage in the goal of finding a global optimum. It is started by the network G0. It collects all simple graph operations (e.g., edge addition, removal, or reversal) that can be performed on the network without violating the restrictions (e.g., introducing a cycle), and we utilize the Generate neighborhood algorithm for [Alg. 2]. Then it chooses the operation that has the greatest impact on the network's score (this phase can be repeated if the network can still be enhanced or if the maximum number of interactions has not been reached).

Algorithm 1 G_0

Require: Initial graph (G_0)

Ensure: BN structure (DAG)

$G \leftarrow G_0$
$Test \leftarrow True$
$S \leftarrow Score(G, D)$
while $Test = True$ **do**
    $N \leftarrow Generate\_neighborhood(G, \phi)$
    $G_{max} = \arg\max_{F \subset N} Score(F, D)$
    if $Score(G_{max}, D) > S$ **then**
        $G \leftarrow G_{max}$
        $S \leftarrow Score(G_{max}, D)$
    else
        $Test \leftarrow False$
    end if
end whilereturn the DAG G found

Algorithm 2 Generate neighborhood(G, G_c)

Require: Current DAG (G); undirected graph of constraints (G_c)

Ensure: Set of neighborhood DAGs (N)

1: **for all** $e \in G$ **do**
2: $N \leftarrow \phi \cup (G \setminus \{e\})\%delete\_edge(e)$
3: **if** $acyclic(G \setminus \{e\} \cup invert(e))$ **then**
4: $N \leftarrow N \cup (G \setminus \{e\} \cup invert(e))\%invert\_edge(e)$
5: **end if**
6: **end for**
7: **for all** $e \in G_c$ and eG **do**
8: **if** $acyclic(G \cup \{e\})$ **then**
9: $N \leftarrow N \cup (G \cup \{e\})\%add\_edge(e)$
10: **end if**
11: **end for**

3.4.2 Max-Min Hill-Climbing (MMHC)

The Max-Min Hill-Climbing (MMHC) algorithm [Alg. 3] can be classified as a hybrid approach, using concepts and techniques from both methods. It is a general Bayesian network learning algorithmic template and can be adapted with different heuristics and search methods. MMHC first uses a local discovery algorithm called Max-Min Parents and Children (MMPC)to learn the skeleton (i.e., edges with no direction) of a Bayesian network, and then, it uses a hill-climbing search with greedy Bayesian scores to determine the direction of the skeleton.

The algorithm starts by identifying the parent and child sets for each variable and then uses a greedy hill-climbing search to explore the space of BNs, starting with an empty graph. During the search, edges are added, deleted, or reversed in direction, with the aim of maximizing the score, and only edges identified by the MMPC in the first phase are considered for addition. To avoid local maxima, a TABU list is used, and the algorithm terminates after 15 changes without an increase in the maximum score.

The best scoring structure is returned at the end. The MMHC algorithm improves efficiency by constraining the search and using a sound algorithm (MMPC) for identifying candidate parent sets. It initializes candidate parent sets to include parents and children and can also handle small sample sizes with pairwise tests of independence.

Algorithm 3 Max-Min Hill-Climbing (MMHC) Algorithm

Require: Dataset D, significance level α, maximum number of changes T_{max}

Ensure: The best scoring Bayesian network structure G

Initialize an empty graph G
Identify the parent and child sets for each variable in D
Initialize the candidate parent sets to include parents and children
Use MMPC algorithm to identify edges for initial graph G
$bestScore \leftarrow$ score(G, D)
$T \leftarrow 0$
while $T < T_{max}$ **do**
    $T \leftarrow T + 1$
    Generate a list of candidate edge changes
    Apply tabu search strategy to the candidate list
    Find the best scoring edge change
    if score($G + \Delta G$, D) $>$ $bestScore$ **then**
        Update G to $G + \Delta G$
        $bestScore \leftarrow$ score(G, D)
        $T \leftarrow 0$
    end if
end whilereturn G

3.4.3 MT-GS

We use Niculescu-Mizil and Caruana's [9] extended score-based search in the MT context in this task, we call it "MT-GS". They consider a configuration of k structures $G_1, G_2, ..., G_k$

consisting of configurations that then follow a greedy search procedure in the DAG space to find the k graphs that best fit the k respective datasets. The neighborhood of a configuration is defined as the set of all configurations that are obtained by considering all possible subsets of graphs and adding, deleting or reversing the same edges in each graph for each subset. The fraction to be maximized is the posterior probability of the configuration given the data defined in [Eq. 3.6].

$$P(G|D) = P(G_1, ..., G_k|D_1, ..., D_k) \propto P(G) \prod_{a=1}^{k} P(D_a|G_a) \tag{3.6}$$

The Multi-Task Greedy Search (MT-GS) algorithm is a search-and-score based approach for learning the structure of Bayesian Networks in a multi-task setting. It is designed to learn multiple Bayesian Network structures simultaneously, leveraging the similarities between tasks to improve the learning process. The primary idea is to share information between tasks so that the learning process for one task can benefit from the knowledge gained from other tasks.

1. Initialization: Create a set of Bayesian Network structures, one for each task. For each task, initialize an empty graph with nodes representing variables and no edges.

2. Iteration: Iterate through a series of edit operations on the edges of the graphs. Edit operations include adding an edge (X $\rightarrow$ Y), removing an edge (X $\rightarrow$ Y), reversing an edge (X $\rightarrow$ Y to Y $\rightarrow$ X), or leaving the edge unchanged.

3. Scoring: For each edit operation, calculate the score of the resulting graph configuration by applying a scoring function, such as the Bayesian Information Criterion (BIC) or the Akaike Information Criterion (AIC). The scoring function typically evaluates the goodness-of-fit of the graph to the data while penalizing model complexity.

4. Task Similarity: Compute the similarities between tasks using a similarity measure that reflects the degree of relatedness between the tasks. The similarity measure can be based on the data distributions, the graph structures, or other task-specific characteristics.

5. Sharing Information: For each task, share information from other tasks according to the task similarity measure. This information can be shared in the form of constraints on the

graph structure or by combining scores from multiple tasks. Sharing information allows the learning process for one task to benefit from the knowledge gained from other tasks.

6. Selecting Edit Operations: Select the edit operation that leads to the maximum score improvement, taking into account the shared information between tasks. Update the corresponding graph with the selected edit operation.

7. Termination: Continue iterating through edit operations until no further improvement can be made or a stopping criterion is met, such as reaching a maximum number of iterations or a threshold on the score improvement.

Algorithm 4 Multi-Task Greedy Search (MT-GS)

Input: Datasets $D_1, D_2, \ldots, D_k$ for k tasks
Output: Bayesian Network structures $G_1, G_2, \ldots, G_k$
Initialize $G_1, G_2, \ldots, G_k$ as empty graphs with nodes representing variables
while not converged **do**
    for each task a **do**
        for each pair of nodes (X, Y) **do**
            for each edit operation op in $\{add, remove, reverse, no\_change\}$ **do**
                Apply op to G_a to create G'_a
                Compute score s_{op} for G'_a using a scoring function
            end for
            Determine the best edit operation op^* with the maximum score improvement
            Update G_a with op^*
        end for
    end for
    Update task similarities and share information between tasks
end while
return $G_1, G_2, \ldots, G_k$

The MT-GS algorithm aims to find an optimal set of Bayesian Network structures for all tasks simultaneously while considering their similarities. By sharing information between tasks, it can leverage the knowledge from related tasks to improve the learning process, potentially leading to better graph structures and more accurate Bayesian Networks.

3.4.4 MT-MMHC

Benikhlef et al.[1] proposed MT-MMHC, a hybrid approach for multi-task problems that aims to learn k BN structures from k similar problems at the same time by combining the benefits of constraint-based algorithms, score and search-based algorithms, TL and MT learning techniques.

The main concept is to apply the MT-MMHC algorithm to the MT scenario, as illustrated in [Fig. 3.1]. The procedure, like ST's algorithm, begins with a constraint-based phase to determine the set of CPCs associated with each task. It uses a local search technique that is guaranteed by the MMPC algorithm (refer to the gray box in [Fig. 3.1]).

Algorithm 5 MT-MMHC Algorithm

Require: Datasets $D = \{D_1, D_2, \ldots, D_k\}$ with k tasks.
Ensure: Bayesian Network structures for each task.
1: **Phase 1**: Constraint-based phase with MMPC
2: **for** $i = 1$ to k **do**
3: Identify CPC set CP_i for task D_i using MMPC with combined association metric
4: **end for**
5: **Phase 2**: Greedy search for edge orientation
6: **for** $i = 1$ to k **do**
7: Initialize BN structure for task D_i with CPC set CP_i.
8: Apply MT greedy search algorithm with constraint to CP_i for edge orientation.
9: **end for**

The BN structure learning algorithms mainly perform transfer learning in single task(ST) scenario with constraint based approaches or multi-task consideration with search-and-score methods.

- **Phase 1** extends the MMHC algorithm to the MT scenario. It starts with a constraint-based phase to identify the CPC sets associated with each task using MMPC with the combined association metric.
- **Phase 2** applies the MT greedy search algorithm adapted to our context by constraining it to the discovered CPCs for edge orientation. The BN structures are initialized with the corresponding CPC sets and the search algorithm is applied for each task D_i.

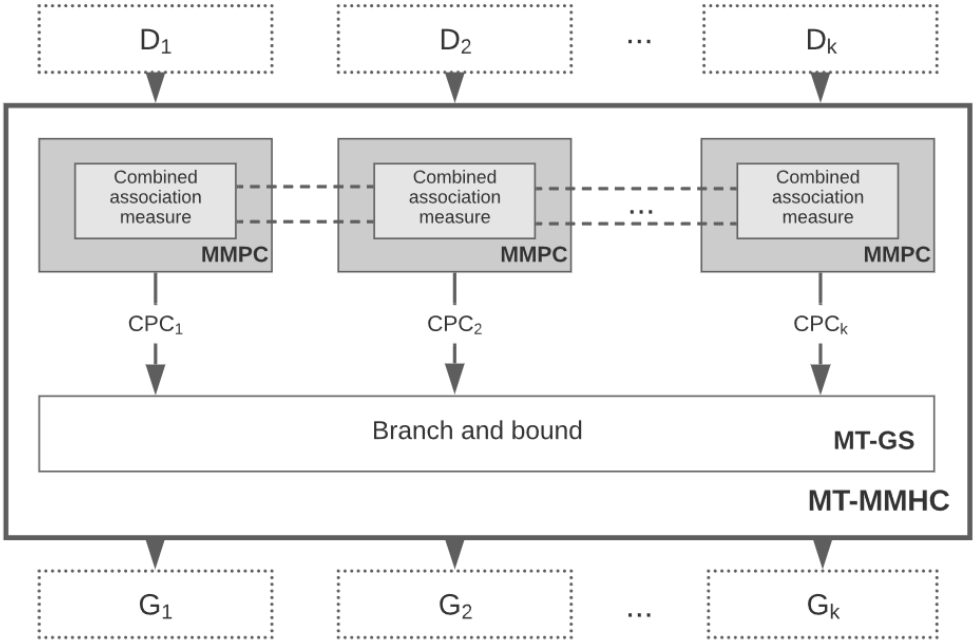


Figure 3.1: Overall process of MT-MMHC.[1]

Chapter 4

Experiments and Results

In this chapter, we will present the various experiments we've conducted as well as the results we've obtained and compare them to those obtained by [1] and draw our conclusions.

4.1 Bayesian network structures

ASIA and ALARM are two well-known Bayesian network structures used in research to evaluate and benchmark various algorithms.

1. **ASIA Bayesian Network:**

   ASIA (Asian Smokers and Airway Diseases) is a simple Bayesian network designed to model the relationship between smoking and lung diseases, particularly in the Asian population. It consists of 8 nodes and 8 directed edges. The nodes in the ASIA network represent the following variables:

   - A: Asia visit (whether a person has visited Asia recently)
   - T: Tuberculosis (whether the person has tuberculosis)
   - S: Smoking (whether the person is a smoker)
   - L: Lung cancer (whether the person has lung cancer)

- B: Bronchitis (whether the person has bronchitis)
- E: Either tuberculosis or lung cancer (whether the person has either tuberculosis or lung cancer)
- X: X-ray result (the result of a chest X-ray)
- D: Dyspnea (whether the person experiences shortness of breath)

The ASIA network models the dependencies between these variables, considering factors like visiting Asia, smoking, and the presence of various respiratory diseases. Due to its simplicity, the ASIA network is widely used in research for testing and comparing Bayesian network learning algorithms.

2. **ALARM Bayesian Network:**

ALARM (A Logical Alarm Reduction Mechanism) is a more complex Bayesian network used in the context of patient monitoring in intensive care units (ICUs). It was designed to model the relationships between various medical measurements and the patient's health status. The ALARM network consists of 37 nodes and 46 directed edges, representing a wide range of variables related to patient monitoring, such as medical tests, physiological variables, and device settings.

Some examples of nodes in the ALARM network include:

- CVP: Central venous pressure
- PCWP: Pulmonary capillary wedge pressure
- CO: Cardiac output
- HR: Heart rate
- BP: Blood pressure
- HREKG: Heart rate from EKG

The ALARM network is used to evaluate the performance of more advanced Bayesian network learning algorithms, as it represents a more realistic and challenging problem compared to simpler networks like ASIA.

Bronchitis	Dyspnea	LungCancer	Smoking	TbOrCa	Tuberculosis	VisitToAsia	XRay
Absent	Present	Present	Smoker	CancerORTuberculosis	Absent	NoVisit	Abnormal
Absent	Absent	Present	Smoker	CancerORTuberculosis	Absent	NoVisit	Abnormal
Present	Present	Absent	NonSmoker	Nothing	Absent	NoVisit	Normal
Present	Absent	Absent	Smoker	Nothing	Absent	NoVisit	Normal
Absent	Absent	Absent	NonSmoker	Nothing	Absent	NoVisit	Abnormal
Present	Present	Absent	NonSmoker	Nothing	Absent	NoVisit	Normal
Absent	Absent	Absent	Smoker	Nothing	Absent	NoVisit	Normal
Present	Present	Absent	Smoker	Nothing	Absent	NoVisit	Normal
Present	Present	Absent	Smoker	Nothing	Absent	NoVisit	Normal
Present	Present	Absent	Smoker	Nothing	Absent	NoVisit	Normal

Table 4.1: Asia Bayesian Network

Anaphylaxis	ArtCO2	BP	CO	CVP	Catechol	...	ErrCauter	ErrLowOutput	ExpCO2
False	Normal	Normal	Normal	Normal	Normal	...	False	False	Normal
False	Normal	Low	Low	Low	High	...	False	False	Normal
False	Normal	High	High	Normal	High	...	False	False	Normal
False	Normal	Normal	Normal	Normal	Normal	...	False	False	Normal
False	Normal	Normal	Low	Normal	Normal	...	True	False	Normal
False	Normal	Normal	Low	Low	Normal	...	False	False	Normal
False	Normal	Low	High	Normal	High	...	False	False	Normal
False	Normal	High	High	Normal	High	...	False	False	Normal
False	Normal	Normal	Low	Normal	High	...	True	False	Zero
False	High	High	High	High	High	...	False	False	High

Table 4.2: Alarm Bayesian Network

Benchmarking ST algorithms is straightforward, however as it has already been specified in the [1] paper, there are no benchmarks for evaluating MT algorithms. We will follow the same protocol proposed by the authors of the paper.

4.2 Implementation Tools

For the implementation of the various algorithms, experiments, and benchmark procedures as well as data generation protocols we're going to be relying on the Pilgrim-general library which was introduced in the first chapter and developed by the LS2N DUKe team.

4.3 Experiments

4.3.1 Single Task (ST)

We'll start by benchmarking different ST algorithms with different strategies and parameters, in order to validate this step before moving on to $k-ST$ paradigm. As specified previously we're going to rely on both 'Alarm' and 'Asia' for our benchmarks. The protocol for the experiments is as follows :

1. Specify the parameters of the experiment :
   (a) 'n_runs': number of iterations to execute
   (b) 'algorithm': the algorithm used, eg: GS, MMHC, etc.
   (c) 'task_type': Single Task or Multi-Task
   (d) 'score': BIC, etc.
2. Run 'n_runs' and calculate the different performance metrics :
   (a) 'loading_data_durations' : time it takes to load the dataset
   (b) 'learning_durations' : time it takes for the algorithm to learn the structure
   (c) 'shd_scores': SHD scores
   (d) 'cache_usage_total' : Cache used during learning.
3. Log all results inside a Rapidjson array
4. Save the results alongside the experiment parameters inside .json file

4.3.2 k-Single Task (k-ST) and Multi-Task (MT)

The protocol for benchmarking both the k-single task and Multi-task learning are a bit different. The k-datasets used were delivred by the LS2N team. The protocol for the experiments is as follows :

1. Specify the parameters of the experiment :
   (a) ‘n_runs‘: number of runs to execute
   (b) ‘n_tasks‘: Number of Tasks (k)
   (c) ‘algorithm‘ : the algorithm used, eg : GS, MMHC, etc.
2. Run ‘n_runs‘ and :
   (a) Start learning for ‘n_tasks‘ to calculate the different performance metrics:
      i. ‘loading_data_durations‘: Total time it takes to load the k datasets
      ii. ‘learning_durations‘: the time it takes for the algorithm to learn the structures for k tasks
      iii. ‘shd_score‘ : sum of shd_score of k learned networks
3. log all results inside a Rapidjson array
4. save the results alongside the experiment parameters inside .json file

4.4 Results

4.4.1 Single Task (ST)

Comparison between different cache values

In our experiment results, we've logged the use of cache scores and their impact on the learning duration of the GS algorithm. During our first experiments, we noticed strange behavior in the 1st run learning duration time, which was identified as a leaky cache and properly fixed.

cache	loading_data_durations	loading_bn_durations	learning_durations	shd_scores	tps	fps	tns	fns	reverses	cache_usage_total
0	5.921856	0.000036	132.408359	5.0	6.0	2.0	18.0	2.0	0.0	-1.0
1000	6.269562	0.000044	21.285311	5.0	6.0	2.0	18.0	2.0	0.0	112.0
100000	6.646858	0.000044	21.253133	5.0	6.0	2.0	18.0	2.0	0.0	112.0
1000000	6.489800	0.000046	20.822084	5.0	6.0	2.0	18.0	2.0	0.0	112.0

Table 4.3: Grouped learning time per cache value

As we can notice in the results, the cache does indeed increase the performance by a factor of 6x. However the cache used is valued at 112, and increasing the cache value from that doesn't amount to any substantial changes in performance.

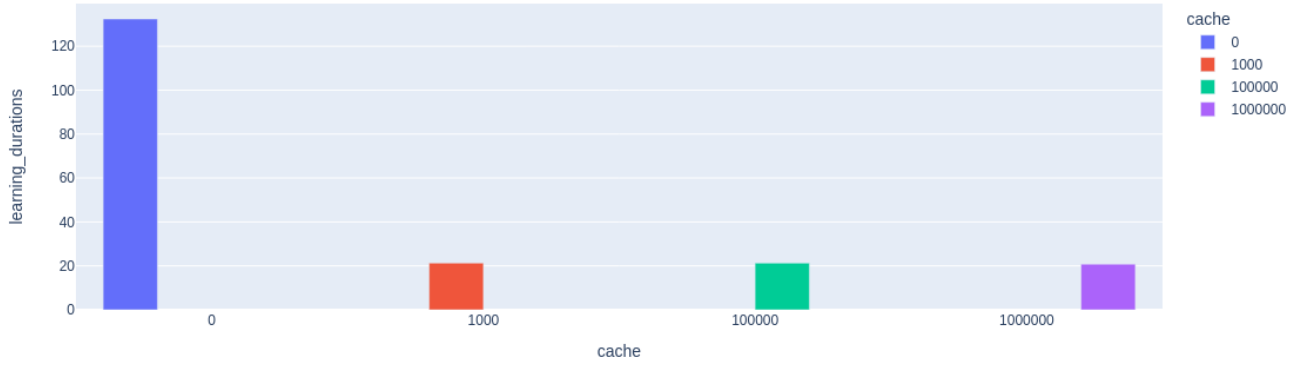


Figure 4.1: Average learning time per cache value

4.4.2 k Single Task (k-ST)

Comparison between different number of tasks

We've benchmarked the k-ST-GS and k-ST-MMHC algorithms for 100 runs and for k=1 to k=10. The objective was to validate the behavior of those two algorithms and confirm the output results. We can observe that the learning time increases in a linear fashion when we

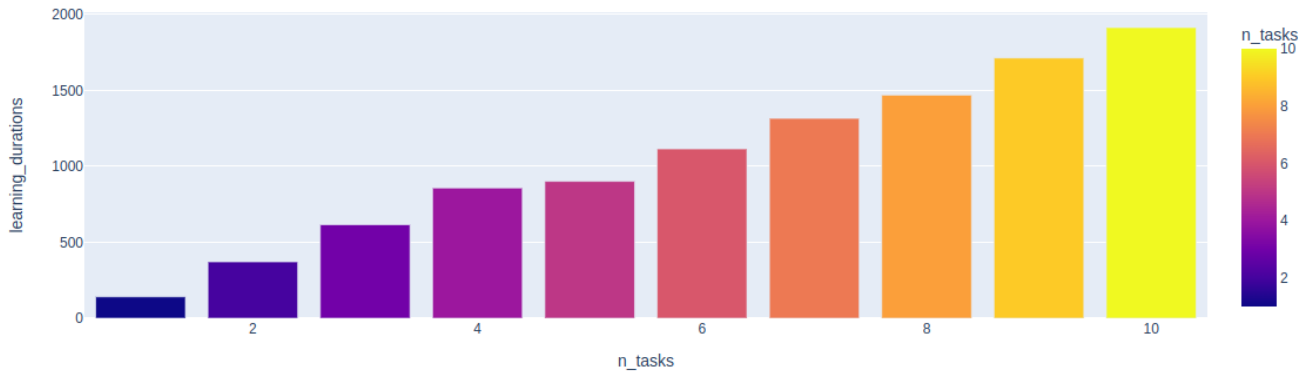


Figure 4.2: k-ST-GS Total learning time for k tasks (averaged on 100 runs)

increase the number of tasks. Furthermore, we can also observe that the average SHD score reaches its max for k = 5 and stays almost constant for k > 5.

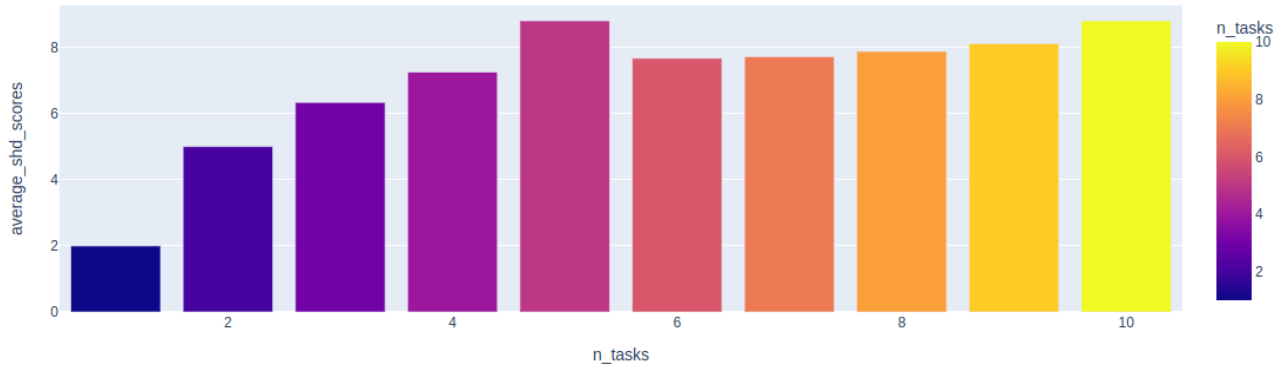


Figure 4.3: kST-GC: Average SHD scores for different number of tasks (Averaged on 100 runs)

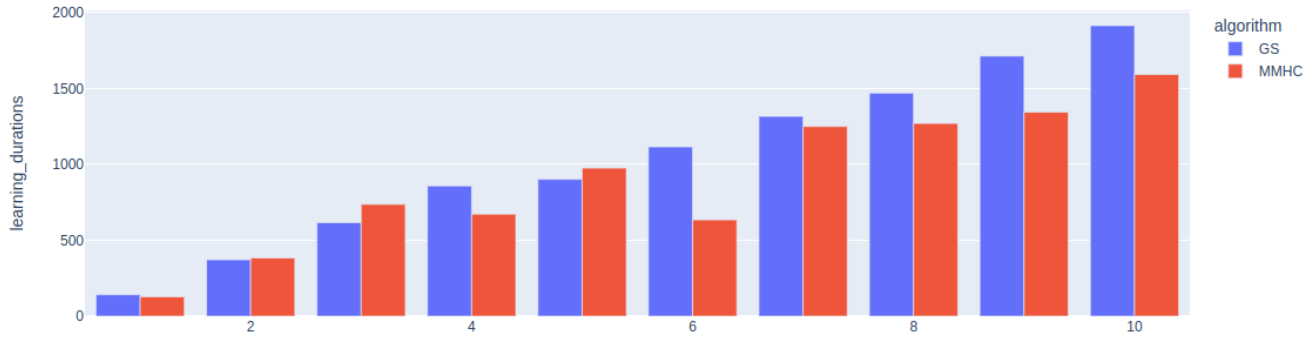


Figure 4.4: k-ST comparison between GS and MMHC Total learning time for k tasks (averages on 100 runs)

We've also performed the same benchmarks for the K-MMHC algorithms, we can observe that k-MMHC is faster for $k > 5$ and that its average SHD score is lower for $k > 3$. However, the objective of our work is to rather compare the performances of k-ST-GS and k-ST-MMHC to the Multi-Task.

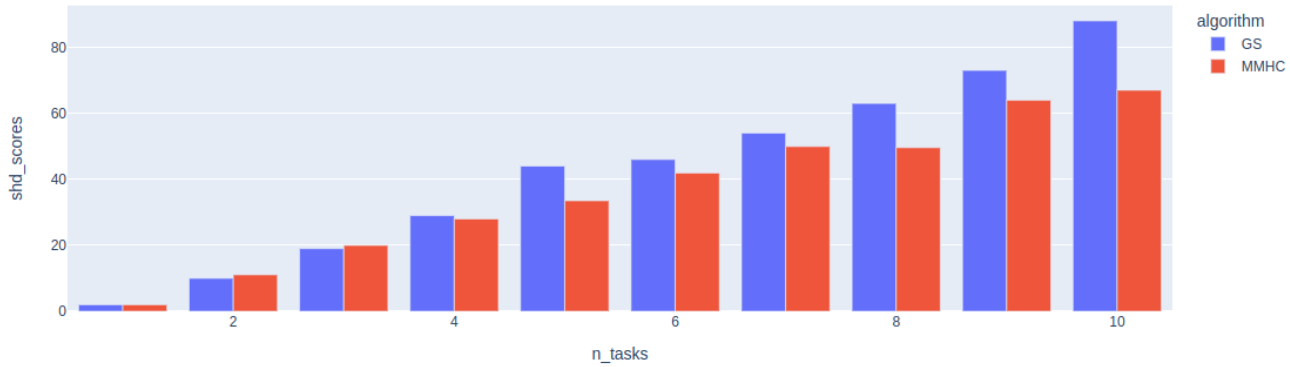


Figure 4.5: k-ST comparasion between GS and MMHC Average SHD for k tasks (averaged on 100 runs)

4.4.3 Multi-Task (MT)

Comparison between different number of tasks

We've benchmarked the algorithms for 100 runs and for k=1 to k=8. We could not benchmark them for any higher values to do lack of computer processing power. The objective was to validate the results of in [1] and compare the performance to the k-ST results we obtained in the previous section.

n_tasks	cpc_durations	loading_data_durations	loading_bn_durations	learning_durations	shd_scores
1	0.119818	70.197792	0.0	232.188810	1.0
2	0.347423	355.598074	0.0	1055.440044	3.0
3	0.664452	688.660892	0.0	2499.779308	6.0
4	1.157698	1431.407139	0.0	4847.900331	11.0
5	2.575414	2478.245670	0.0	12425.550262	15.0
6	5.511110	3987.054286	0.0	29146.253882	16.0
7	4.688785	5427.293086	0.0	33645.908047	19.0
8	5.332844	5182.163442	0.0	35605.730972	22.0

Table 4.4: MT-GS results for k tasks (averaged on 100 runs)

We've added an additional metric *cpc_durations* which is the duration necessary to set the full CPCs because we initially suspected that it was impactful on the total learning time. However, this is not the case as we can observe in the previous table.

n_tasks	tps	fps	tns	fns	reverses	cache_usage_total	average_shd_scores	average_learning_duration
1	8.0	1.0	19.0	0.0	0.0	0.0	1.000000	232.188810
2	16.0	3.0	37.0	0.0	0.0	0.0	1.500000	527.720022
3	23.0	5.0	55.0	1.0	0.0	0.0	2.000000	833.259769
4	28.0	7.0	73.0	4.0	0.0	0.0	2.750000	1211.975083
5	37.0	12.0	88.0	3.0	0.0	0.0	3.000000	2485.110052
6	45.0	13.0	107.0	3.0	0.0	0.0	2.666667	4857.708980
7	51.0	14.0	126.0	5.0	0.0	0.0	2.714286	4806.558292
8	58.0	16.0	144.0	6.0	0.0	0.0	2.750000	4450.716372

Table 4.5: MT-GS additional results for k tasks (averaged on 100 runs)

Comparison between K-ST-GS, K-ST-MMHC and MT-GS

We can observe that the total learning time of MT-GS is increasing exponentially when we increase the number of tasks k compared to k-ST-GS and k-ST-MMHC which is explained by the fact that MT-GS explores a combinatorially larger space.

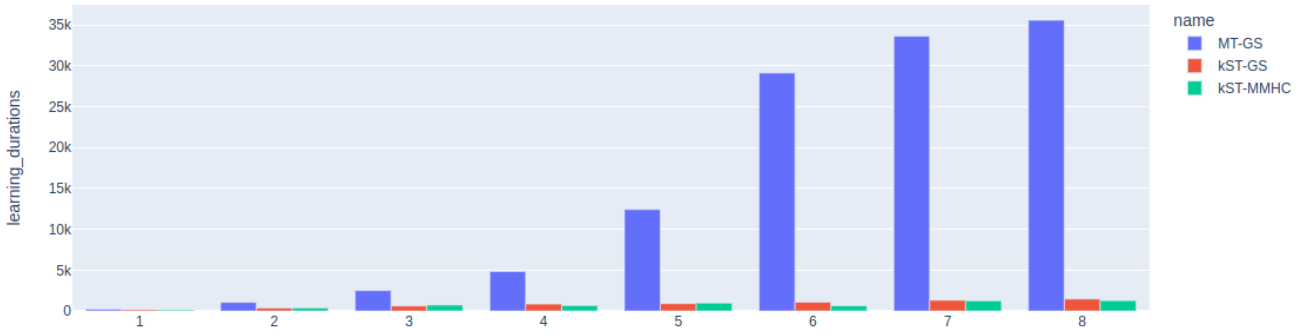


Figure 4.6: Comparaison between k-ST-GS and k-ST-MMHC and MT-GS Total Learning time for k tasks (averaged on 100 runs)

However we can also observe that MT-MMHC outperforms both k-ST-GS and k-ST-MMHC in terms of Average SHD scores which confirms the results obtained in [1].

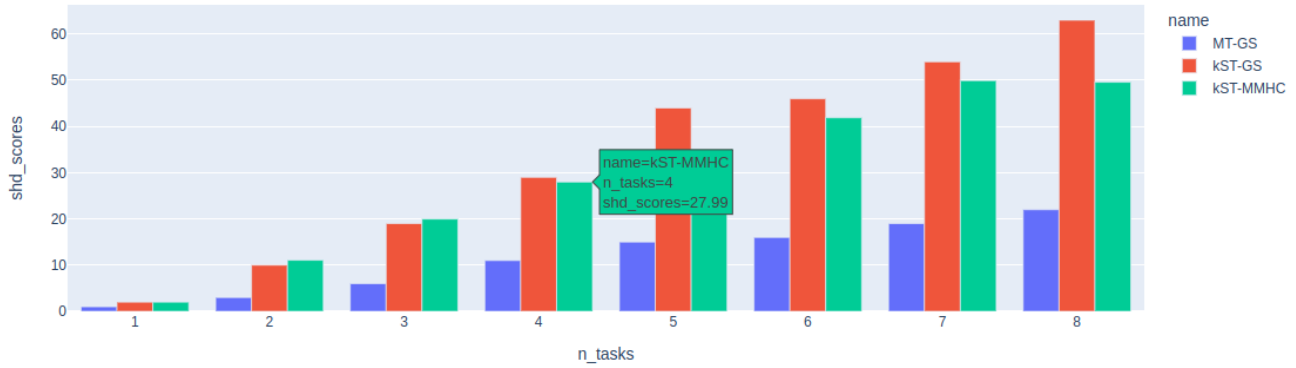


Figure 4.7: Comparaison between k-ST-GS and k-ST-MMHC and MT-GS Average SHD for k tasks (averaged on 100 runs)

Bibliography

[1] S. Benikhlef, P. Leray, G. Raschia, M. B. Messaoud, and F. Sakly. Multi-task transfer learning for bayesian network structures. In J. Vejnarová and N. Wilson, editors, *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, pages 217–228, Cham, 2021. Springer International Publishing.

[2] C. C. Clogg and L. A. Goodman. Latent structure analysis of a set of multidimensional contingency tables. *Journal of the American Statistical Association*, 79(388):762–771, 1984.

[3] L. M. de Campos and J. G. Castellano. Bayesian network learning algorithms using structural restrictions. *International Journal of Approximate Reasoning*, 45(2):233–254, 2007.

[4] A. Fernández, J. D. Nielsen, and A. Salmerón. Learning bayesian networks for regression from incomplete databases. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 18(01):69–86, 2010.

[5] M. Hornỳ. Bayesian networks. *Boston University School of Public Health*, 17, 2014.

[6] P. Larranaga, M. Poza, Y. Yurramendi, R. H. Murga, and C. M. H. Kuijpers. Structure learning of bayesian networks by genetic algorithms: A performance analysis of control parameters. *IEEE transactions on pattern analysis and machine intelligence*, 18(9):912–926, 1996.

[7] D. T. LS2N. Pilgrim.

[8] S. Moral, R. Rumí, and A. Salmerón. Mixtures of truncated exponentials in hybrid bayesian networks. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty: 6th*

European Conference, ECSQARU 2001 Toulouse, France, September 19–21, 2001 Proceedings 6, pages 156–167. Springer, 2001.

[9] A. Niculescu-Mizil and R. Caruana. Inductive transfer for bayesian network structure learning. In *Artificial intelligence and statistics*, pages 339–346. PMLR, 2007.

[10] V. Romero, R. Rumí, and A. Salmerón. Learning hybrid bayesian networks using mixtures of truncated exponentials. *International Journal of Approximate Reasoning*, 42(1-2):54–68, 2006.

[11] S. Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[12] M. Scanagatta, A. Salmerón, and F. Stella. A survey on bayesian network structure learning from data. *Progress in Artificial Intelligence*, 8:425–439, 2019.

[13] M. Scutari, C. Vitolo, and A. Tucker. Learning bayesian networks from big data with greedy search: computational complexity and efficient implementation. *Statistics and Computing*, 29(5):1095–1108, 2019.

[14] I. Tsamardinos, L. E. Brown, and C. F. Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Machine learning*, 65:31–78, 2006.