Confidentiality-Preserving Collaborative Bayesian Networks

Abele Malan

TU Delft

Confidentiality-Preserving Collaborative Bayesian Networks

by

Abele Malan

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday August 24, 2023 at 13:00.

Student number:	4764307		
Project duration:	November 14, 2022 – August 24, 2023		
Thesis committee:	Dr. L. Chen,	TU Delft,	supervisor
	Dr. J. Decouchant,	TU Delft	supervisor
	Dr. T. Guzella,	ASML,	supervisor
	Dr. B. Kulahcioglu Özkan,	TU Delft,	committee

This thesis is confidential and cannot be made public until January 24, 2024.

An electronic version of this thesis is available at `https://repository.tudelft.nl/`.

TU Delft

Preface

This thesis was a joint project between the university and ASML, and I thank both parties for their help and support. Specifically, I want to thank my supervisors in the distributed systems group, Lydia Chen and Jérémie Decouchant, for their invaluable guidance through the process and positive attitude alongside the many other group members I have had the pleasure to interact with and learn from. Just as importantly, I want to thank my ASML supervisor, Thiago Guzella, for his unrivaled availability and consistent feedback alongside his team and colleagues for their warm welcome during my time with them. I also want to profoundly thank my family and friends for their patience, understanding, and encouragement during these months and in general. Furthermore, I want to thank Burcu Kulahcioglu Özkan for being very approachable and agreeing to be part of my thesis committee.

Abele Malan
Delft, August 2023

Contents

1 Introduction **1**

2 Research Paper **3**

3 Background **15**

3.1 Bayesian Networks . . . 15

3.1.1 Related Model Types . . . 15

3.1.2 Structure and Parameter Learning . . . 16

3.1.3 Discretization . . . 17

3.1.4 Probabilistic Inference . . . 18

3.2 Secure Computation . . . 20

3.2.1 Homomorphic Encryption . . . 20

3.2.2 Secret Sharing Schemes . . . 21

4 Additional Experiments **23**

4.1 Probability Function Merge Operator . . . 23

4.2 Variable Elimination Ordering Heuristic . . . 24

4.3 GPU-Accelerated Inference . . . 26

5 Conclusion **29**

1 Introduction

As probabilistic graphical models, Bayesian networks allow great flexibility in data used during modeling and inference, particularly relevant in manufacturing use cases [32, 30, 22, 50, 20] for industries like semiconductor production [51] or steelmaking [31]. They explicitly yet compactly represent interactions between features by expressing them as nodes in a graph where edges denote probabilistic relations [25]. The creation of Bayesian networks can happen without relying on data instances when unavailable or of subpar quality, but prior knowledge exists about the target system. Having experts directly encode domain-specific knowledge in the model once can make later analysis more accessible and reduce the need for their involvement. Furthermore, inference outputs can be any non-overlapping subset of the features. Thus, any available observations help refine estimates for concepts of interest.

Collaborative learning entails creating a global model for some domain by merging information that multiple parties, like different expert groups in a manufacturing setting, contribute. The general problem is prevalent in machine learning, partly due to its real-world applicability, as information extractable from a single party may often not suffice to achieve an appropriate model. Many variations exist, each with its challenges. Most, however, make some assumptions about data samples available within parties but aim to maintain its confidentiality. Data partitioning amongst parties is commonly considered horizontal (i.e., having the same features but different instances) or vertical (i.e., having the same instances but different features). Furthermore, features shared amongst a (sub-)set of parties can either be identically and independently distributed or non. Similarly, each party may either get (a personalized version of) the final model, or it can remain split between them, making analysis a distributed process [28].

The collaboration challenge increases when supporting Bayesian networks that may not be learned from data and contain only partly overlapping features while simultaneously considering confidentiality at the model and data level. Intuitively, information must be merged at the model level when certain parties have a model but no underlying data instances. Furthermore, even if all parties contain local observations, but they have different features and are not fragments of the same instances, merging models could still be a worthwhile strategy, given the lack of data commonality. Fortunately, the graphical backbone of Bayesian networks makes models naturally composable, at least structurally, to some degree. Most importantly, should party Bayesian networks explicitly encode confidential information, any combined model would likely still contain much of it unaltered and readable. Thus, simply storing a complete model in parties should be forbidden.

Federated learning is a ubiquitous collaboration approach, studied in the context of various model types, that protects the locality of party data from which it learns [27, 53]. For Bayesian networks, most existing work [33, 17, 21] focuses on horizontal data splits and learns only the graph structure without probability function definitions. The method of [1] additionally explores vertical splits, but only for structure. Finally, [11] learns a complete model strictly on vertical partitions.

Some works investigate the direct combination of probabilistic graphical models with varying levels of assumptions about their properties. As a base approach, [13] studies intersection and union, albeit only from a structural standpoint. Also, it assumes networks are equally important and, to avoid cycles, the existence of a shared ancestral ordering for the union of the combined network's nodes, under which each node comes before its parents from all networks. Later, [16] proposes an approach between the union and intersection while also giving a procedure for combining probability functions but maintaining

the shared ancestral ordering requirement and equal weighting assumption. Similarly, [43] gives an alternative version of [16] using simulated annealing to forego the ancestral ordering condition. More recently, [2] outlines a way of combining the general class of causal models, which includes Bayesian networks, in a weighted manner, without updating parameters but with stricter compatibility clauses.

Others have trialed additional methods for incorporating party knowledge in specific contexts. A human-in-the-loop strategy is showcased in [47], which in devising a methodology for assisted diagnostics, creates a domain-specific language for working with models backed by Bayesian networks that also support composition but relies on user input for resolving inconsistencies. The system of [44] chains models under the assumption of a discrete timeline where they get used one at a time in some given order, refining their priors based on information gathered from earlier steps. Despite the confidentiality advantages of not having to exchange networks, many workloads do not limit themselves to localized interactions with Bayesian networks or at least cannot adhere to a predetermined ordering. Furthermore, two-way interactions between features from different models are possibly only approximated by reusing them in different chain parts.

This thesis covers multi-party Bayesian network modeling and analysis, primarily in data-scarce and model-confidential scenarios. Specifically, it looks to confidentially join party models via common features without additional data, restricting compatible network types, or involving a trusted coordinator.

The research questions are:

1. Can predictive performance comparable with a classic centralized combination be achieved in such a collaborative, confidential environment?

2. What would be the overhead of a solution exhibiting the desired properties?

3 What are some of the choices influencing probabilistic inference prediction quality and overhead?

The thesis consists of three main parts. The first is a research paper that presents the main contributions in the form of the proposed `CCBNet` framework and results, alongside further motivation from the steel manufacturing industry. It aims to address the first two research questions. The second chapter provides extra insight into concepts relevant to the paper's content, like probabilistic graphical models, namely Bayesian networks, and secure computation methods, namely homomorphic encryption and secret sharing schemes. The third chapter contains additional experiments showcasing possible differences between alternative (implementation) approaches for certain framework parts, aiming to address the third research question.

2

Research Paper

CCBNet: Confidential Collaborative Bayesian Networks Inference

Abstract

Effective large-scale process optimization in manufacturing industries requires close cooperation between different parties of human experts who encode their knowledge of related domains as Bayesian network models. For example, parties in the steel industry must collaboratively use their Bayesian networks on process parameters at the maker, steel properties, and application demands at the client to identify process optimizations effectively. However, business confidentiality across domains hinders collaboration, demanding alternatives to centralized inference. We propose `CCBNet`, the first **C**onfidentiality-preserving **C**ollaborative **B**ayesian **Net**work inference framework. `CCBNet` leverages secret sharing to securely perform analysis on the combined knowledge of party models by joining two novel subprotocols: `CABN`, which augments probability distributions for features across modeling parties into secret shares of their normalized combination; and (ii) `SAVE`, which aggregate party inference result shares through distributed variable elimination. We extensively evaluate `CCBNet` on nine public Bayesian networks. Our results show `CCBNet` achieves similar predictive quality to centralized methods while preserving model confidentiality. We finally demonstrate that `CCBNet` scales to challenging manufacturing use cases, where involving many (16-128) parties in large networks (223-1003 features), on average, enables 45% less computation while communicating 251k values/request.

Introduction

Improving productivity and quality standards in manufacturing demands effectively expressing complex interaction between domain items. Bayesian networks (BNs) are commonly adopted to graphically model causality in manufacturing (Nannapaneni, Mahadevan, and Rachuri 2016), with nodes representing features and directed edges showing dependencies. An essential trait of these models is the ability to run inference queries with arbitrary inputs and outputs.

Let us consider the steel industry. Steelmakers must collaborate with clients to better serve their needs and supplier to improve production efficiency (Miśkiewicz and Wolniak 2020) while protecting trade secrets on all sides. Specifically, steelmakers encode their insight about settings dictating the production process and its outcome in a BN. At the same time, clients craft BNs describing the effects of the different steels' properties (e.g., durability, ductility, corrosion resistance) on their products (e.g., appliances, automotive, railroad). Figure 1 illustrates such a scenario. Pooling together parties' knowledge would allow new kinds of higher-quality analysis for optimizing production environments, leading to new business opportunities.

Existing studies on collaborative inference for BNs disregard model confidentiality constraints or make concessions about which party information can be merged and how. Many focus on centralized scenarios that combine local BNs' knowledge into a larger one (Del Sagrado and Moral 2003; Feng, Zhang, and Shaoyi Liao 2014) without protecting confidential knowledge within the input networks and global output. Models get stitched together based on common nodes, remaining in the final version as largely unaltered sub-models whose encoded knowledge is easily inspectable. (Pavlin et al. 2010) proposes a distributed combination variant that partially preserves confidentiality by maintaining the locality of combined party models. However, it leaks information between parties when connecting them and does not allow merging inner graph nodes with both parents and children. The method of (Tedesco et al. 2006) maintains confidentiality about how nodes are linked within parties but only allows propagating information between them in a fixed sequential order. (Kim and Ghahramani 2012) is another distributed approach with similar confidentiality properties but even greater compatibility restrictions by requiring models to share inputs and outputs.

In this paper, we propose `CCBNet`, the first confidential, collaborative BNs inference framework that combines knowledge of multiple parties involved in inference queries through a novel secret sharing scheme. `CCBNet` does not require a trusted third party, protecting confidentiality at the level of both party models and data instances. The two key components of `CCBNet` are: (i) confidential sharing of a normalized combination of features' probability distributions across all overlapping parties; and (ii) distributed inference based on variable elimination for aggregating party results. The novelty of the augmentation procedure lies in constructing discrete conditional probability distributions for all features present in more than one party, which represent secret shares of a combined and normalized distribution from a centralized scenario without exposing any party's initial probability function. Augmentation constructs shares for a

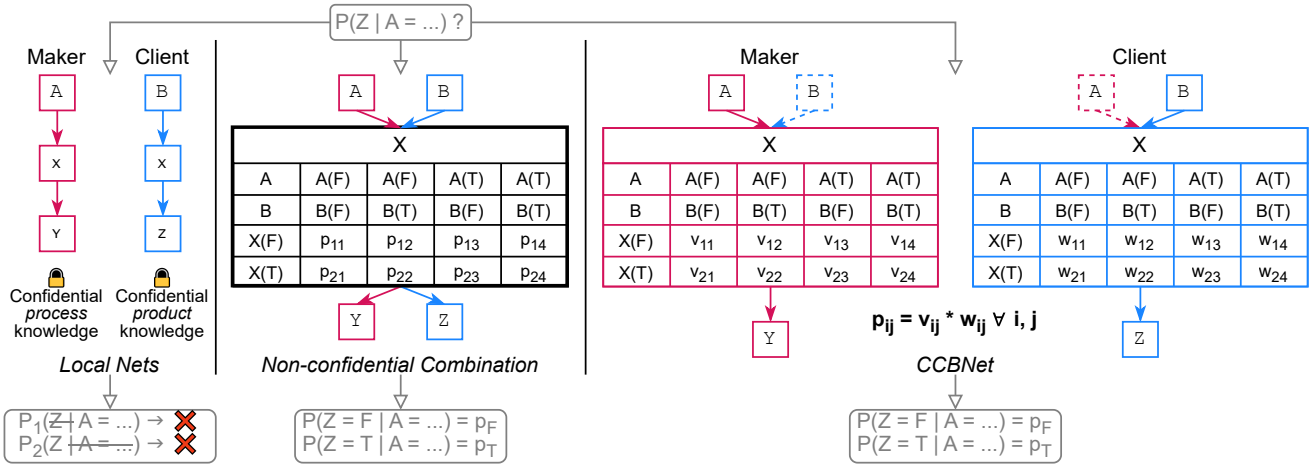


Figure 1: **Sample Bayesian networks collaboration in steel industry:** Parties' separate models overlap within feature X, ductility. Maker features A and Y inform about carbon content and storage requirements. Client features B and Z inform about operating temperatures and max load for the steel-based product. After observing A, inferring updated state probabilities for Z requires propagating information between A, B, and X, which is impossible by merely averaging model outputs. The typical non-confidential approach reveals the combined graph and probability tables to parties. `CCBNet` exchanges minimal structural information amongst parties, and secret shares overlap feature tables, reconstructing the final result by distributed inference.

feature are consistent within the parties modeling it, while the distributed inference variant joins the shares together after parties perform the necessary local computation on them. We use different public BNs as references and simulate knowledge compartmentalization to evaluate `CCBNet`.

In summary, we make the following contributions:

- We design the first confidential, collaborative framework for BNs, `CCBNet`, to satisfy the needs of industry use cases like process optimization in manufacturing.
- We design a novel secret sharing-based protocol, `CABN`, to confidentially augment the conditional probability function of overlapping features in parties.
- We define `SAVE`, a new method for performing distributed inference on augmented party models backed by variable elimination that aggregates their result shares.
- We evaluate `CCBNet` over various scenarios based on nine public BNs to show similar predictive accuracy to non-confidential centralized alternatives and, for many collaborators in large networks, an average 45% computation decrease with 251k communicated values/request.

Background

The following paragraphs outline BN notions relevant to this work. We illustrate specific discussed points via Figure 1.

Bayesian Networks are probabilistic graphical models that maintain explicit conditional probability distributions (CPDs) for features whose dependencies form a directed acyclic graph (Stephenson 2000). Within such context, features are often referred to as (probabilistic) variables or (graph) nodes. Shown in Figure 1, features and the influences between them give the graph nodes and edges, respectively. Principally for performance and interpretability, features in practical applications are generally discrete, with CPDs specifically embodying conditional probability tables. Learning may be algorithmic, by human experts, or hybrid (Koller and Friedman 2009; Daly, Shen, and Aitken 2011), as with other human-readable models like decision trees. Automated learning discovers the graph structure and then populates CPD parameters from training data. Manual learning is desirable when incorporating concepts with known governing rules that need no approximation from observations. Examples are physical phenomena or, in manufacturing, human-engineered processes and tools, like Party 2's scanner wafer table from Figure 1.

Inference in BNs finds updated posterior probabilities for the states of target variables, given the observed states of any evidence variables (Russell 2010; Pearl 2009). As general inference is NP-Hard, approximate algorithms help decrease computation costs compared to exact ones while sacrificing some precision in the result. The main exact inference techniques are variable elimination and junction tree belief propagation, which decomposes the network into a tree of variable clusters, running variable elimination within them and then disseminating updates between neighbors by message-passing (Koller and Friedman 2009). In Figure 1's query, Z is the target, given some observed state of A.

Computation in discrete BNs relies on a few base operations for propagating information: normalization, reduction, marginalization, and products (Koller and Friedman 2009). We outline them with help from the non-confidential combination in Figure 1. Normalizing a CPD divides its entries by their column sum. Thus column summations, like $p_{11} + p_{21}$, would equal 1. Reduction and marginalization remove variables from a CPD by fixing their states or, respectively, summing them out. Reducing or normalizing A from X would

leave B as its sole parent. Flattening CPD structures yields factors that specify a value for each state combination of their variables without discriminating between the child and parents. Previous operations apply to both representations. Products operate on CPDs of the same variable or factors and create a new CPD/factor over the input variables' union, where each entry is the multiplication of the corresponding ones in the original representations. The product of X and Z's factors would, thus, additionally contain A and B.

Markov random fields (MRFs) are a generalization of BNs, backed by undirected graphs, into which every BN is easily transformable via moralization (Li 2009; Scutari and Denis 2021). Apart from lacking acyclicity constraints, MRFs directly define parameters as factors and can deal with scenarios where directionality is unspecified, but BNs are more compact and efficient for generative use. For inference, BN properties and algorithms remain applicable.

Prior Art

We identify two high-level categories of collaborative analysis for BNs: single- and multi-model. The first synthesizes one global model, while the others use multiple local ones.

Single-model approaches harness party data instances or models but neglect confidentiality. Federated learning discovers models (Ng and Zhang 2021; Gao et al. 2021; Huang et al. 2022; Abyaneh et al. 2022; van Daalen et al. 2022) from private party instances with a coordinator, but fully decentralized methods exist (Campbell and How 2014; Gholami, Yoon, and Pavlovic 2016). Direct network combination fuses structure (Del Sagrado and Moral 2003; Alrajeh, Chockler, and Halpern 2020) and also parameters (Feng, Zhang, and Shaoyi Liao 2014) from party models.

Multi-model methods have parties work together during inference to produce a complete analysis result. The systems of (Tedesco et al. 2006) chains model without exchanging their contents but only allows using them one at a time in a predetermined order. Less confidential but more flexible, (Pavlin et al. 2010) fuses party networks based on common nodes but still requires them to be roots or leaves in the party's directed acyclic graph. The patent of (Ypma, Koopman, and Middlebrooks 2018) envisions a similar solution for industrial processes, also mentioning anonymization to help preserve data confidentiality. The approach of (Kim and Ghahramani 2012) runs models autonomously and only averages their final outputs, maintaining confidentiality but expecting models to accept the same inputs and outputs.

In summary, single-model techniques break confidentiality by centralizing knowledge, and multi-model ones trade modeling power for it. `CCBNet` addresses both concerns.

CCBNet

We propose a two-part framework for secure distributed analysis over a related set of confidential, discrete BNs, regardless of learning method and local data instance availability. The first augments party BNs through overlapping variables, and the second performs joint inference on them.

The assumptions we make are that features from different parties have the same name only if they represent the same concept, and independently treating distinct parents for the same node across parties still yields a reasonable approximation of the ground truth. These are shared by previous BN combination works. Thus, names identify the overlapping (common) nodes between models, giving the contact points for graph fusion. Since features modeled by parties may be any subset of those from the full domain, modeling direct interactions between their non-overlapping variables requires great amounts of often unavailable information.

Our adversarial model includes semi-honest parties that follow the protocol while trying to abuse gained information (Goldreich 2005) but do not collude. Further, no trusted third party exists. The goal is to protect all network parameters and only disclose common nodes' structure/state information amongst parties containing them.

Algorithm 1: `CABN`

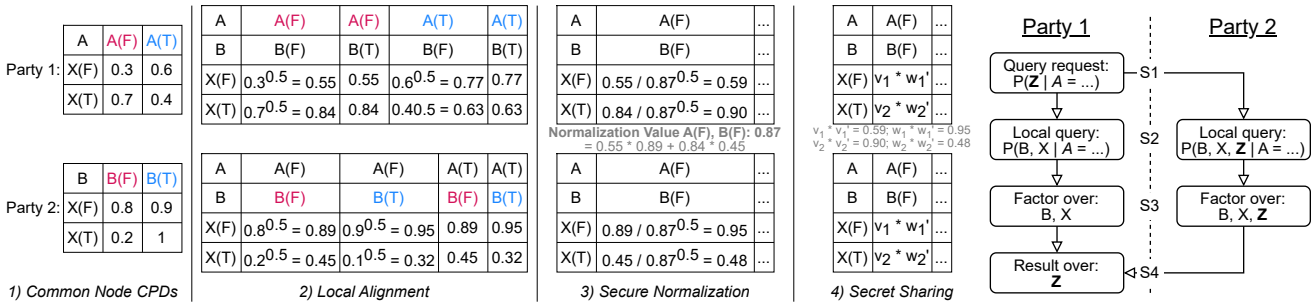
for pX, pY ← Parties × Parties **do**
  for *node* ← *PrivateNodeIntersect(pX, pY)* ***do***
    overlaps[node] ∪ ← {*pX, pY*}
for node, parties ← overlaps **do**
  states ← $\bigcup_{p \in parties}$ p.*obfuscatedStatesCPD*(node)
  idCPD ← *IdentityCPD*(node, states)
  weightSum ← $\sum_{p \in parties}$ p.weight
  for p ← parties **do**
    p.CPD[node] ∗ ← idCPD
    p.CPD[node] ∗∗ ← p.weight / weightSum
  for col ← idCPD.cols **do**
    colsHE ← $\bigcup_{p \in parties}$ *HE*(p.CPD[node | col])
    normVal ← *L1HadamardProdHE*(colsHE)
    for p ← parties **do**
      p.CPD[node | col] / ← normVal$^{1/|parties|}$
  MultipicSecretShare($\bigcup_{p \in parties}$ p.CPD[node])

Confidentially Augmented Bayesian Networks

We now present the `CABN`[1] protocol, which updates local CPDs for overlap variables to hold secret shares of their normalized central combination while protecting the initial probabilities.Algorithm 1 details the four steps of the protocol, illustrated in Figure 2b: (i) private common node identification; (ii) local alignment; (iii) secure normalization; and (iv) secret sharing. The protocol updates parties whenever (enough) changes exist in local models to propagate.

Overview. Structurally, `CABN` imitates a union of the involved networks, like in (Del Sagrado and Moral 2003), and parameter-wise, it uses the same process as (Feng, Zhang, and Shaoyi Liao 2014) but replaces the superposition operator with the geometric mean. We use the union instead of the more complex ruleset of (Feng, Zhang, and Shaoyi Liao 2014) for deciding which overlapping node parents to retain because the more straightforward logic reduces the surface area for attacks and allows for combining more than two BNs at a time, lowering the number of communication rounds. For fusing probabilities, the geometric mean enables a multiplication-based secret sharing scheme in `CABN` where

[1] **C**onfidentially **A**ugmented **B**ayesian **N**etworks



(a) Key Steps of computing augmented CPDs

(b) Distributed inference flow

Figure 2: CABN & SAVE steps for Figure 1 scenario

reconstruction happens automatically during distributed inference when computing intermediary party factor products. The mean also outperformed the superposition in our centralized tests. Having described the general strategy, we continue with the protocol phases.

Step 1: Private Common Node Identification. CABN starts with party pairs identifying their common nodes like in the central case, albeit privately. We use private set intersection protocol (De Cristofaro and Tsudik 2010) to achieve this. The pseudocode highlights this step in ll. 1-3. When only a subset of parties has updates, they will be the only ones recalculating their intersections with others. Outside the private intersection context, node and state names are communicated obfuscated to prevent information leakage about which nodes are modeled by which party. Parties can choose any unique representation for non-overlapping nodes, but involved parties agree on an obfuscated representation for overlapping ones.

Step 2: Local Alignment. After parties know which own nodes overlap with which peers, they start solving overlaps by exchanging shape and weight information about their local CPDs and independently updating local representations accordingly (ll. 4-10). First, the obfuscated union of CPD nodes and states for overlapping CPDs is determined (l. 5). From it, an identity CPD containing all the parents across parties gets created for the union (l. 6). An identity CPD (or factor) has all entries equal to 1, so its product with another replicates the later's columns over their joint state space. The initial CPD gets replaced by the product with the identity in each party, giving all overlap CPDs the same shape (l. 9), as seen in Figure 2a. Parties have a public weight representing confidence in their BN, which in the default unweighted case is 1. They compute the sum of their weights (l. 7) and individually raise the entries of their CPD to the ratio between their weight and the sum, computing the partial geometric mean (l. 10). By the exponent product rule $(XY)^k = X^k Y^k$, the CPDs' product would already yield the unnormalized central combination CPD.

Model Weighting. As previously explained, CABN allows weighting CPDs through the geometric mean, contrary to previous BN works that cover parameter fusion. A natural integration of unequal weighting of inputs is another advantage of using a geometric mean instead of (Feng, Zhang, and Shaoyi Liao 2014)'s superposition operator. We implement weights at the model level as values from 0 to 1, encoding the human expert's confidence in the network or data availability for algorithmic learning. Nevertheless, if desired, weighting can trivially be applied at the CPD level.

Step 3: Secure Normalization. Homomorphic encryption (HE) (Cheon et al. 2017) helps privately compute column normalization values (ll. 11-15). One party is elected to generate the public/private key pair, while another does the computation. All parties receive the public key and send their encrypted columns (l. 12) to the party that calculates normalization value ciphers (l. 13), which the private key party later receives, decrypts, and shares with the rest. A column normalization value is the sum of entries obtained by multiplying corresponding party columns element-wise. Letting P_{ij} denote party i CPD column j, the calculated value is $||\odot_i P_{ij}||_1$. Then, the K overlapping parties individually divide each column by the K-th root of the appropriate normalization value (l. 14), so their factor product is the normalized geometric mean. Figure 2a shows an example.

Because local columns no longer sum to 1 after exponentiation, even in a two-party overlap where the variable has only two possible states, a party cannot reconstruct the other's entries by only knowing the normalization values and its own entries. Furthermore, vital for HE schemes in practice, we know that the number of consecutive multiplications needed for each column is equal to the party count, which allows configuring the scheme accordingly. Functional encryption, in which completing the desired computation also decrypts the output (Boneh, Sahai, and Waters 2011), would be an even more fitting choice, but existing implementations have overly stringent limits on the number of inputs and complexity of the applied functions. Using a secret sharing scheme (SSS) (Cramer, Damgård, and Nielsen 2015) instead of HE is also possible, but we favor decreasing the communication count over computing overhead for this step. A SSS has the advantage of requiring fewer computational resources and being more robust against collusion. However, it requires communication for each multiplication operation and, depending on the scheme, the presence of a third party. Despite the expectation that CABN needs to run more rarely than inference, we still favor optimizing for message count, as high communication latency is likelier to be a bottleneck

Algorithm 2: SAVE

Input: Q={x, y, ...}, E={a_i, b_j, ...}
Output: Factor

1: auxFacts ← {}
2: **for** party ← Parties **do**
3: partyFacts ← $\bigcup_{cpd \in party.CPDs}$ Factor(cpd)
4: auxFacts ∪ ← {*VarElim*(Q, E, partyFacts)}
5: **return** *VarElim*(Q, {}, auxFacts)

than processing for envisioned deployments.

Step 4: Secret Sharing. Finally, to prevent specific kinds of inference attacks, we secret share (Kilbertus et al. 2018) the CPD entries of parties in each overlap through a multiplication-based scheme (l. 16). The common shape of updated local CPDs facilitates the procedure. In the classic additive secret sharing scheme, a secret value is split into shares distributed amongst parties whose sum is the secret. It allows efficient and secure computation of expressions summing multiple secret values and applying other operations involving non-secret values. Parties perform the computation with their local share of each secret and all aggregate their results to reconstruct the answer. The utilized scheme functions similarly but uses multiplication as the base operation instead. Reconstruction happens during inference as before, with no extra overhead compared to skipping the step since party CPDs already contain different information that needs merging. Figure 2a exemplifies the share splitting.

Handling potential cycles. To avoid compatibility restrictions between combinable BNs, if solving overlaps creates a cycle, the distributed global network gets treated as an MRF, with no changes to the inference, which operates on factors regardless. Edges that form cycles in the BN are effectively incorporated into the moralized MRF and treated as undirected. Since the main target is not to share the complete combined network, the readability advantages of BNs are not applicable. There often needs to be more information to decide which edges to remove from cycles reasonably, and alternatives like treating all nodes within a cycle as a single node (Ypma, Koopman, and Middlebrooks 2018) are coarse-grained and threaten confidentiality.

Share Aggregation Variable Elimination

SAVE[2] is the inference protocol that has all parties run variable elimination locally before aggregating their outputs into the final factor. Algorithm 2 describes its steps, and Figure 2b visualizes them for an example query. The party requesting inference sends the evidence and query variables to all others in obfuscated form (S1 in fig.). Parties run the query locally, adding to the target set their overlapping variables and direct parents (S2 in fig.). Unmodeled variables are ignored. Their intermediate factors, representing shares of the final result, are sent back to the requesting party in obfuscated form (S3 in fig.), which runs a final round of variable elimination for the result after receiving all replies (S4 in fig.). Adding overlap variables and parents to local target sets avoids marginalization before reuniting all information related to them. Marginalization entails summing values, which cannot happen locally with the chosen SSS, so we delay it until implicit share reconstruction within the last variable elimination call at the initiating party.

[2]**S**hare **A**ggregation **V**ariable **E**limination

Class	Name	#Nodes	#Edges	#Params
Small (<20 Nodes)	ASIA	8	8	18
Medium (20-49 Nodes)	CHILD	20	25	230
	ALARM	37	46	509
	INSURANCE	27	52	1008
Large (50-99 Nodes)	WIN95PTS	76	112	574
Very Large (101-999 Nodes)	ANDES	223	338	1157
	PIGS	441	592	5618
	LINK	724	1125	14211
Massive (>= 1000 Nodes)	MUNIN2	1003	1244	69431

Table 1: Evaluation datasets w/ node, edge, parameter counts

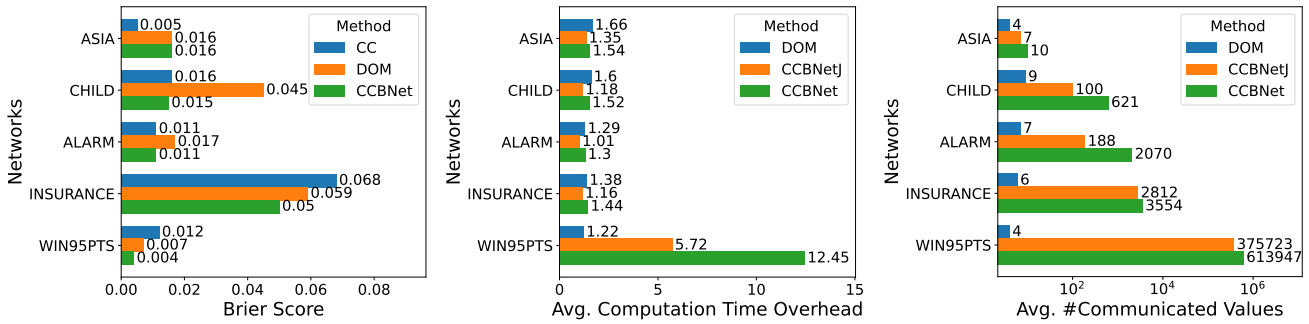
Queryable Nodes. To maintain confidentiality, parties can only specify modeled variables in inference by default, even if the result will still reflect the effect of prior knowledge about others, so we propose mechanisms for expanding the set of possible queries. The first involves all parties that own a node agreeing to expose its unobfuscated name and states with select others to use as a target or evidence. Doing so only requires revealing a node's existence, not its place in the network(s). The other mechanism implicitly enhances evidence with the help of some key shared between parties (e.g., timestamp, product batch identifier). If a query request also includes a value for the shared key, parties will incorporate any observations for the key's value as evidence during their local inference step. Parties do not have to disclose the value of the observed data, but the query output will be the same as if it had been part of the initial evidence.

Communication Properties. The number of messages exchanged within an inference request is of magnitude $O(N)$ (where N is the number of parties), but the size of the messages varies. Regarding count, the requester sends out $N-1$ messages and receives the same amount of replies adding up to $2N-2$. The size of the messages, particularly replies, varies greatly depending on the number and complexity of the responder's overlaps and the query itself.

Performance Evaluation

Setup

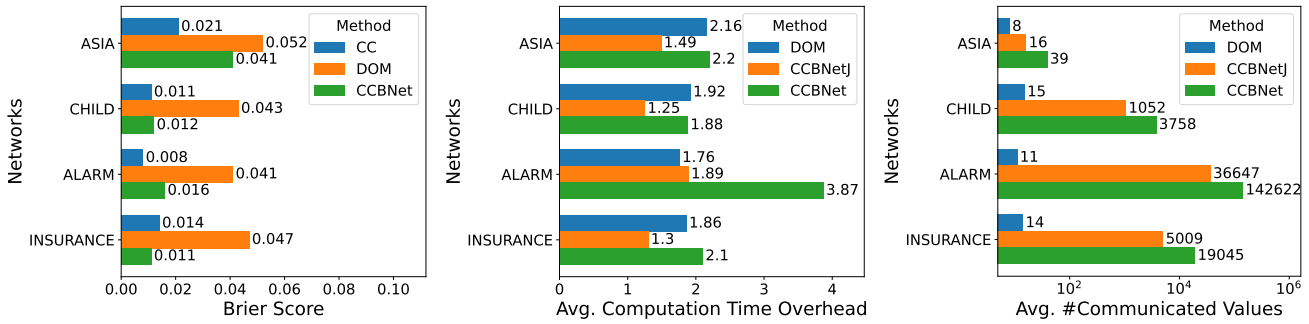
Our experiments evaluate average predictive performance, computation overhead, and communication cost in single-machine simulations. We measure prediction quality via the Brier Score ($= \frac{1}{N}\sum_{t=1}^{N}\sum_{i=1}^{R}(f_{ti} - o_{ti})^2$ where N is the number of queries, R is the number of target variables state combinations, while f and o are predicted and reference probabilities). We report the total processing time ratio between examined methods and the ground truth network for computation overhead. Regarding communication, we consider the count of factor values exchanged per query.



(a) Brier scores relative to original network, related splits

(b) Average computation time overhead relative to original network, related splits

(c) Average #communicated values rounded to nearest integer, related splits (log scale)



(d) Brier scores relative to original network, random splits

(e) Average computation time overhead relative to original network, random splits

(f) Average #communicated values rounded to nearest integer, random splits (log scale)

Figure 3: Results for 4 parties, 30% of vars in >1 party (lower is better for all)

We test on public networks[3] (Table 1), consider two variable splitting methods, and multiple overlap variable ratios. **Related splits** assign to parties variables connected in the ground truth network. **Random splits** ensure parties have equal variable counts and share the same overlaps. Test sets contain 2000 queries, each randomly specifying an overlap variable as the target and fixing 60% of others as evidence.

Baselines

Centralized Combination (CC) iteratively combines parties with (Feng, Zhang, and Shaoyi Liao 2014)'s method while treating the network as an MRF once any cycles form.

Centralized Union (CU), the approach confidentially mimicked by `CCBNet`, is structurally based on the union of (Del Sagrado and Moral 2003), combines parameters via geometric mean, and also applies the MRF principle.

Decentralized Output Mean (DOM) is a naïve approach that takes the geometric mean for each target variable's state probabilities over independently operating contained parties. It trades modeling long-range effects for confidentiality.

`CCBNetJ` is a degenerate `CCBNet` variant that stores the fully combined central CPDs for overlaps in one of the concerned parties, trading some safety for faster inference.

`CCBNet` Performance Overview

Here, we summarize the Brier score, computation time, and communication overhead of `CCBNet` under two splitting methods, different overlapping ratios (10%, 30%, and 50%), and the party number (2, 4, and 8). Due to the space limitation, we present the full results in an appendix and highlight in Figure 3 the performance trend through the representative case of four parties with an overlap ratio of 30%.

Predictive Performance. Regardless of split type, `CCBNet` predictions often outperform or match the classic CC and always beat the naïve DOM. Figure 3a gives results for related splits, `CCBNet` only scores worse than CC on the smallest network (0.016 versus 0.005) and gains a significant advantage over the two largest ones (0.05 versus 0.068 and 0.004 versus 0.012). Apart from a tie on the smallest network, `CCBNet` has an advantage over DOM in all scenarios. Examining the random splits in Figure 3d, `CCBNet` flips to only scoring a win over CC on the largest tested network (0.011 vs 0.014) but extends its lead over DOM in all but the smallest network. Since `CCBNet` yields the same predictive ability as its centralized counterpart CU, the differentiating points with CC are the structure combination policy and parameter combination operator. With related splits, the geometric mean of `CCBNet` fares better than superposition, but CC's parent selection policy is more effective than the union approach over random splits, even with the nondeterminism it involves. Finally, complete test results confirm the expectation that adding more parties tends to decrease per-

[3] https://www.bnlearn.com/bnrepository/

formance while increasing overlaps has the opposite effect.

Computation Overhead. Regarding computation cost relative to centralized inference on the original network, average slowdowns are 1.65x for DOM, 1.82x for CCBNetJ, and 3.15x for CCBNet. Communication latency is unaccounted for as it can vary greatly based on the deployment. Still, we overestimate wall time by summing computation time across parties, as much processing would happen concurrently in reality. In the related splits from Figure 3b, for all networks but the large one, CCBNetJ is the fastest, while others follow closely. The final network is a tipping point for both CCBNet and CCBNetJ, as their overhead (12.45x and 5.72x, respectively) explodes, while DOM observes no change in pattern. The networks also examined for random splits in Figure 3e show a similar trend, with slightly higher general overhead, especially for CCBNet in one medium network (3.87x). Thus, as expected, DOM is less affected by network complexity and overlap variable density, despite the implementation having a higher base overhead. Between the sibling approaches, CCBNetJ is faster than CCBNet but both perform reasonably in most scenarios, although they degrade rapidly when dealing with too many complex overlaps. Further, complete results certify that adding parties improves speed while increasing overlaps decreases it.

Communication Cost. Regarding the number of communicated CPD values per query, DOM averages merely 9, while CCBNetJ and CCBNet need orders of magnitude more at 46k and 85k, respectively. Since the number of communicated values depends on which party initiates a query, the reported figures include communication internal to the initiator to eliminate variability, somewhat overestimating reality. The number of messages to complete a query is the same for all methods. Furthermore, the raw data transmitted in bytes remain in the low megabyte range for hundreds of thousands of values before compression. Figure 3c shows the mentioned discrepancy over related splits for all but the smallest network, in which the three methods are comparable. DOM merges complete party outputs and cannot propagate evidence between parties. Thus, it does not increase communication with the number of overlaps, and parties that do not contain any target variables send empty replies. The situation for random splits, illustrated in Figure 3f, is very similar, although the disadvantage of CCBNet over CCBNetJ widens slightly. As for adding parties and increasing overlaps, the complete appendix results attest that they both increase communication overhead for all methods.

Party Weighting

Table 2 shows the weighted version of the proposed method having better predictive performance than the unweighted one in most scenarios with random splits. In weighting tests, we reduce the overall amount of data used for learning local BNs to ensure more variance and randomly assign each party a fraction of the max training data. Over the few scenarios where the unweighted version performs better, parties with lower data get overly punished for its perceived imprecision.Since each CPD has a single weight, all parents of a node within the party are still treated uniformly according to that value, even if there is a mismatch between it and the actual quality of the estimates. Similarly, if a party with lower overall confidence is the only one to model a highly-influential parent, its importance will be somewhat misrepresented in the final result. Nevertheless, the weighting has a positive, albeit contained, overall impact in tested scenarios.

#Parties		2				4			
Vars in >1 Party		10%		30%		10%		30%	
Method		UW	W	UW	W	UW	W	UW	W
Network	ASIA	0.051	**0.031**	0.052	**0.03**	**0.238**	0.251	0.166	**0.145**
	CHILD	**0.128**	0.13	0.14	**0.107**	0.196	**0.19**	0.133	**0.118**
	ALARM	0.086	**0.068**	0.077	**0.061**	**0.117**	0.12	0.16	**0.153**
	INSURANCE	0.056	**0.054**	0.137	**0.111**	0.206	**0.2**	0.185	**0.175**

Table 2: Unweighted **(UW)** & Weighted **(W)** Brier scores for CCBNet relative to original network - random splits, imbalanced learning data (lower is better)

Networks/	Brier Score			Avg Comp Time Overhead			Avg #Comm Values		
Parties	CC	DOM	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet
ANDES/16	0.051	0.047	**0.041**	0.56	0.6	0.67	4	4368	10651
PIGS/32	0.073	0.083	**0.064**	0.36	0.78	0.93	6	742527	745361
LINK/64	0.107	0.104	**0.098**	0.23	0.3	0.33	6	62636	67470
MUNIN2/128	0.017	0.16	**0.014**	0.18	0.21	0.24	11	50705	181859

Table 3: Metrics for large networks & many parties - related splits, 10% of vars in >1 party (lower is better for all)

Large Networks & Many Parties

Lastly, in Table 3, our tests for challenging use cases with large networks (223-1003 features), many parties (16-28), and related variable splits, but lower overlaps reconfirm prediction/communication trends, yet computation improves over original networks. As previously, in larger networks, CCBNet's predictions beat CC, and communication size increases with parties and network size, averaging 251k values/request. Computation overhead is always <1 (i.e., a speedup) by 45% on average, as the hardness of inference makes approximating a big network by splitting it into chunks faster, even before considering parallel party solving.

Conclusion

We propose CCBNet to address the issue of collaborative analysis for BNs in confidential (manufacturing) settings. The framework allows distributed analysis spanning involved models without revealing their contents. It has no model compatibility restrictions and allows unequally weighting parties. We extensively evaluate the method and a lower-overhead but less robust variant, CCBNetJ, against non-confidential central approaches and a naïve, distributed, confidential formulation. Results show CCBNet having predictive performance similar to the naïve option and similar centralized approaches. Inference computation overhead is often reasonable, barring challenging scenarios with large networks with many overlapping party variables. However, in specific situations, our approach can even reduce total computation. Compared to the minimal interaction naïve formulation, messages are many times but remain acceptable in most cases, and their count is equal. Future work could improve model merger quality (e.g., by exploiting extra party data instances when updating probability functions) and reduce communication costs.

References

Abyaneh, A.; Scherrer, N.; Schwab, P.; Bauer, S.; Schölkopf, B.; and Mehrjou, A. 2022. FED-CD: Federated Causal Discovery from Interventional and Observational Data.

Alrajeh, D.; Chockler, H.; and Halpern, J. Y. 2020. Combining experts' causal judgments. *Artificial Intelligence*, 288: 103355.

Annamalai, M. S. M. S.; Gadotti, A.; and Rocher, L. 2023. A Linear Reconstruction Approach for Attribute Inference Attacks against Synthetic Data. arXiv:2301.10053.

Boneh, D.; Sahai, A.; and Waters, B. 2011. Functional Encryption: Definitions and Challenges. In Ishai, Y., ed., *Theory of Cryptography*, 253–273. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-19571-6.

Campbell, T.; and How, J. P. 2014. Approximate Decentralized Bayesian Inference. arXiv:1403.7471.

Cheon, J. H.; Kim, A.; Kim, M.; and Song, Y. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In Takagi, T.; and Peyrin, T., eds., *Advances in Cryptology – ASIACRYPT 2017*, 409–437. Cham: Springer International Publishing. ISBN 978-3-319-70694-8.

Clauset, A.; Newman, M. E. J.; and Moore, C. 2004. Finding community structure in very large networks. *Physical Review E*, 70(6).

Cramer, R.; Damgård, I. B.; and Nielsen, J. B. 2015. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press.

Daly, R.; Shen, Q.; and Aitken, S. 2011. Learning Bayesian networks: approaches and issues. *The knowledge engineering review*, 26(2): 99–157.

De Cristofaro, E.; and Tsudik, G. 2010. Practical Private Set Intersection Protocols with Linear Complexity. In Sion, R., ed., *Financial Cryptography and Data Security*, 143–159. Berlin, Heidelberg: Springer Berlin Heidelberg.

Del Sagrado, J.; and Moral, S. 2003. Qualitative combination of Bayesian networks. *International Journal of Intelligent Systems*, 18(2): 237–249.

Feng, G.; Zhang, J.-D.; and Shaoyi Liao, S. 2014. A novel method for combining Bayesian networks, theoretical analysis, and its applications. *Pattern Recognition*, 47(5): 2057–2069.

Gao, E.; Chen, J.; Shen, L.; Liu, T.; Gong, M.; and Bondell, H. 2021. FedDAG: Federated DAG Structure Learning.

Gholami, B.; Yoon, S.; and Pavlovic, V. 2016. Decentralized Approximate Bayesian Inference for Distributed Sensor Network. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1).

Goldreich, O. 2005. Foundations of Cryptography – A Primer. *Foundations and Trends® in Theoretical Computer Science*, 1(1): 1–116.

Huang, J.; Yu, K.; Guo, X.; Cao, F.; and Liang, J. 2022. Towards Privacy-Aware Causal Structure Learning in Federated Setting.

Kilbertus, N.; Gascon, A.; Kusner, M.; Veale, M.; Gummadi, K.; and Weller, A. 2018. Blind Justice: Fairness with Encrypted Sensitive Attributes. In Dy, J.; and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, 2630–2639. PMLR.

Kim, H.-C.; and Ghahramani, Z. 2012. Bayesian Classifier Combination. In Lawrence, N. D.; and Girolami, M., eds., *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, 619–627. La Palma, Canary Islands: PMLR.

Koller, D.; and Friedman, N. 2009. *Probabilistic graphical models: principles and techniques*. MIT press.

Li, S. Z. 2009. *Markov random field modeling in image analysis*. Springer Science & Business Media.

Li, T.; Sahu, A. K.; Talwalkar, A.; and Smith, V. 2020. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Processing Magazine*, 37(3): 50–60.

Miśkiewicz, R.; and Wolniak, R. 2020. Practical Application of the Industry 4.0 Concept in a Steel Company. *Sustainability*, 12(14).

Nannapaneni, S.; Mahadevan, S.; and Rachuri, S. 2016. Performance evaluation of a manufacturing process under uncertainty using Bayesian networks. *Journal of Cleaner Production*, 113: 947–959.

Ng, I.; and Zhang, K. 2021. Towards Federated Bayesian Network Structure Learning with Continuous Optimization.

Pavlin, G.; de Oude, P.; Maris, M.; Nunnink, J.; and Hood, T. 2010. A multi-agent systems approach to distributed bayesian information fusion. *Information Fusion*, 11(3): 267–282. Agent-Based Information Fusion.

Pearl, J. 2009. *Causality*. Cambridge University Press, 2 edition.

Russell, S. J. 2010. *Artificial intelligence a modern approach*. Pearson Education, Inc.

Scutari, M.; and Denis, J.-B. 2021. *Bayesian networks: with examples in R*. CRC press.

Stephenson, T. A. 2000. An introduction to Bayesian network theory and usage. Technical report, Idiap.

Tedesco, R.; Dolog, P.; Nejdl, W.; and Allert, H. 2006. Distributed Bayesian Networks for User Modeling. In Reeves, T.; and Yamashita, S., eds., *Proceedings of E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2006*, 292–299. Honolulu, Hawaii, USA: Association for the Advancement of Computing in Education (AACE).

van Daalen, F.; Ippel, L.; Dekker, A.; and Bermejo, I. 2022. VertiBayes: Learning Bayesian network parameters from vertically partitioned data with missing values.

Ypma, A.; Koopman, A. C. M.; and Middlebrooks, S. A. 2018. Methods & apparatus for obtaining diagnostic information, methods & apparatus for controlling an industrial process.

Zhang, J.; Cormode, G.; Procopiuc, C. M.; Srivastava, D.; and Xiao, X. 2017. PrivBayes: Private Data Release via Bayesian Networks. *ACM Trans. Database Syst.*, 42(4).

Algorithm 3: Related Split

Input: bnDAG, nrSplits, nrOverlaps, randGen
Output: Splits

dfsTree ← *DFSTree*(bnDAG)
splits ← *GreedyModularityComms*(DFSTree, nrSplits) (Clauset, Newman, and Moore 2004)
shuffledEdges ← randGen.*shuffle*(bnDAG.edges)
initNodeSplit ← {}
for splitNr ← 1 . . . nrSplits **do**
  for node ← splits[nrSplit] **do**
    initNodeSplit[node] ← splitNr
ovNodes ← { }
connSplits ← { }
extraEdges ← [ ]
for nodeO, nodeI ← shuffledEdges **do**
  if |ovNodes| ≥ nrOverlaps **then**
    break
  es ← {initNodeSplit[nodeO], initNodeSplit[nodeI]}
  if |es| == 1 **then**
    continue
  if |connSplits| < nrSplits AND es ⊆ connSplits **then**
    extraEdges += (nodeO, nodeI)
  else
    ovNodes ∪= {nodeO, nodeI}
    connSplits ∪= es
    splits[initNodeSplit[nodeO]] ∪= {nodeO}
    splits[initNodeSplit[nodeI]] ∪= {nodeI}
for nodeO, nodeI ← extraEdges **do**
  if |ovNodes| ≥ nrOverlaps **then**
    break
  ovNodes ∪= {nodeO, nodeI}
  splits[initNodeSplit[nodeO]] ∪= {nodeO}
  splits[initNodeSplit[nodeI]] ∪= {nodeI}
return splits

Experiment Method & Results Details

We detail the procedures used for related and random splits of ground truth variables amongst parties, used throughout all experiments, in Algorithm 3 and Algorithm 4, respectively. Table 4, Table 5, and Table 6. Regarding additional experiment results with related splits, Table 4 covers predictive performance, Table 5 covers computation overhead, and Table 6 covers communication cost. In a few networks under related splits, adjacent overlap figures (e.g., 30% and 50%) have the same splits and inherently score, either because of the small number of nodes (ASIA) or because all possible overlaps for the given topology are formed (CHILD, ALARM). Table 7 shows extra results for all three metrics under random splits. For choosing the order of variable elimination during inference, we use a min neighbors heuristic, which greedily chooses a variable such that the product of factors containing it has the smallest size.

We ran each experiment once, with a fixed seed determining the randomness for sampling data instances from the reference network and splitting it into overlapping variables sets of parties. In terms of computing infrastructure, we utilized 16 threads of an Intel(R) Xeon(R) Gold 6134 CPU @ 3.20GHz (note that inference in the single-machine simulation was single-threaded), 120 GB of RAM, and RedHat Enterprise Linux 7.9 OS. We implemented the codebase in Python 3.10, using pgmpy 0.1.22[4] as the backbone for BNs in our framework, tenseal 0.3.14[5] for homomorphic encryption, and openmined.psi 2.0.1[6] for private set intersection.

Algorithm 4: Random Split

Input: bnDAG, nrSplits, nrOverlaps, randGen
Output: Communities

shuffledNodes ← randGen.*shuffle*(bnDAG.nodes)
ovs ← randGen.*sample*(shuffledNodes, nrOverlaps)
splits ← *SplitEqualParts*(shuffledNodes, nrSplits)
for split ← splits **do**
  split ∪= ovs
return splits

Attacks on `CCBNet`

To recap, a classic combination of related BNs, which encode confidential information into a single global model, has a very high risk of leaking information to all parties with direct access to it. As seen in the toy example from Figure 1, even at a purely structural level, the centralized combination from the middle part can contain much, if not all, of the local party information. Furthermore, at a parameter level, probability functions for any non-overlap nodes remain unmodified. Since BNs are human-readable, inspection can compromise sensitive information before any inference.

Barring encrypting the model or only allowing access to it through a trusted party, no existing protections directly target the model itself. Defensive (Zhang et al. 2017) and offensive (Annamalai, Gadotti, and Rocher 2023) measures for Bayesian models consider the privacy of training dataset instances. Similarly, differential privacy (DP) is a generally applicable technique that injects minimal noise into the model to achieve the desired level of privacy (Li et al. 2020). However, constructing a complete model and applying DP to it before distributing it to the parties still involves a trusted third party, and, unlike network parameters, its structure would be unchanged, breaking confidentiality.

We briefly review two attacks to reconstruct CPDs during `CABN` and `SAVE`, respectively, along with their implications in `CCBNet` and `CCBNetJ`. The attacks do not bypass the obfuscation of unowned variable names and states but still expose potentially sensitive information via the recovered probability values. We successfully execute the attacks on ASIA and CHILD network instances involving two parties.

`CABN` Attack. The first attack concerns only `CCBNetJ` in two-party overlap scenarios and has the attacker reconstruct the private CPD of its overlap coutnerpart. Since in `CCBNetJ`, one party recreates the combined CPD before inference, if only two parties are involved, the holder can remove its contribution from the result and apply the `CABN`

[4]https://github.com/pgmpy/pgmpy
[5]https://github.com/OpenMined/TenSEAL
[6]https://github.com/OpenMined/PSI

	#Parties	2									4						8								
	Vars in >1 Party	10%			30%			50%			10%			50%			10%			30%			50%		
	Method	CC	DOM	CCBNet	CC	DOM	CCBNet	CC	DOM	CCBNet	CC	DOM	CCBNet	CC	DOM	CCBNet	CC	DOM	CCBNet	CC	DOM	CCBNet	CC	DOM	CCBNet
Network	ASIA	**0**	0.008	0.001	**0**	0.008	0.001	**0**	0.012	0.001	**0.005**	0.016	0.016	**0.005**	0.015	0.011	**0.068**	**0.068**	**0.068**	**0.068**	**0.068**	**0.068**	**0.046**	**0.046**	**0.046**
	CHILD	0.026	0.039	**0.015**	**0.001**	0.02	**0.001**	**0.001**	0.02	**0.001**	**0.104**	0.128	**0.104**	0.014	0.03	**0.01**	**0.177**	0.194	0.178	**0.087**	0.105	**0.087**	0.063	0.054	**0.037**
	ALARM	**0.001**	0.017	0.005	**0.001**	0.017	0.005	**0.001**	0.017	0.005	0.036	0.04	**0.025**	**0.002**	0.011	0.004	**0.021**	0.045	0.029	0.039	0.044	**0.035**	0.037	0.024	**0.014**
	INSURANCE	**0.012**	0.026	0.016	0.009	0.018	**0.007**	0.009	0.01	**0.007**	0.106	0.112	**0.104**	0.06	0.054	**0.047**	0.134	0.14	**0.133**	0.146	0.147	**0.143**	0.082	0.083	**0.077**
	WIN95PTS	0.003	0.002	**0.001**	0.017	0.007	**0.004**	0.017	0.007	**0.004**	**0.003**	0.006	0.004	0.013	0.008	**0.005**	0.045	0.036	**0.031**	0.021	0.018	**0.014**	0.013	0.014	**0.007**

Table 4: Brier score relative to original Network - related splits (lower is better)

	#Parties	2									4						8								
	Vars in >1 Party	10%			30%			50%			10%			50%			10%			30%			50%		
	Method	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet
Network	ASIA	1.56	1.22	1.37	1.56	1.23	1.48	1.69	1.37	1.46	1.65	1.48	1.53	1.7	1.25	1.57	1.83	1.61	1.79	1.82	1.6	1.78	2.22	1.64	1.98
	CHILD	1.18	1.05	1.13	1.42	1.08	1.27	1.36	1.08	1.31	1.21	1.11	1.17	1.74	1.18	1.7	1.36	1.23	1.3	1.57	1.27	1.58	1.92	1.34	1.94
	ALARM	1.13	0.99	1.05	1.11	0.99	1.06	1.11	0.99	1.06	1.11	0.97	1.09	1.37	1.06	1.43	1.15	1.03	1.1	1.33	1.1	1.34	1.61	1.21	1.67
	INSURANCE	1.14	0.98	1.07	1.32	1.08	1.31	1.44	1.7	2.54	1.16	0.99	1.08	1.66	1.74	2.82	1.18	1.08	1.15	1.32	1.15	1.35	1.65	1.21	1.66
	WIN95PTS	1.03	0.98	1.07	1.15	101.23	137.92	1.17	100.66	137.46	0.91	0.85	0.91	1.32	51.33	141.93	0.9	0.84	0.92	1.14	1.03	1.82	1.44	1.29	19.27

Table 5: Average computation time overhead relative to original network - related splits (lower is better)

	#Parties	2									4						8								
	Vars in >1 Party	10%			30%			50%			10%			50%			10%			30%			50%		
	Method	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet
Network	ASIA	4	6	8	4	6	8	4	8	11	4	7	10	4	11	15	4	10	12	4	10	12	4	11	14
	CHILD	9	47	78	7	122	230	7	122	230	9	34	54	9	144	575	9	22	34	8	40	97	9	131	526
	ALARM	6	43	78	6	43	78	6	43	78	7	63	199	7	843	5277	8	59	79	6	172	321	7	2961	4849
	INSURANCE	6	66	108	6	862	1656	6	31784	63349	6	44	73	7	25821	57966	6	36	48	6	221	338	8	395	1195
	WIN95PTS	4	1693	3365	4	4266160	8532220	4	4266160	8532220	4	107	129	4	2407170	4658690	4	49	145	4	897	60102	5	8646	887145

Table 6: Average #communicated values rounded to nearest integer - related splits (lower is better)

	Metric	Brier Score									Average Computation Time Overhead									Average #Communicated Values								
	#Parties	2						4			2						4			2						4		
	Vars in >1 Party	10%			30%			10%			10%			30%			10%			10%			30%			10%		
	Method	CC	DOM	CCBNet	CC	DOM	CCBNet	CC	DOM	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet
Network	ASIA	0.06	**0.025**	0.026	0.038	0.017	**0.016**	**0.005**	0.068	**0.005**	1.45	1.28	1.38	1.67	1.33	1.52	1.9	1.36	1.55	4	11	15	4	17	25	8	7	8
	CHILD	0.034	**0.021**	0.025	**0.007**	0.019	**0.007**	0.04	0.053	**0.037**	1.21	1.07	1.16	1.4	1.15	1.37	1.36	1.14	1.37	6	130	233	7	1038	1879	12	85	275
	ALARM	**0.009**	0.022	0.013	**0.011**	0.017	0.012	**0.037**	0.06	**0.037**	1.07	1	1.12	1.27	1.62	2.16	1.22	1.02	1.25	6	667	1298	6	23332	46368	11	448	1574
	INSURANCE	0.012	0.013	**0.007**	0.008	0.019	**0.006**	0.03	0.033	**0.013**	1.1	1.02	1.12	1.24	1.26	1.61	1.31	1.05	1.3	7	443	843	7	9356	16496	13	376	1366

Table 7: Results for random splits (lower is better for all)

steps (except secret sharing) in reverse to retrieve the other peer's original CPD. Although secret sharing augmented local CPDs protects them during the computation of the combined version, once the combiner knows the result alongside one of the two inputs, it can find the other. `CCBNet` is not vulnerable, as it does not join shares before applying inference operations. A minimal-change fix for `CCBNetJ` in the assumed no-collusion setting is to store the combined CPD for overlaps between two parties in another party if available. Name obfuscation ensures secrecy toward the additional party, but it still gains some insight, like the node's final parent count and which parties model it. For overlaps with three or more parties, the attacker can only reconstruct a mix of the other overlap parties' CPDs.

`SAVE` Attack. The second attack builds upon the first, utilizing specific inference patterns to threaten `CCBNet` (and consequently `CCBNetJ`), also in two-party overlaps. By querying for a known overlap variable and specifying states for all its parents as evidence, an attacker makes other parties containing the variable return the corresponding column from their secret-shared local CPDs. Repeating the procedure for all combinations of parent states retrieves the complete secret-shared local CPDs from overlap parties. By the nature of secret sharing, these do not encode any confidential information. However, since their product yields the combined CPD, the previously described attack becomes applicable in two-party overlap cases. A simple avoidance strategy involves parties refusing to serve peer queries that target two-party overlap nodes and fix all their parents as evidence.

3
Background

3.1. Bayesian Networks

A Bayesian network (BN) is a type of probabilistic graphical model represented by a directed acyclic graph structure, with features as nodes and dependencies amongst them as edges, and parameters in the form of a probabilistic variable for each node, specifying a conditional probability distribution (CPD) over its parents [42]. Figure 3.1 shows a small example. Its explicit yet compact representation of features and their relationships allows it to accept queries with arbitrary inputs and outputs while remaining suitable for construction and inspection involving human experts. The flexibility comes at a computation cost, as even in the classic setting where CPDs are represented in tabular form, modeling only discrete features, learning, and inference are NP-Hard [8, 7]. `CCBNet` also focuses on the discrete case. Overviews of related models, learning, discretization, and inference follow.

3.1.1. Related Model Types

A Markov random field (MRF) is a sibling model to a BN, which instead of a DAG, is based on a cyclic undirected graph [26]. Within a BN, a CPD discriminating between the parents and child corresponds to each node in the standard discrete case. Meanwhile, an MRF directly uses factors that do not make that distinction, so a node is associated with all factors containing it. Any BN is transformable into an MRF through moralization, which involves removing the directionality of existing edges and adding new ones between all parents of a node. Figure 3.2 has an example. CPDs become factors by flattening their structure, shown in the right-hand side of Figure 3.3. Although largely overlapping in capabilities, both models have specific advantages. For example, MRFs offer enhanced flexibility by allowing cyclic dependencies or when dealing with problems where the direction of dependencies is undetermined. When it comes to generating data instances by sampling, BNs are much more suitable thanks to their directionality. An easy procedure, also used to generate party train data from the reference network for `CCBNet` experiments, is forward sampling [25]. It starts with fixing the state of DAG root nodes by randomly sampling their prior probability and recursively repeats the process for children (whose state for all parents has become known) until all leaf nodes also have a state.

Although most real-world use cases work specifically with BNs where all variables are discrete (including `CCBNet`), some usage of variants involving continuous variables exists [49], and the two main such ones both rely on Gaussian variable distributions [38]. The first type, a Gaussian BN, has only continuous variables of the previously mentioned type. In contrast, the second, a Conditional Gaussian BNs, contains discrete and continuous ones, hence sometimes being referred to as hybrid, but discrete variables are not allowed to have continuous parents. Furthermore, an extra condition by which variables are linearly related to their parents applies to both cases, so they are sometimes also referred to as (Conditional) linear Gaussian BNs.

Other BN variants differ mainly from a semantic perspective, and as such, `CCBNet` requires little to no adaptation to accommodate them, with dynamic and causal BNs being two important ones [37]. Dynamic BNs express temporal dependencies within and between discrete time steps. Any variable within some time slice may depend on other variables from slices not later than its own. By extension, BNs not dealing with time series data are described as static. Causal BNs are a restricted class of BNs

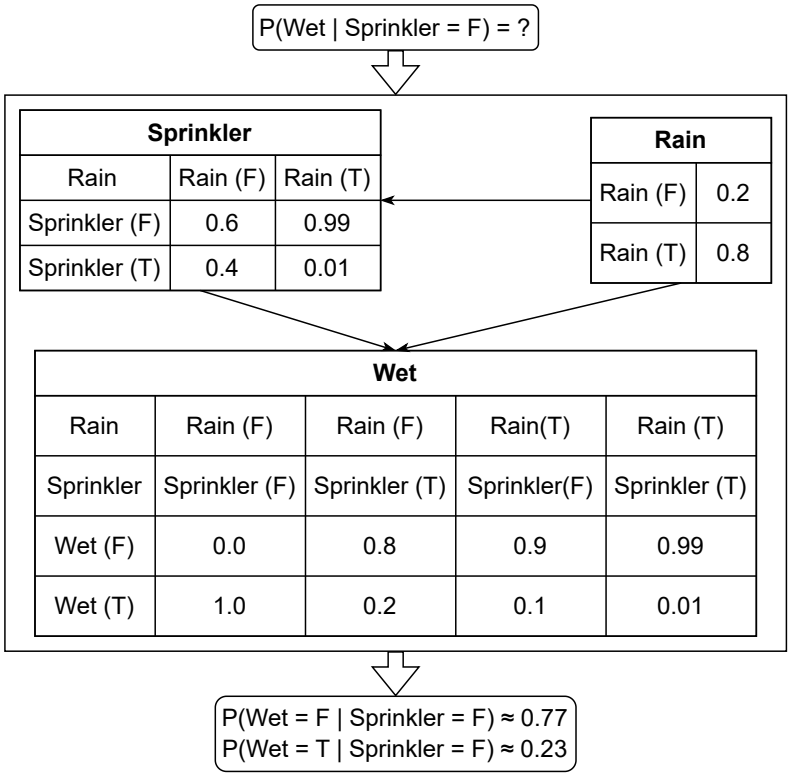


Figure 3.1: Example Bayesian network & inference request

that specifies that the dependencies between BN nodes encoded in edges should always correspond to a causal relation [35], which, even to obtain a network encoding equivalent knowledge, does not necessarily have to be the case otherwise. The main benefit of causal representations over any other non-causal equivalent is that it tends to be more compact and more accessible to assess, even though both would yield the same results during analysis.

3.1.2. Structure and Parameter Learning

BN learning is a two-step process: first, structure learning fixes the graph structure in place before parameter learning finds the CPDs for the variable represented within each node. In terms of human involvement, learning can either be fully manual, mixed, or fully automated. Manual learning is favored when experts have extensive pre-existing knowledge to encode into the model. For the mixed case, the most natural scenario involves humans entirely devising the structure or guiding a semi-automated

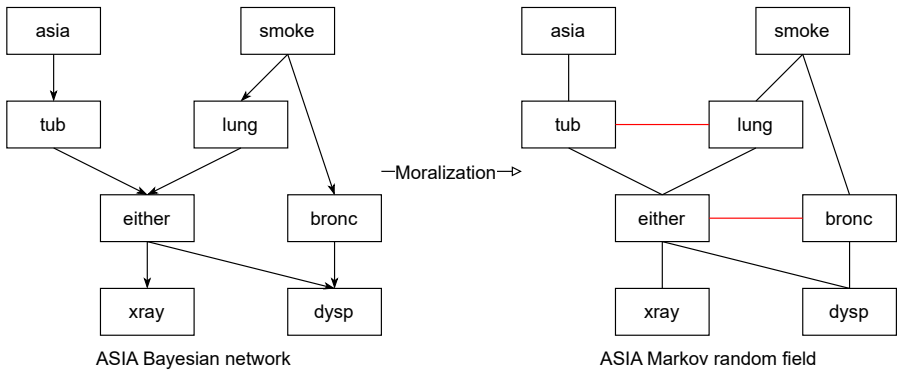


Figure 3.2: Example moralization of ASIA Bayesian network into a Markov random field - existing edges remove directionality, new undirected edges added between variable parents (in red)

process, for example, to ensure the output satisfies causality, while parameter learning likely happens algorithmically. Finally, fully automated learning is most similar to usual machine learning settings, where a computer algorithm for each step learns the corresponding part of the model from a dataset. Finally, as with forests of decision trees, Bayesian model averaging is an approach that takes multiple models and, during analysis, averages their outputs in some way to obtain the final one. However, it requires enough data or expert opinions to create multiple models, and they all have to represent the same feature space, making it often unfeasible despite potentially higher-quality analysis [25].

Several structure learning algorithms exist, falling into three categories: constraint-based, score-based, and hybrid [38]. `CCBNet` experiments use a score-based, greedy hill-climbing approach.

Constraint-based structure algorithms trace their roots back to the Inductive Causation algorithm [48], which learns through conditional independence tests. A variable X is conditionally independent of A given B if $P(X|A,B) = P(X|B)$, which can be interpreted in a graph context as B being on the path between X and A. The algorithm uses these tests to identify which edges from the fully connected undirected graph to discard. Then, with the help of the same tests, it determines, between triples of non-connected nodes that share some neighbor, those where the common node is a child of the other two. Finally, based on the partial directionality information acquired, it iteratively fills out the orientation of the remaining undirected edges. The first practical implementation is the PC algorithm [41, 9]. It involves heuristics for the first two steps determining which of the exponentially many conditional independence tests to perform. Later efforts, like Grow-Shrink [29] and Incremental Association [46], further tweak the utilized heuristics.

Score-based structure algorithms treat learning as an optimization problem where different candidate DAGs are assigned a score based on some chosen function. Thus, possibly within a predetermined computation budget, the algorithm searches for the DAG with the best score possible. The search can be exact, yielding a globally optimal result, or more often heuristic, trading some potential model quality for speed improvements. Greedy heuristics, like hill-climbing, repeatedly add, delete or reverse arcs one at a time, starting from some initial network, possibly employing random restarts to decrease the chances of getting stuck in a local optimum. Heuristics based on genetic algorithms combine parts of parent network pairs into a new child, with a chance of undergoing random mutations. Simulated annealing randomly picks between variations, weighting them based on the change in score.

Hybrid structure algorithms combine the other two techniques to harness their strengths while curtailing their weaknesses but often do not result in better empirical performance [39]. They have two core steps: restrict, which uses a constraint-based method to reduce the DAG search space, and maximize, which uses a score-based method to find the best candidate amongst the remaining ones. One of the most known realizations is the Max-Min Hill-Climbing algorithm [45].

Parameter learning is a more straightforward process thanks to knowing which variables describe each distribution, and the usually applied techniques are Maximum Likelihood and Bayesian estimation [38]. The simpler Maximum Likelihood directly optimizes the likelihood function based on available data instances, while its more complex Bayesian counterpart also accounts for prior and evidence information. `CCBNet` experiments use Bayesian Estimation to learn parameters.

3.1.3. Discretization

Discretization is the most common way to integrate continuous variables into the ubiquitous discrete BN case, and there are three main types: manual, supervised, and unsupervised [4]. In short, it consists of partitioning a continuous variable's input range into some selected amount of consecutive intervals (bins), forming a discrete version. Manual discretization, like learning, is done by humans and has an interpretability advantage by allowing experts to select threshold values of importance to the domain and assign more appropriate names to the intervals. Unsupervised discretization is based solely on the target variable's distribution, and some typical example strategies are equal length, where all intervals span ranges of (approximately) the same size, and equal frequency, where intervals contain (roughly) the same amount of data instances. Supervised discretization requires observing the output of some discrete variable(s) to inform the optimization process. They tend to perform better than unsupervised ones [34], but the auxiliary variable requirement may sometimes be unsatisfiable.

Extending `CCBNet` to perform joint discretization of party variables while preserving confidentiality properties is possible with an unsupervised method like equal length, which exchanges minimal additional information between parties during the process. For each variable present in more than one party, assuming all agree the variable is inherently continuous, they can convene on a discretization,

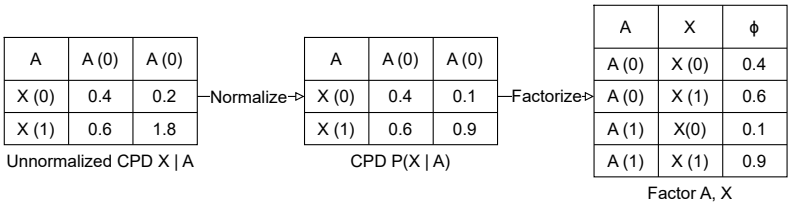
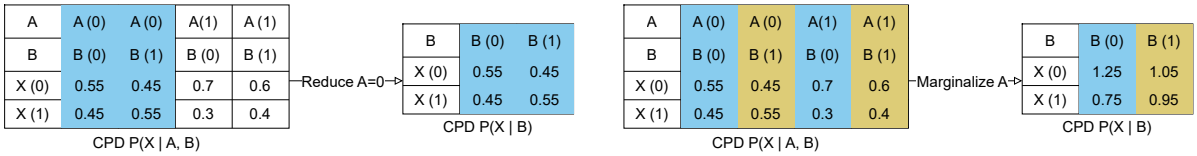


Figure 3.3: Example normalization on CPD & transformation to factor



(a) Reduce A=0

(b) Marginalize A

Figure 3.4: Examples for operations that remove variables on CPD

even if any subset performed discretization locally beforehand. Specifically, the average of some bin count proposed by each, rounded to the nearest integer, could give the final number of bins. Additionally, the bounds for the variable would be the minimum, respectively, the maximum value known across all parties. The protocol would then be on hold until involved parties modify their local models to apply the discretization, later continuing normally. Joint manual discretization would be unfit as it would equate to the involvement of a trusted third party in human form. Supervised methods would also be unsuitable because the different parties are not guaranteed to have a shared discrete variable to observe the output of, and they would, in principle, require more information shared between parties, increasing the risk of leaking undesired information.

3.1.4. Probabilistic Inference

Probabilistic inference in BNs (or MRFs) finds the updated posterior probabilities for a specified set of target variables, given the observed states of any other variables as evidence [25]. Figure 3.1 illustrates a sample query and its output. In the default marginal case, inference outputs one factor spanning all target variables. The result is easily transformable into a set of separate factors, one for each target. Further, each variable's most likely state is trivially obtainable from such a factor set, giving the result for the Maximum a posteriori case. Similar to learning, its high computational cost has led to the development of approximate variants alongside exact ones. `CCBNet` focuses on exact inference.

The two principal exact inference techniques are variable elimination (VE) and junction tree belief propagation, which builds upon VE [38]. Both were ideated for discrete BNs, but have seen adaptations for networks containing continuous variables.

VE takes the network parameters in factor form, fixes the state of evidence variable within them,

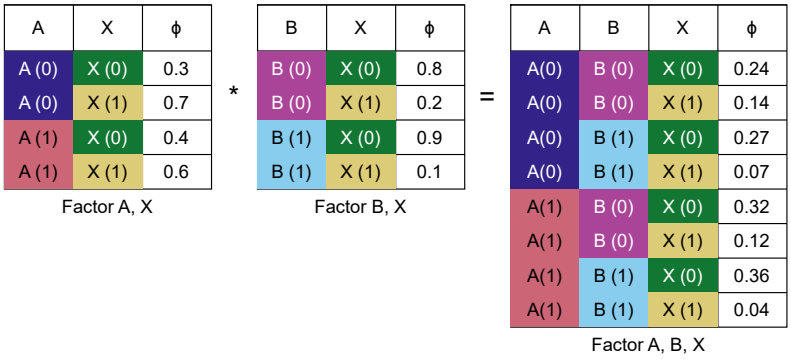


Figure 3.5: Example product between two factors

Algorithm 1 Variable elimination exact inference

Input: Q={x, y, ...}, E={a_i, b_j, ...}, facts=[f_1, f_2, ...]
Output: Factor

```
1: vars ← {v | f ← facts, v ← facts.vars, v ∉ Q ∪ E}
2: workingFacts ← {f.reduce(E) | f ← facts}
3: for v ← VarElimOrderHeuristic(vars, facts) do
4:     vars \= v
5:     relFacts ← {f | f ← workingFacts, v ∈ f.variables}
6:     workingFacts \= relFacts
7:     workingFacts ← Marginalize(Product(relFacts), v)
8: return Normalize(Product(workingFacts))
```

Algorithm 2 MinNeighbors variable elimination ordering heuristic

Input: vars={x, y, ...}, facts=[f_1, f_2, ...]
Output: Variable

```
1: varNrFacts ← {v: 0 | v ← vars}
2: for f ← facts do
3:     for v ← f.vars do
4:         if v ∈ vars then
5:             varNrFacts[v] += 1
6: return MinValKey(varNrFacts)
```

and uses a dynamic programming formulation to iterate through each non-query variable and merge factors that contain it before removing it from the result. In the case of BNs, the factors passed to VE are the converted CPDs. VE forms the backbone of inference in `CCBNet`.

VE only involves the following operations: normalization, reduction, marginalization, and products. The first three apply to CPDs and factors, while the last one applies to any factors but only to CPDs targeting the same variable. Normalization, exemplified in the left-hand side of Figure 3.3, divides the entries of each column within a CPD to ensure they sum up to 1. Since a factor has a single column, its entries get divided by the sum of all. For VE to provide correct results, the starting factors must be normalized. Reduction, shown in Figure 3.4a, fixes the state of a variable, removing it from the CPD or factor and discarding entries in which its state differs. Marginalization, shown in Figure 3.4b, removes a variable from a CPD or factor by summing all entries with the same state instead. A product, illustrated in Figure 3.5, takes two or more CPDs or factors and creates a new CPD or factor over all their variables, where each entry in the output is the multiplication of corresponding input ones.

The VE procedure, outlined in Algorithm 1, takes as input the query and evidence variables alongside a list of factors for the network and outputs a factor with the refined probabilities of requested variables. It starts by searching within factors for variables that are not targets or evidence to find all those it must eliminate (l. 1) and reducing all evidence from factors (l. 2). From there, variables get iterated through, usually in an order imposed by some heuristic (l. 3). In each iteration, the selected variable leaves the set of pending eliminations (l. 4), all relevant factors containing the chosen variable are found (l. 5) and removed from the list of factors (l. 6), while the output of their product after

Algorithm 3 MinWeight variable elimination ordering heuristic

Input: vars={x, y, ...}, facts=[f_1, f_2, ...]
Output: Variable

```
1: varSibblings ← {v:  | v ← vars}
2: for f ← facts do
3:     for v ← f.vars do
4:         if v ∈ vars then
5:             varSibblings[v] ∪= f.vars
6: return MinValKey({v: Prod({NrStates(s) | s ← varSibblings[v]}) | v ← varSibblings})
```

marginalizing the eliminated variable is added to the list (l. 7). After eliminating all variables, the normalized product of all factors left over in the list forms the answer (l. 8). Should only one factor exist in a product chain, the output factor is simply the input.

The order of selecting variables for elimination does not affect the final result. However, the difference in the sizes of resulting intermediate factors can significantly impact runtime. Since all of the $n!$ ways to order n variables can lead to different sets of intermediate factors, the problem of elimination order is itself NP-Hard, like learning and inference, so different heuristics are employed. One is MinNeighbors (Algorithm 2), which greedily selects among the variables to eliminate one which appears in the fewest factors. Another is MinWeight (Algorithm 3), which selects, also greedily, a variable yielding the intermediary factor of the smallest size. The names for both come from the factor graph representation of a network, which contains factors as nodes in addition to variables and connects each variable to the factors in which it appears. Unless otherwise specified, `CCBNet` experiments use MinWeight.

The belief propagation algorithm only works on undirected tree networks and computes inference results by passing messages (which contain intermediate factors from ordinary VE) through the tree. However, by caching messages, effectively storing intermediate factors from VE for longer than the execution of a single inference call, it can answer subsequent queries for different variables with the same evidence much more efficiently.

A junction tree represents any undirected network as a tree. Its nodes contain clusters of variables such that for every factor, some cluster contains all its variables. Additionally, for every cluster pair, any other cluster on the path between them must contain their shared variables.

Junction tree belief propagation brings the benefits of belief propagation to all networks (moralizing them first if directed) with the help of a junction tree representation while running VE within resulting clusters. Finding the junction tree representation for a network most computationally efficient for inference is NP-Hard, so heuristics are employed. Harnessing junction tree belief propagation is unfeasible for `CCBNet`, as the overlapping nodes between parties do not generally yield a tree structure, to begin with, and attempting to create a junction tree by assigning each party one or more variable clusters would require parties to receive information from variables they do not model.

The two principal approximate inference techniques are sampling-based and variational [37]. Sampling, or particle filtering, methods repeatedly draw from the network distribution and estimate the probability for each outcome of the target variables by checking how many of the samples with that outcome also match the evidence. Asymptotically, the results of sampling methods approach the exact one, but estimating the pace of convergence is hard. Variational methods reduce the problem to a simpler one that still tracks the original as close as possible. They tend to produce less precise results but can have faster runtime and often offer bounds on accuracy.

3.2. Secure Computation

The primary secure computation methods in `CCBNet` are homomorphic encryption for computing normalization values and secure multiparty computation via secret sharing for the actual probability table entries. The following first defines some shared basic concepts before giving additional context into the techniques in general and the specific versions employed in the proposed framework.

Assumptions about the attacker's computational power and ability to corrupt parties dictate the security properties of different protective measures, which may also differ in the types of information they can protect. An adversary is passive (semi-honest) if it merely has access to the internal state of corrupt parties and active (malicious) if it can also alter corrupt party behavior [19]. Furthermore, information-theoretic security means an adversary with unlimited computation power can not break the system, while computational security assumes an adversary with bounded computational resources [14]. A dishonest majority setting involves the corruption of more than half of all parties. Finally, some techniques may only be able to protect (and manipulate) integer values directly, so an often-used trick to allow floating-point values involves transforming the latter into the former via fixed-precision encoding.

3.2.1. Homomorphic Encryption

Homomorphic encryption (HE) allows performing computation on encrypted data and getting (almost) the same result upon decrypting the cyphertext as if having operated on plaintext data. Although literature outlines secret key variants [52], where the same secret key handles encryption and decryption, in practice, most rely on the more flexible public key cryptography, where an additional public key exists

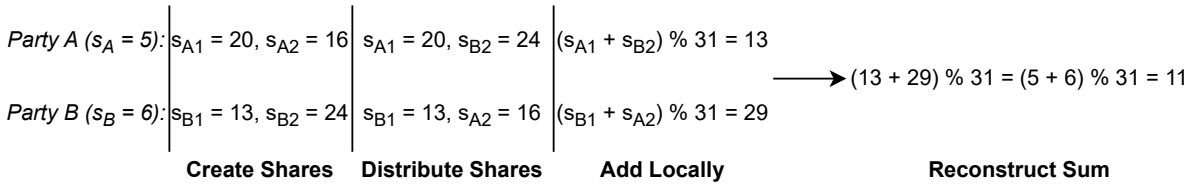


Figure 3.6: Example of computing the sum of two parties' additively secret shared variables in $\mathbb{Z}_{31}$

only for encrypting data. An outline of HE types, the primary available methods/schemes, and overall strengths/weaknesses follow.

There are three main types of HE based on the operations they allow on encrypted data and how many times they can be applied. Although the concept has been known for a long time, more powerful realizable schemes are relatively recent. The general idea of HE formalized in 1978 [36], can be further broken down into different subclasses: partial HE, which allows one of addition (typically) or multiplication unlimited times; somewhat HE, which allows performing both a limited number of times; and full HE, which allows performing both unlimited times. The first full scheme with reasonable enough computational requirements to be practical works on integers only came in 2009, courtesy of Gentry [18]. It achieves full homomorphism by bootstrapping a somewhat homomorphic scheme, which involves homomorphically decrypting the cyphertext into an equivalent one to refresh the remaining number of applicable operations.

BGV [5] and CKKS [6] are currently two of the most popular encryption schemes, and libraries like Microsoft Seal[1] implement both. BGV is a leveled scheme for exact computation on integers that functions without the expensive bootstrapping of Gentry's prior work [18]. Leveled HE lies between somewhat HE and full HE, allowing unlimited additions but having a limited budget for multiplications. BFV [15] is a scheme with similar properties. CKKS is a leveled homomorphic for approximate arithmetic on floating point numbers. It provides a more efficient and feasible choice for handling floating-point values than using a fixed-precision encoding with integer schemes. All mentioned approaches achieve computational security. The Microsoft SEAL and OpenFHE[2] libraries contain optimized implementations of the abovementioned techniques.

Underpinning its most significant advantage and downside, existing HE trades communication for local computation. The only interaction involves transferring the encrypted data to the party performing the computation and back to the one in charge of decryption. Thus, the number of messages and total data transmitted remains low. Unfortunately, although entirely local, computation on the cyphertext is many times more demanding than on the equivalent plaintext, and balancing security and performance tends to require appropriately selected parameters for the scheme. The public key nature is also a double-edged sword. Any parties can encrypt input once they generate or receive the key, but private key ownership directly determines decryption, even after falling into the wrong hands.

3.2.2. Secret Sharing Schemes

Definition 1 (Group)**.** A group [23] is a set $G(\neq \emptyset)$ and operation $\bullet : G \times G \longrightarrow G$ such that:

1. Asociativity: $a \bullet (b \bullet c) = (a \bullet b) \bullet c, \forall a, b, c \in G$
2. Neutral element: $\exists! e \in G$ such that $e \bullet a = a \bullet e = a, \forall a \in G$
3. Inverse element:$a \bullet a' = a' \bullet a = e, \forall a \in G, \exists! a'$, where e is the neutral element

Definition 2 ($\mathbb{Z}_n$)**.** $\mathbb{Z}_n$ is a group under $\{0, 1, \dots, n-1\}$ and addition modulo (%) n.

Definition 3 ($\mathbb{Z}_p^*$)**.** $\mathbb{Z}_p^*$, for p prime, is a group under $\{1, 2, \dots, n-1\}$ and multiplication modulo (%) p.

Proposition 1. *The inverse x' of $x \in \mathbb{Z}_p^*$ is $x^{p-2}\%p$.*

Secret sharing schemes are a family of methods that allow the distribution of a secret value among a group of parties by assigning each a share that does not yield any information about the secret but can,

[1] https://www.microsoft.com/en-us/research/project/microsoft-seal/
[2] https://www.openfhe.org/

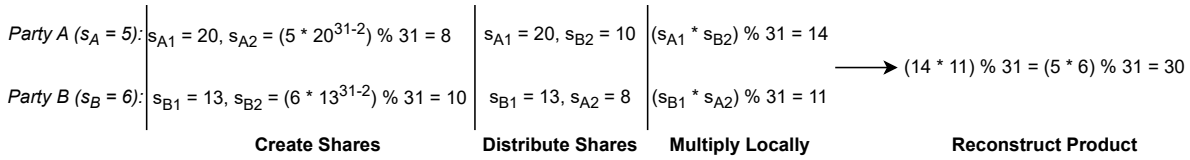


Figure 3.7: Example of computing the product of two parties' multiplication-based secret shared variables in $\mathbb{Z}_{31}^*$

when pooled with enough others, reveal the secret [10]. The following describes certain terminology surrounding such schemes and gives examples of specific approaches before diving deeper into the multiplication-based version used for `CCBNet`.

The threshold and linearity help describe specific properties of different schemes, whose building blocks are often constructs like groups or multiplication triples. Under a threshold of (t, n), where $t, n \in \mathbb{N}^*$ and $t \leq n$, at least t of the n parties must correctly bring together their shares to reconstruct the secret. A scheme is linear if reconstructing the secret entails applying linear operations on the shares. Consequently, performing linear operations on the secret happens efficiently by mirroring them on each share before reconstruction. Not leaking any information about a secret requires sampling it from a uniform distribution, which is impossible for any infinite set, like $\mathbb{Z}$ or any subset of $\mathbb{R}$. Thus, most schemes perform their computation within (derivatives of) finite integer groups (Definition 1) large enough to contain all possibly required values for computations on the secrets. The previously mentioned fixed-precision encoding is the usual method for incorporating floating-point values. Multiplication in linear schemes requires additional interaction between the parties. Usually, it involves the help of a secretly sharing an additional set of values a, b, c, called Beaver triples [3], obeying $a * b = c$, with a and b chosen arbitrarily. Protocols exist for parties to generate triples amongst themselves securely, but a trusted third party, if available, can simplify the procedure.

Additive [23] and Shamir's [40] secret sharing are two popular schemes in literature, but more recent ones like SPDZ [12] offer good security guarantees in more scenarios, alongside computation and communication efficiency improvements. Additive secret sharing is an (n, n) scheme defined on $\mathbb{Z}_n$ (Definition 2), with information-theoretic security for up to $n - 1$ passively corrupt parties. Figure 3.6 exemplifies adding two parties' secret values. Shamir's secret sharing has a configurable (t, n) threshold scheme, which produces secret shares based on a polynomial whose constant coefficient is the secret, and all others are random. It has information-theoretic security for up to $\lfloor n/2 \rfloor$ passive corrupt parties and $\lfloor n/3 \rfloor$ active ones. SPDZ is another (n, n) scheme that achieves computational security for up to $n - 1$ actively corrupt parties, requires less computation than prior similar efforts, and has spawned multiple other schemes based upon in like MASCOT [24]. All three schemes are linear and use Beaver triples to allow multiplication.

The multiplication-based scheme utilized in `CCBNet` is similar to the additive one but becomes non-linear by swapping efficient share addition for multiplication. As it works under $\mathbb{Z}_p^*$ (Definition 3), not $\mathbb{Z}_n$, instead of agreeing on a large enough n, parties agree on a large enough prime p to instantiate the group. As in the additive case, to split a secret s into k shares, parties get shares s_1 through s_{k-1} by uniformly sampling the group's set of values and fix $s_k = s \bullet (s_1 \bullet \ldots \bullet s_{k-1})'$. From Proposition 1, it follows that $s_k = s * (s_1 * \ldots * s_{k-1})^{p-2}$, where all multiplications are modulo p. Modular exponentiation can be efficiently computed even for large exponents. Reconstruction still happens by applying the group operator to all shares. Note that, although $0 \notin \mathbb{Z}_p^*$, assuming that the party holding the secret keeps one of the shares, it can set its share to 0, and sample $\mathbb{Z}_p^*$ for the remaining ones. Figure 3.7 gives a small example of multiplying two parties' secret values.

4

Additional Experiments

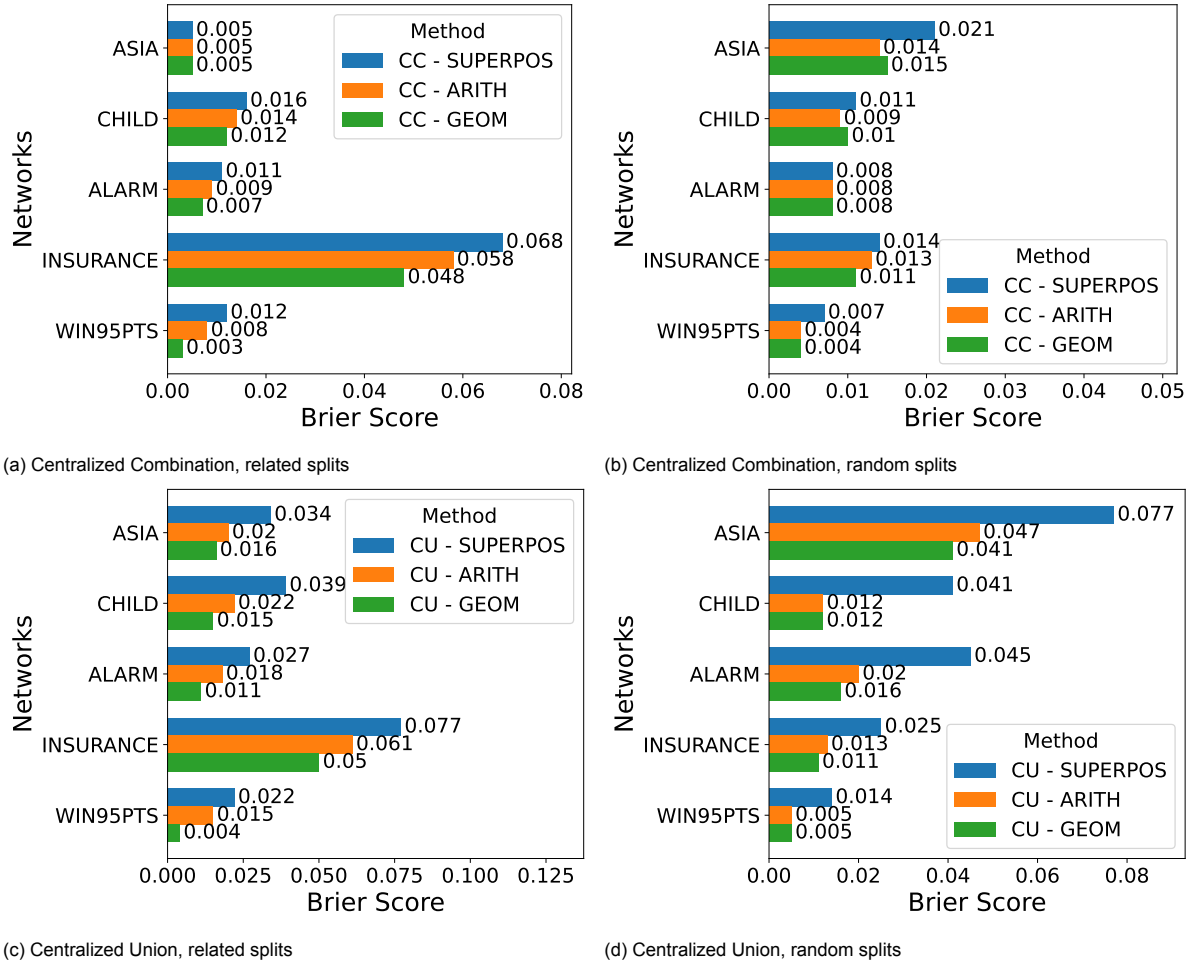
The quality and speed of analysis in a BN and related models can be affected by various (implementation) choices within the inference itself and, leading up to it, in the steps for preparing the model. The following subsections examine a few such choices, aiming to quantify them: the operator for merging probability values during the combination of networks, the heuristic for ordering in variable elimination, and the possibility of hardware-accelerating inference. A representative slice of the results is presented as figures for all examined criteria, while the accompanying tables contain the complete results. Employed metrics are also the same. Moreover, all performed experiments involve four parties to contain the number of runs while retaining helpful insight. As a reminder, the two considered cases for splitting reference network variables amongst parties are: related, ensuring the connectedness of variables assigned to a party in the original network, and random, only ensuring equally sized splits.

4.1. Probability Function Merge Operator

The three candidates chosen to compare the effect on prediction quality of different operators for merging probability function values are the superposition of Feng [16], the arithmetic mean, and the geometric mean used in `CCBNet`. The operator choice does not impact the structure of a combined network but gives different parameters for variables contained in more than one member. To recap, the superposition ($\oplus$) of x and y is $x + y - x * y$, while the arithmetic mean is $\frac{x+y}{2}$, and the geometric mean is $\sqrt[2]{x * y}$. Notably, with normalized input CPDs, the arithmetic mean output CPD does not require renormalization, which is not the case for the other two operators. Additionally, unlike the superposition, both means can naturally incorporate weighting and apply to more than two inputs. The performed experiments cover the two centralized baselines from the paper, only swapping the different operators: Centralized Combination (CC), which combines the structure of networks two at a time via Feng's ruleset [16], ordinarily using the superposition, and Centralized Union (CU), which takes the union of all networks' structures, employing the geometric mean in prior experiments. `CCBNet` relies on multiplication-based parameter combination and inference products for its secret sharing scheme to function as intended, so its probability merging operator can not be swapped out like for the central methods. However, given its equivalence to the default geometric mean CU, those results still show the effects of different operators on hypothetical framework variants. The reference networks, number of variables in multiple

Vars in >1 Party		10%						50%				
Method		CC			CU			CC			CU	
Operator		Superpos	Arith	Geo	Superpos	Arith	Geo	Superpos	Arith	Geo	Superpos	Arith	Geo
Network	ASIA	0.005	0.005	0.005	0.034	0.02	0.016	0.005	0.005	0.005	0.019	0.013	0.011
	CHILD	0.104	0.104	0.104	0.108	0.103	0.104	0.014	0.014	0.014	0.035	0.018	0.01
	ALARM	0.036	0.031	0.024	0.053	0.039	0.025	0.002	0.002	0.002	0.015	0.008	0.004
	INSURANCE	0.106	0.104	0.104	0.109	0.104	0.104	0.06	0.053	0.046	0.075	0.056	0.047
	WIN95PTS	0.003	0.003	0.003	0.008	0.006	0.004	0.013	0.01	0.007	0.03	0.019	0.005

Table 4.1: Brier scores for different probability function merge operators and model fusion strategies - related splits, 4 parties, 60% of vars evidence in queries (lower is better)



(a) Centralized Combination, related splits

(b) Centralized Combination, random splits

(c) Centralized Union, related splits

(d) Centralized Union, random splits

Figure 4.1: Brier scores for different probability function merge operators and model fusion strategies, 4 parties, 30% of vars in >1 party, 60% of vars evidence in queries (lower is better)

parties, and amount of evidence per query are unchanged from the main paper experiments.

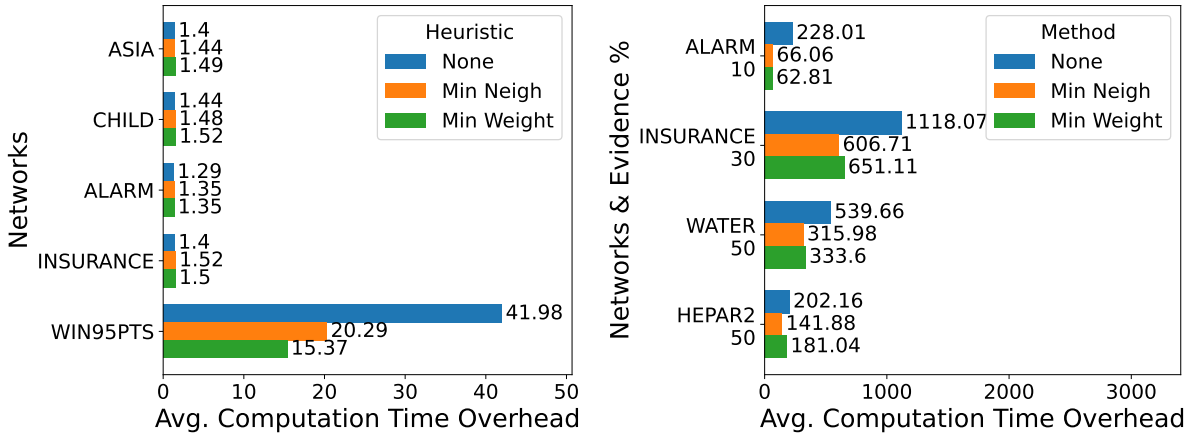
Regardless of the variable splitting method, the geometric mean performs best overall, while the arithmetic one comes second and the superposition third across both CC and CU. Figure 4.1a and Figure 4.1b show the results for CC under related and random splits, respectively. The geometric mean fares worse than the arithmetic one only slightly in two cases, specifically under random splits, while the superposition never outperforms any other operator. Figure 4.1c and Figure 4.1d give the results for CU under the abovementioned splits, showcasing the same trends even more strongly, with the geometric mean never performing worse than the alternatives. The difference between operators is more pronounced in CU than CC for both splits. The predictive performance for both techniques is higher across the board with related splits. In contrast, there is a more significant relative difference between the operators in the more challenging random splits. The full related (Table 4.1) and random (Table 4.2) results with different numbers of variables in multiple parties reconfirm the same patterns.

4.2. Variable Elimination Ordering Heuristic

The strategies selected to explore the effect of VE ordering on inference computation overhead are a baseline unoptimized approach, which eliminates variables in the order they appear within given factors, alongside the two previously described ordering heuristics, MinNeighbors (Algorithm 2), and MinWeight (Algorithm 3). Naturally, the baseline no heuristic approach is the most lightweight, running in constant time, while the other two need to iterate through all factors and variables within them. MinNeighbors is the less intensive heuristic, as it merely has to count the number of factors in which each variable appears. In contrast, MinWeight must keep track of the variables forming each possible

Vars in >1 Party		10%						50%					
Method		CC			CU			CC			CU		
Operator		Superpos	Arith	Geo	Superpos	Arith	Geo	Superpos	Arith	Geo	Superpos	Arith	Geo
Network	ASIA	0.005	0.005	0.005	0.006	0.005	0.005	0.005	0.004	0.003	0.048	0.023	0.019
	CHILD	0.04	0.034	0.035	0.049	0.036	0.037	0.013	0.009	0.009	0.053	0.016	0.016
	ALARM	0.037	0.035	0.033	0.057	0.04	0.037	0.004	0.003	0.002	0.03	0.01	0.008
	INSURANCE	0.03	0.022	0.019	0.035	0.017	0.013	0.02	0.017	0.015	0.044	0.021	0.017
	WIN95PTS	0.019	0.016	0.016	0.029	0.019	0.018	0.011	0.009	0.008	0.018	0.008	0.007

Table 4.2: Brier scores for different probability function merge operators and model fusion strategies - random splits, 4 parties, 60% of vars evidence in queries (lower is better)



(a) 30% of vars in >1 party, 60% of vars evidence in queries

(b) 50% of vars in >1 party, varying % of vars evidence in queries

Figure 4.2: Average computation time overhead relative to original network, 4 parties, related splits (lower is better)

new factor and calculate their size accordingly, implicitly optimizing for reduced memory usage. With the expectation that related splits would more closely represent most real-world applications than random ones, conducted experiments forgo the latter to explore more scenarios involving the former. Some tests involve two new reference networks not used in the paper, one medium and one large in terms of node count, namely WATER (32 nodes, 66 edges, 10083 parameters) and HEPAR2 (70 nodes, 123 edges, 1453 parameters). Since the goal is to measure the differences between choices in the framework, experiments use `CCBNet` directly. As in all other assessments, inference queries in the reference networks always utilize the same MinWeight heuristic for a fair comparison.

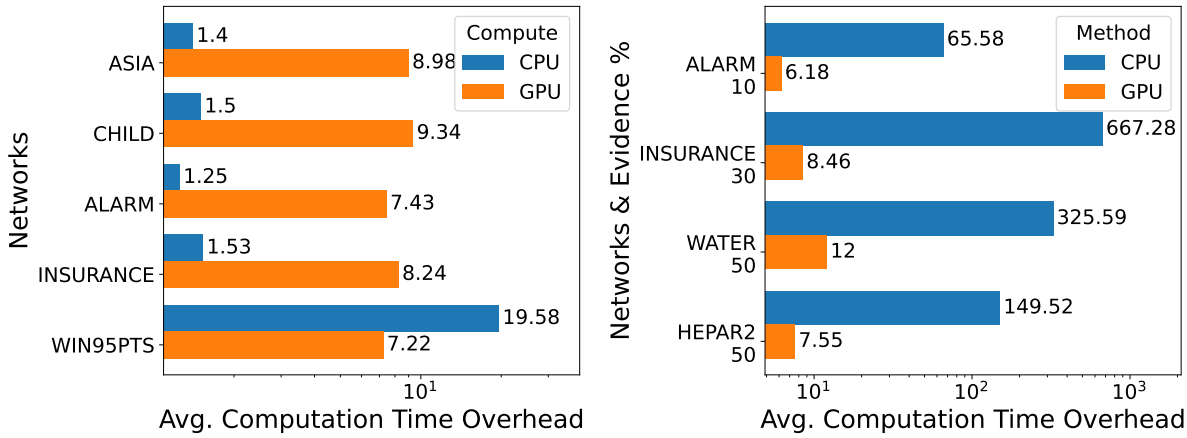
In the usual scenario from Figure 4.2a, with 30% of variables in more than one party and query evidence supplied for 60% of variables, the approaches perform similarly in all but the largest network, where the heuristics are much faster. The heuristic-less approach is consistently the fastest in all small and medium networks, albeit by a minimal margin. At the same time, MinNeighbors is marginally better than or equal to MinWeight in all but the largest medium network. On the large network, however, not

Vars in >1 Party		10%			30%			50%		
Heuristic		None	Min Neigh	Min Weight	None	Min Neigh	Min Weight	None	Min Neigh	Min Weight
Network	ASIA	1.6x (0.79s)	1.45x (0.72s)	1.46x (0.72s)	1.4x (0.69s)	1.44x (0.72s)	1.49x (0.74s)	1.41x (0.81s)	1.43x (0.82s)	1.45x (0.83s)
	CHILD	1.24x (1.53s)	1.17x (1.45s)	1.2x (1.49s)	1.44x (1.82s)	1.48x (1.87s)	1.52x (1.92s)	1.45x (1.97s)	1.49x (2.03s)	1.52x (2.08s)
	ALARM	1.08x (2.67s)	1.1x (2.72s)	1.12x (2.78s)	1.29x (3.11s)	1.35x (3.28s)	1.35x (3.27s)	1.55x (3.69s)	1.47x (3.5s)	1.54x (3.66s)
	INSURANCE	1x (1.85s)	1.04x (1.93s)	1.07x (1.98s)	1.4x (2.46s)	1.52x (2.67s)	1.5x (2.62s)	2.87x (5.08s)	2.74x (4.86s)	2.85x (5.05s)
	WIN95PTS	0.83x (5.09s)	0.88x (5.4s)	0.93x (5.67s)	41.98x (256.65s)	20.29x (124.21s)	15.37x (94.08s)	236.68x (1460.5s)	140.61x (867.66s)	144.87x (893.96s)

Table 4.3: Average computation time overhead relative to original network & total compute time for different variable elimination ordering heuristics - related splits, 4 parties, 60% of vars evidence in queries (lower is better)

Network, Evidence %	Heuristic	None	Min Neigh	Min Weight
ALARM, 10		228.01x (1201.22s)	66.06x (348.02s)	62.81x (330.89s)
INSURANCE, 30		1118.07x (3310.74s)	606.71x (1796.55s)	651.11x (1928.03s)
WATER, 50		539.66x (1480.25s)	315.98x (866.7s)	333.6x (915.03s)
HEPAR2, 50		202.16x (1489.3s)	141.88x (1045.27s)	181.04x (1333.75s)

Table 4.4: Average computation time overhead relative to original network & total compute time for different variable elimination ordering heuristics - related splits, 4 parties, 50% of vars in >1 party, varying % of vars evidence in queries (lower is better)



(a) 30% of vars in >1 party, 60% of vars evidence in queries

(b) 50% of vars in >1 party, varying % of vars evidence in queries

Figure 4.3: Average computation time overhead relative to original network, 4 parties, related splits (log scale, lower is better)

involving a heuristic is an order of magnitude slower, while the more complex and expensive MinWeight is ≈25% faster than MinNeighbors. The complete results of Table 4.3, including absolute figures for the total computation time and different numbers of variables present in multiple parties, yield similar conclusions, noting that, for the large network, the differences between approaches only start showing up when the overlap in party variables increases.

Table 4.4's further tests with an overlap in party variables increased to 50%, and lower levels of query evidence verify that in more complex scenarios, the time spent by heuristics searching for a better variable more than pays off. These tests on the more complex medium networks from the previous setting and the two new ones mentioned previously lower the evidence used in queries to create challenging enough scenarios. Thus, the evidence reduction for simpler networks is more drastic than for more intricate ones. The heuristics are always faster, by around an order of magnitude or more in most cases one case. MinNeighbors is only bested by MinWeight in one case, by a small amount, while in the others, it performs better, although also by a modest amount, except for the last one, where its advantage expands. Table 4.4 also contains the absolute computation time values for the tests. All in all, heuristics for elimination ordering provide a net benefit also in `CCBNet`, but increasing their complexity does not necessarily result in a speedup across all workloads.

4.3. GPU-Accelerated Inference

Knowing that inference computation cost explodes as network and query complexity increase and that the values of discrete factors take the form of an array makes it worthwhile to investigate whether introducing accelerator hardware like a GPU can lessen variable elimination computation time. As the implementation of `CCBNet` uses the pgmpy[1] BN Python package, which depends on arrays from

[1] https://pgmpy.org/

Vars in >1 Party		10%		30%		50%	
Compute		CPU	GPU	CPU	GPU	CPU	GPU
Network	ASIA	1.64x (0.82s)	9.24 (4.59s)	1.4x (0.72s)	8.98x (4.61s)	1.41x (0.82s)	8.95x (5.17s)
	CHILD	1.15x (1.43s)	7.36x (9.17s)	1.5x (1.88s)	9.34x (11.7s)	1.63x (2.07s)	10.08x (12.75s)
	ALARM	1.13x (2.71s)	6.99x (16.74s)	1.25x (3.2s)	7.43x (18.98s)	1.48x (3.55s)	8.49x (20.3s)
	INSURANCE	1.1x (1.96s)	6.94x (12.33s)	1.53x (2.73s)	8.24x (14.7s)	2.81x (4.96s)	10.82x (19.08s)
	WIN95PTS	0.88x (5.5s)	5.34x (33.23s)	19.58x (123.83s)	7.22x (45.69s)	134.52x (828.03s)	8.01x (49.3s)

Table 4.5: Average computation time overhead relative to original network & total compute time for CPU-only and GPU-aided execution - related splits, 4 parties, 60% of vars evidence in queries (lower is better)

Method		CPU	GPU
Network, Evidence %	ALARM, 10	65.58x (314.68s)	6.18x (29.65s)
	INSURANCE, 30	667.28x (1909.09s)	8.46x (24.2s)
	WATER, 50	325.59x (877.8s)	12x (32.34s)
	HEPAR2, 50	149.52x (1048.77s)	7.55x (52.93s)

Table 4.6: Average computation time overhead relative to original network & total compute time for CPU-only and GPU-accelerated execution - related splits, 4 parties, 50% of vars in >1 party, varying % of vars evidence in queries (lower is better)

the NumPy[2] package for factor values, the CuPy[3] package is a fitting choice to GPU-accelerate the array-based part of the computation, thanks to its NumPy-compatible interface. Apart from attempting to limit undue information transfer between the two devices, the trialed GPU-accelerated version is an unoptimized implementation, identical to the CPU-based one in terms of the variable elimination code, that only makes the minimum required changes to pgmpy's class for factors to allow choosing between working with the probability values in CPU and GPU. Additionally, only the manipulation of factor values itself is switchable. Other logic related to computation with the factor, like determining the set of variables in the resulting product between factors, still always happens on the CPU. The utilized GPU is a Tesla V100-SXM2-32GB, alongside version 12.1.0 of cupy-cuda11x. Both devices store values at full (64-bit) precision. On a related note, GPUs tend to have access to considerably less memory than CPUs, which can be problematic when working with expansive, densely connected networks. Reducing the precision of stored numbers is a common trick to maximize available GPU memory, but not all workloads can tolerate it to the same extent. Because of the multiplication-based secret sharing, `CCBNet` is also sensitive, so precision must be carefully decreased based on the task. The examined scenarios are the same as the previous comparison of VE ordering strategies.

The usual setting with fixed 30% overlaps in party variables and 60% evidence in Figure 4.3a shows the GPU as many times slower than the CPU for all networks but the large one, where it is twice as fast. The communication overhead between the two devices dominates the computation time in more straightforward cases, corroborated by GPU utilization being uneven and relatively low. Although lowering it further should be possible, a non-trivial amount of overhead always exists so long as the two devices communicate over a standard interface instead of being tightly integrated into the same package. Covering additional numbers of overlaps in party variables, the unabridged figures in Table 4.5 reaffirm the results and report absolute total computation time alongside relative. Nevertheless, the speedup in the largest network hints that, similarly to VE ordering heuristics, the advantage only unveils itself in more complicated situations.

[2]https://numpy.org/
[3]https://cupy.dev/

The more involved Figure 4.3b tests with 50% party variable overlaps and varying, lower query evidence show, as before, the expected benefits, amounting to speedups of 10-78x. Thus, despite the room for device-specific improvements, the GPU implementation achieves over 34x more performance on average than the CPU counterpart. Table 4.6, which also contains the absolute computation time for the results, reveals that the GPU implementation finishes all tests in under one minute. At the same time, on the CPU, they all require many (tens of) minutes. Thus, acceleration seems unlikely to provide many advantages in more mundane scenarios, where thousands of queries can run in, at most, a couple of minutes. However, in demanding environments, the improvement is massive. Choosing which device to assign queries based on prior knowledge could provide the best of both worlds.

5

Conclusion

With the learnings from previous chapters, some conclusions for the initial research questions follow:

1 Can predictive performance comparable with a classic centralized combination be achieved in such a collaborative, confidential environment?

Yes, `CCBNet` achieves performance equal to its mimicked centralized union variant and similar to other centralized baselines while protecting confidential party knowledge by having parties collaboratively perform analysis on an augmented version of their local model instead of a global one. Its predictive performance remains similar to the tested non-union centralized baseline, even when sometimes being outperformed in small networks. In larger networks, it and the corresponding centralized union tend to outperform other methods significantly. Although a straight union over input network structures produces denser outputs with potentially superfluous edges, it avoids the risk of discarding essential connections and exchanging extra statistics likely to leak private party knowledge. Similarly, for parameters, merging corresponding party probability values via the geometric mean yields good interpolation while allowing for a secret sharing scheme where reconstruction happens at inference.

2 What would be the overhead of a solution exhibiting the desired properties?

Considering the implemented version of the proposed `CCBNet` framework, inference computation overhead is negatively correlated with party count but positively correlated with the overlap in their knowledge and network size, ranging from many times slower to faster than the central case, while communication size increases with all. The communication count is linear in the party count. Overlap in party knowledge contributes most to computation and communication increases. Still, the exponential scaling of probabilistic inference means that spreading the computation over enough parties can improve its efficiency for large networks. Computation and communication needed to prepare the system for inference by updating party models also increase with the three abovementioned criteria. However, the process typically only happens seldom with decreased cost after the initial execution.

3 What are some of the choices influencing probabilistic inference prediction quality and overhead?

All examined choices, probability function merge operator, variable elimination ordering heuristic, and GPU acceleration significantly influence inference performance in various scenarios. The geometric mean provides better predictions than other operators almost always. Both tested variable ordering heuristics yield sizeable speedups in more demanding workloads over the baseline. The overhead of working with a GPU brings slowdowns to simpler settings but has enormous benefits in complex ones.

Many future work avenues exist, ranging from efficiency improvements to enhanced functionality. Two important ones also mentioned in the paper are improving the quality of model mergers, possibly by utilizing any extra vertically partitioned data instances when available and reducing communication, especially for inference message size. Furthermore, allowing joint variable discretization as previously outlined or even directly supporting continuous variables would be helpful extensions. Another already-mentioned improvement is caching intermediate factors between inference calls to speed up similar queries. Finally, other possible enhancements are supporting approximate inference and relaxing the assumption about variables being equivalent only when their names match exactly.

Bibliography

[1] Amin Abyaneh et al. *FED-CD: Federated Causal Discovery from Interventional and Observational Data*. 2022. DOI: 10.48550/ARXIV.2211.03846. URL: https://arxiv.org/abs/2211.03846.

[2] Dalal Alrajeh, Hana Chockler, and Joseph Y. Halpern. "Combining experts' causal judgments". In: *Artificial Intelligence* 288 (2020), p. 103355. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2020.103355. URL: https://www.sciencedirect.com/science/article/pii/S0004370220301065.

[3] Donald Beaver. "Efficient Multiparty Protocols Using Circuit Randomization". In: *Advances in Cryptology — CRYPTO '91*. Ed. by Joan Feigenbaum. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 420–432. ISBN: 978-3-540-46766-3.

[4] Tomas Beuzen, Lucy Marshall, and Kristen D. Splinter. "A comparison of methods for discretizing continuous variables in Bayesian Networks". In: *Environmental Modelling & Software* 108 (2018), pp. 61–66. ISSN: 1364-8152. DOI: https://doi.org/10.1016/j.envsoft.2018.07.007. URL: https://www.sciencedirect.com/science/article/pii/S1364815217313269.

[5] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) Fully Homomorphic Encryption without Bootstrapping". In: *ACM Trans. Comput. Theory* 6.3 (July 2014). ISSN: 1942-3454. DOI: 10.1145/2633600. URL: https://doi.org/10.1145/2633600.

[6] Jung Hee Cheon et al. "Homomorphic Encryption for Arithmetic of Approximate Numbers". In: *Advances in Cryptology – ASIACRYPT 2017*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Cham: Springer International Publishing, 2017, pp. 409–437. ISBN: 978-3-319-70694-8.

[7] David Maxwell Chickering. "Learning Bayesian Networks is NP-Complete". In: *Learning from Data: Artificial Intelligence and Statistics V*. Learning from Data: Artificial Intelligence and Statistics V. Springer-Verlag, 1996, pp. 121–130. URL: https://www.microsoft.com/en-us/research/publication/learning-bayesian-networks-is-np-complete/.

[8] David Maxwell Chickering, Christopher Meek, and David Heckerman. *Large-Sample Learning of Bayesian Networks is NP-Hard*. 2012. DOI: 10.48550/ARXIV.1212.2468. URL: https://arxiv.org/abs/1212.2468.

[9] Diego Colombo, Marloes H Maathuis, et al. "Order-independent constraint-based causal structure learning." In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 3741–3782.

[10] Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015. DOI: 10.1017/CBO9781107337756.

[11] Florian van Daalen et al. *VertiBayes: Learning Bayesian network parameters from vertically partitioned data with missing values*. 2022. DOI: 10.48550/ARXIV.2210.17228. URL: https://arxiv.org/abs/2210.17228.

[12] Ivan Damgård et al. "Multiparty Computation from Somewhat Homomorphic Encryption". In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 643–662. ISBN: 978-3-642-32009-5.

[13] José Del Sagrado and Serafín Moral. "Qualitative combination of Bayesian networks". In: *International Journal of Intelligent Systems* 18.2 (2003), pp. 237–249. DOI: https://doi.org/10.1002/int.10086. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/int.10086. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/int.10086.

[14] Daniel Escudero. *An Introduction to Secret-Sharing-Based Secure Multiparty Computation*. Cryptology ePrint Archive, Paper 2022/062. https://eprint.iacr.org/2022/062. 2022. URL: https://eprint.iacr.org/2022/062.

[15] Junfeng Fan and Frederik Vercauteren. “Somewhat practical fully homomorphic encryption”. In: *Cryptology ePrint Archive* (2012).

[16] Guang Feng, Jia-Dong Zhang, and Stephen Shaoyi Liao. “A novel method for combining Bayesian networks, theoretical analysis, and its applications”. In: *Pattern Recognition* 47.5 (2014), pp. 2057–2069. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2013.12.005. URL: https://www.sciencedirect.com/science/article/pii/S0031320313005232.

[17] Erdun Gao et al. *FedDAG: Federated DAG Structure Learning*. 2021. DOI: 10.48550/ARXIV.2112.03555. URL: https://arxiv.org/abs/2112.03555.

[18] Craig Gentry. “Fully Homomorphic Encryption Using Ideal Lattices”. In: *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*. STOC ’09. Bethesda, MD, USA: Association for Computing Machinery, 2009, pp. 169–178. ISBN: 9781605585062. DOI: 10.1145/1536414.1536440. URL: https://doi.org/10.1145/1536414.1536440.

[19] Oded Goldreich. “Foundations of Cryptography – A Primer”. In: *Foundations and Trends® in Theoretical Computer Science* 1.1 (2005), pp. 1–116. ISSN: 1551-305X. DOI: 10.1561/0400000001. URL: http://dx.doi.org/10.1561/0400000001.

[20] Seyedmohsen Hosseini, Abdullah Al Khaled, and MD Sarder. “A general framework for assessing system resilience using Bayesian networks: A case study of sulfuric acid manufacturer”. In: *Journal of Manufacturing Systems* 41 (2016), pp. 211–227. ISSN: 0278-6125. DOI: https://doi.org/10.1016/j.jmsy.2016.09.006. URL: https://www.sciencedirect.com/science/article/pii/S0278612516300632.

[21] Jianli Huang et al. *Towards Privacy-Aware Causal Structure Learning in Federated Setting*. 2022. DOI: 10.48550/ARXIV.2211.06919. URL: https://arxiv.org/abs/2211.06919.

[22] B. Jones et al. “The use of Bayesian network modelling for maintenance planning in a manufacturing industry”. In: *Reliability Engineering & System Safety* 95.3 (2010), pp. 267–277. ISSN: 0951-8320. DOI: https://doi.org/10.1016/j.ress.2009.10.007. URL: https://www.sciencedirect.com/science/article/pii/S0951832009002518.

[23] Daniel Kales. *Secret Sharing*. URL: https://www.iaik.tugraz.at/wp-content/uploads/teaching/mfc/secret_sharing.pdf.

[24] Marcel Keller, Emmanuela Orsini, and Peter Scholl. “MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’16. Vienna, Austria: Association for Computing Machinery, 2016, pp. 830–842. ISBN: 9781450341394. DOI: 10.1145/2976749.2978357. URL: https://doi.org/10.1145/2976749.2978357.

[25] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[26] Stan Z Li. *Markov random field modeling in image analysis*. Springer Science & Business Media, 2009.

[27] Tian Li et al. “Federated Learning: Challenges, Methods, and Future Directions”. In: *IEEE Signal Processing Magazine* 37.3 (2020), pp. 50–60. DOI: 10.1109/MSP.2020.2975749.

[28] Yang Liu et al. *Vertical Federated Learning*. 2022. arXiv: 2211.12814 [cs.LG].

[29] Dimitris Margaritis et al. “Learning Bayesian network model structure from data”. PhD thesis. School of Computer Science, Carnegie Mellon University Pittsburgh, PA, USA, 2003.

[30] Ken McNaught and Andy Chan. “Bayesian networks in manufacturing”. In: *Journal of Manufacturing Technology Management* 22.6 (Jan. 2011). Ed. by Khairy A.H. Kobbacy and Sunil Vadera, pp. 734–747. ISSN: 1741-038X. DOI: 10.1108/17410381111149611. URL: https://doi.org/10.1108/17410381111149611.

[31] Radosław Miśkiewicz and Radosław Wolniak. “Practical Application of the Industry 4.0 Concept in a Steel Company”. In: *Sustainability* 12.14 (2020). ISSN: 2071-1050. DOI: 10.3390/su12145776. URL: https://www.mdpi.com/2071-1050/12/14/5776.

[32] Saideep Nannapaneni, Sankaran Mahadevan, and Sudarsan Rachuri. “Performance evaluation of a manufacturing process under uncertainty using Bayesian networks”. In: *Journal of Cleaner Production* 113 (2016), pp. 947–959. ISSN: 0959-6526. DOI: https://doi.org/10.1016/j.jclepro.2015.12.003. URL: https://www.sciencedirect.com/science/article/pii/S0959652615018144.

[33] Ignavier Ng and Kun Zhang. *Towards Federated Bayesian Network Structure Learning with Continuous Optimization*. 2021. DOI: 10.48550/ARXIV.2110.09356. URL: https://arxiv.org/abs/2110.09356.

[34] Farnaz Nojavan A., Song S. Qian, and Craig A. Stow. “Comparative analysis of discretization methods in Bayesian networks”. In: *Environmental Modelling & Software* 87 (2017), pp. 64–71. ISSN: 1364-8152. DOI: https://doi.org/10.1016/j.envsoft.2016.10.007. URL: https://www.sciencedirect.com/science/article/pii/S1364815216308672.

[35] Judea Pearl. *Causality*. 2nd ed. Cambridge University Press, 2009. DOI: 10.1017/CBO9780511803161.

[36] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. “On data banks and privacy homomorphisms”. In: *Foundations of secure computation* 4.11 (1978), pp. 169–180.

[37] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[38] Marco Scutari and Jean-Baptiste Denis. *Bayesian networks: with examples in R*. CRC press, 2021.

[39] Marco Scutari, Catharina Elisabeth Graafland, and José Manuel Gutiérrez. “Who Learns Better Bayesian Network Structures: Constraint-Based, Score-based or Hybrid Algorithms?” In: *Proceedings of the Ninth International Conference on Probabilistic Graphical Models*. Ed. by Václav Kratochvíl and Milan Studený. Vol. 72. Proceedings of Machine Learning Research. PMLR, Sept. 2018, pp. 416–427. URL: https://proceedings.mlr.press/v72/scutari18a.html.

[40] Adi Shamir. “How to Share a Secret”. In: *Commun. ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176. URL: https://doi.org/10.1145/359168.359176.

[41] Peter Spirtes, Clark N Glymour, and Richard Scheines. *Causation, prediction, and search*. MIT press, 2000.

[42] Todd Andrew Stephenson. *An introduction to Bayesian network theory and usage*. Tech. rep. Idiap, 2000.

[43] Vahid Rezaei Tabar and Fatemeh Elahi. “A Novel Method for Aggregation of Bayesian Networks without Considering an Ancestral Ordering”. In: *Applied Artificial Intelligence* 32.2 (2018), pp. 214–227. DOI: 10.1080/08839514.2018.1451134. eprint: https://doi.org/10.1080/08839514.2018.1451134. URL: https://doi.org/10.1080/08839514.2018.1451134.

[44] Roberto Tedesco et al. “Distributed Bayesian Networks for User Modeling”. In: *Proceedings of E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2006*. Ed. by Thomas Reeves and Shirley Yamashita. Honolulu, Hawaii, USA: Association for the Advancement of Computing in Education (AACE), Oct. 2006, pp. 292–299. URL: https://www.learntechlib.org/p/23699.

[45] Ioannis Tsamardinos, Laura Brown, and Constantin Aliferis. “The Max-Min Hill-Climbing Bayesian Network Structure Learning Algorithm”. In: *Machine Learning* 65 (Oct. 2006), pp. 31–78. DOI: 10.1007/s10994-006-6889-7.

[46] Ioannis Tsamardinos et al. “Algorithms for large scale Markov blanket discovery.” In: *FLAIRS conference*. Vol. 2. St. Augustine, FL. 2003, pp. 376–380.

[47] Marina Velikova et al. “Assisted Diagnostics Methodology for Complex High-Tech Applications”. In: *2019 4th International Conference on System Reliability and Safety (ICSRS)*. 2019, pp. 457–463. DOI: 10.1109/ICSRS48664.2019.8987704.

[48] TS Verma and Judea Pearl. “Equivalence and Synthesis of Causal Models”. In: *Probabilistic and Causal Inference: The Works of Judea Pearl*. 1st ed. New York, NY, USA: Association for Computing Machinery, 2022, pp. 221–236. ISBN: 9781450395861. URL: https://doi.org/10.1145/3501714.3501732.

[49] Wim Wiegerinck, Bert Kappen, and Willem Burgers. “Bayesian Networks for Expert Systems: Theory and Practical Applications”. In: *Interactive Collaborative Information Systems*. Ed. by Robert Babuška and Frans C. A. Groen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 547–578. DOI: 10.1007/978-3-642-11688-9_20. URL: https://doi.org/10.1007/978-3-642-11688-9_20.

[50] Eric Wolbrecht et al. “Monitoring and diagnosis of a multistage manufacturing process using Bayesian networks”. In: *AI EDAM* 14.1 (2000), pp. 53–67. DOI: 10.1017/S0890060400141058.

[51] Wei-Ting Yang. “An Integrated Physics-Informed Process Control Framework and Its Applications to Semiconductor Manufacturing”. Theses. Université de Lyon, Jan. 2020. URL: https://theses.hal.science/tel-03461289.

[52] Xun Yi, Russell Paulet, and Elisa Bertino. “Homomorphic Encryption”. In: *Homomorphic Encryption and Applications*. Cham: Springer International Publishing, 2014, pp. 27–46. ISBN: 978-3-319-12229-8. DOI: 10.1007/978-3-319-12229-8_2. URL: https://doi.org/10.1007/978-3-319-12229-8_2.

[53] Chen Zhang et al. “A survey on federated learning”. In: *Knowledge-Based Systems* 216 (2021), p. 106775. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2021.106775. URL: https://www.sciencedirect.com/science/article/pii/S0950705121000381.