Fast Fourier Transform Reductions for Bayesian Network Inference

Vincent Hsiao
University of Maryland,
College Park

Dana Nau
University of Maryland,
College Park

Rina Dechter
University of California,
Irivine

Abstract

Bayesian Networks are useful for analyzing the properties of systems with large populations of interacting agents (e.g., in social modeling applications and distributed service applications). These networks typically have large functions (CPTs), making exact inference intractable. However, often these models have additive symmetry. In this paper we show how summation-based CPTs, especially in the presence of symmetry, can be computed efficiently through the usage of the Fast Fourier Transform (FFT).

In particular, we propose an efficient method using the FFT for reducing the size of Conditional Probability Tables (CPTs) in Bayesian Networks with summation-based *causal independence* (CI). We show how to apply it directly towards the acceleration of Bucket Elimination, and we subsequently provide experimental results demonstrating the computational speedup provided by our method.

1 INTRODUCTION

There is increasing interest in analyzing the properties of systems with large populations of interacting agents. Examples include social modeling using Evolutionary Game Theory (EGT) models, and distributed service applications such as Function-as-a-Service (FaaS), the Smart Grid, and autonomous drone delivery.

Bayesian Networks are useful for analyzing the properties of such distributed systems for tasks such as load balancing (Bassamzadeh and Ghanem, 2017), reliability analysis (Jiang et al., 2012), or fault diagnosis (Cai et al., 2014). These networks typically have large functions (CPTs), making even the specification of BN intractable and clearly exact inference intractable. It is well known that Bayesian-Network inference is exponential in the network's induced width (or tree-width), making exact inference intractable especially over large CPT's, as they lead to very high induced widths (Dechter, 2013). However, a subset of these distributed service models such as k-out-of-n reliability models include large summations of a distributed resource (e.g. number of computational hosts, unit of power available per generator in a Smart Grid) which have stochastic availability. The unique additive symmetry present in these models make them amenable to efficient inference and CPT reduction algorithms through the usage of the Fast Fourier Transform (FFT) (Cooley and Tukey, 1965).

In this paper we make the following contributions:

1. We propose an efficient method using the FFT for reducing the size of Conditional Probability Tables (CPTs) in Bayesian Networks with summation-type *causal independence* (CI) [to be defined later].
2. We show how to apply this reduction directly towards the acceleration of Bucket Elimination.
3. We empirically demonstrate the computational speedup our method provides over the naive inference approach and existing temporal based decomposition approaches.

We will conclude this introductory section with a few motivating domains. Next, in (Section 2), we provide background on causal independence (CI) in Bayesian networks, and on computing sum of random variables using Fast Fourier Transform (FFT). We then show how FFT can be used for efficient reduction of a BN with large CI functions (section 3.1) and subsequently (section 3.2) show how FFT can speed up general probabilistic inference (e.g., Bucket elimination) when the network has summation-based CI fragments. Section 4 provides empirical evaluation and section 5 concludes.

Proceedings of the 25$^{\text{th}}$ International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151. Copyright 2022 by the author(s).

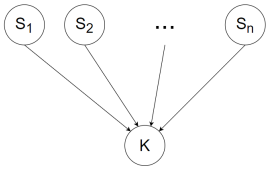

Figure 1: Bayesian Network for summation of random variables, $K = \sum_i S_i$

1.1 Distributed Resources

Distributed resource problems can be modeled using k-out-of-n Bayesian Networks (Bibartiu et al., 2019). The general idea is to model n different resource providers $\{S_1, \ldots, S_n\}$ which each provide some varying amount of distributed resource (server nodes, power generation, etc.) with stochastic availability and/or quantity. For example, in a distributed computation application, each server location can be modeled as a separate node S_i which provides some k_i amount of computing resource (e.g. S_1 is some server cluster with 300 total available computing nodes). As mentioned in Bibartiu et al. (2019), the naive k-out-of-n model is one converging node K with n parents $\{S_1, \ldots, S_n\}$ with the structure in Fig. 1. The value of the K node is the sum of the variables $\{S_i, \ldots, S_n\}$. The CPT for node K is very sparse as a table and can be represented symbolically as the function:

$$P(K = k|S_1 = k_1, \ldots, S_N = k_n) = \mathbb{1}_{k=\sum_i k_i} \quad (1)$$

Similar and more complex models can be constructed for the evaluation of other distributed resource problems where there are multiple providers and the goal is to check whether the sum of resources sourced from individual providers satisfies a given constraint.

This type of convergent structure can also be found in Bayesian Networks used to represent Fault Trees (Bibartiu et al., 2019). Fault Trees are frequently used to analyze system reliability and a key component in many of them are k-out-of-n voting gates (Portinale and Bobbio, 2013), which only activate if more than k out of n parents encounter a fault event. In the distributed resource model, each S_i's domain is simply $\{0, 1\}$ and we query the probability that the summation node K has a value greater or equal to k.

1.2 Evolutionary Game Theory

Bayesian networks have also been used in modeling evolutionary games (Hsiao et al., 2021). In an evolutionary game, the fitness of an agent is evaluated as the sum of payoff values obtained by playing normal form games with neighboring agents $N(i)$:

$$Pay(s_i) = \sum_{j \in N(i)} U[s_i, s_j] \quad (2)$$

These Bayesian networks can have large sets of nodes that are topologically identical with respect to their position in the network. The number of nodes in these sets is dependent on a model parameter d denoting the number of neighboring agents. Inference on the network can become cumbersome for large d since the size of the conditional probability table of a payoff valued node is exponential in d. Like in the aforementioned distributed resource models, these payoff valued nodes can be thought of as a summation nodes.

2 BACKGROUND

2.1 Bayesian Networks

A Bayesian Network is a graphical model (X, D, F), consisting of a discrete variable set $X = \{X_1, X_2, \ldots, X_N\}$, a set of corresponding domains: $D = \{D_{X_1}, D_{X_2}, \ldots, D_{X_N}\}$ with $x_i \in D_{X_i}, \forall i$, and a set of parent functions $F = \{F_1, F_2, \ldots, F_N\}$. Each x_i is associated with a parent function $F_i = \Pr(x_i \mid pa_i)$ where pa_i is the set of parent variables of X_i. The conditional probability functions F_i's are typically specified in a tabular format (CPTs).

Bayesian Networks are useful for modeling probabilistic distributions through an efficient representation of conditional independence. Exact inference on Bayesian Network is typically done using bucket elimination (aka variable elimination) (Darwiche, 2009; Dechter, 2013). These exact inference algorithms are known to be exponential in the size of the induced width (or tree-width) of the network and can easily become intractable depending on their induced widths.

2.2 Causal Independence

Casual independence is a probabilistic relationship between a set of causes $\{c_1, \ldots, c_n\}$ and an effect e where the effect can be seen as a deterministic function of hidden variables $\{h_1, ..., h_n\}$ such that $e = h_1 * h_2 ... * h_n$ where each h_i is a probabilistic function of its corresponding c_i and $*$ is a commutative and associative binary operator (Rish and Dechter, 1998; Zhang and Poole, 1996). This paper is directed towards networks possessing causal independence where the $*$ operator is the addition operator $+$ and the effect e can then be expressed as: $e = \sum_i h_i$.

Causal independence in Bayesian Networks such as Fig. 1 enables efficient network transformations (e.g., temporal transformation (Bibartiu et al., 2019; Heckerman and Breese, 1994)) to significantly reduce the size of parent sets in the network.

Network Transformations. Given a CI Bayesian Network fragment $\{X, D, F\}$ with a set of causes $\{c_1, \ldots, c_n\}$, an effect e: $X = \{c_1, ..., c_n, e\}$ and a set of hidden variables $\{h_1, \ldots, h_n\}$, consider a computation ordering over the equation $e = h_1 * h_2 ... * h_n$. An example ordering for a *temporal transformation* is:

$$e = (\ldots(((h_1 * h_2) * h_3) * h_4) * \ldots) * h_n \qquad (3)$$

Given an ordering, denote the quantities enclosed in each parenthesis set as intermediate variables y_i: $\{y_1 = h_1 * h_2,\ y_2 = y_1 * h_3,\ y_3 = y_2 * h_4 \ldots\}$.

A network transformation $\{X', D', F'\}$ is a network such that $X' = \{c_1, \ldots, c_n, h_1, \ldots, h_n, y_1, \ldots, y_m, e\}$ is expanded to include hidden variables h_i and intermediate variables y_i defined over a valid computation ordering. The resulting network is also called a *decomposition network*. The goal is to transform large parent sets to small ones (e.g. 2 variables per set).

Further work on the topic (Rish and Dechter, 1998; Zhang and Poole, 1996) exploit the decomposition graphs resulting from network transformations to accelerate bucket elimination (*ci-elim-bel* in Algorithm 1). It was found that using *ci-elim-bel* to exploit casual independence can significantly improve the performance of exact inference on polytrees (from $O(Nd^m)$ to $O(Nmd^3)$ where N is the number of nodes, d is the domain size and m is the size of the largest parent set) as well as in two layer *k-n-networks* (see Fig. 6) (from $O((k+n)d^k)$ to $O((k+n)d^{min\{k,2n\}})$).

We next provide background into the theory of probability generating functions and the use of Fourier transforms for computing random variable sums that is the main tool we will use to speedup some computations applied to Bayesian Network inference and reductions.

2.3 Random Variable Sums

Suppose we have a set of independent identically distributed (*i.i.d*) random variables $X = \{X_1, X_2, \ldots, X_N\}$ with domain D and a random variable Z s.t:

$$Z = \textstyle\sum_i X_i \qquad (4)$$

Naively, we can find $P(Z=k)$ by enumerating all X_i values that sum to k, taking $O(|D|^N)$ time. However, we can calculate $P(Z=k)$ more efficiently through probability generating functions as described next.

Definition 2.1 (Probability Generating Functions)**.** *A probability generation function (or polynomial) $\mathbb{P}_X$ for a discrete random variable X, with $D_X \subset \mathbb{N}$, having a distribution P_X is defined as:*

$$\mathbb{P}_X(x) = \textstyle\sum_{k=0}^{\infty} x^k P_X(k) \qquad (5)$$

Algorithm 1: ci-elim-bel (Rish and Dechter, 1998)
Input: A Bayesian network $B = (X, D, F)$ where F are CI, evidence e
Output: $P(x_1|e)$

Generate a decomposition network B' from B
 with pair-wise hidden variables $\{u_1, \ldots, u_m\}$
Generate ordering $o = \{Z_1, \ldots, Z_n\}$ using B' s.t.
 $Z_1 = \{x_1\}$ and $Z_i = \{x_j\} \mid \{u_j; u_k\}$
 [for details, see (Rish and Dechter, 1998)]
for $i = n \to 1$ **do** `// create buckets`
 $\forall x \in Z_i$, put all network functions with x as highest ordered variable in $bucket_i$
end
for $i = n \to 1$ **do** `// process buckets`
 `//` $h_1, \ldots, h_m$ `are functions in` $bucket_i$
 if $(x = e_j) \in bucket_i$ *for* $e_j \in e$ **then**
 replace x by e_j in each h_i and put the result in appropriate lower bucket.
 else
 if $Z_i = \{x\}$ **then** `// input variable`
 $h^{Z_i} = \sum_x \prod_j h_j$
 else `//` $Z_i = \{u_l; u_k\}, u = u_l * u_k$
 $h^{Z_i} = \sum_{u_l, u_k | u = u_l * u_k} \prod_j h_j$
 Put h^{Z_i} in the highest bucket that mentions h^{Z_i} 's variable.
end
Return αh^{x_1}, (α is a normalizing constant).

The k-th coefficient of $\mathbb{P}_X$, $P_X(k)$, can be found by taking derivatives of the generating polynomial:

$$P_X(k) = \mathbb{P}_X^{(k)}(0)/i! \qquad (6)$$

where $\mathbb{P}_X^{(k)}$ is the k-th derivative of $\mathbb{P}_X$.

Theorem 2.1 (Generating Function Multiplication, (8.37 in Graham et al. (1989)))**.** *For a set of random variables $X = \{X_1, \ldots, X_N\}$ with distributions $\{P_{X_1}, \ldots, P_{X_N}\}$ and their corresponding generating functions $\{\mathbb{P}_{X_1}, \ldots, \mathbb{P}_{X_N}\}$, the distribution of $Z = \sum_i X_i$ obeys:*

$$P(Z=k) = \mathbb{P}_Z^{(k)}(0)/k!, \quad \textit{where}\ \ \mathbb{P}_Z = \textstyle\prod_j^N \mathbb{P}_{X_j} \qquad (7)$$

Proposition 1. *Computing $(\mathbb{P}_Z)_k, \forall k$ using generating polynomial multiplication takes time $O(|D|^2 \cdot N^2)$.*

Proof. (See supplemental materials for a more in-depth tutorial). For two generating polynomials $\mathbb{P}_{X_1}, \mathbb{P}_{X_2}$, the product is computed as:

$$(\mathbb{P}_{X_1} \times \mathbb{P}_{X_2})_k = \textstyle\sum_{j=0}^{k} (\mathbb{P}_{X_1})_k (\mathbb{P}_{X_2})_{k-j} \qquad (8)$$

where subscripts denote the coefficient of the corresponding polynomial. This computation is quadratic

in the size of the polynomials (number of coefficients). In the process of computing $\mathbb{P}_Z$, the size of intermediate polynomials is bounded by $N|D|$. Therefore the total time for computing $\mathbb{P}_Z$ is:

$$\sum_{i=1}^{N} i|D| \cdot |D| = O(|D|^2 \cdot N^2) \tag{9}$$

□

Definition 2.2 (Convolution)**.** *Let $F_X(k) = P(X = k)$ and $F_Y(k) = P(Y = k)$ be two probability density functions for random variables X, Y. The convolution of functions F_X and F_Y is defined as:*

$$(F_X * F_Y)(k) = \sum_j F_X(k) F_Y(k - j) \tag{10}$$

Clearly $(\mathbb{P}_{X_1} \times \mathbb{P}_{X_2})_k = (P_{X_1} * P_{X_2})(k)$. Consequently, we can use tools for performing convolutions to compute the distribution of a sum.

The Convolution Theorem. The convolution theorem provides an alternative method for the calculation of Eq. 10 using the Fourier Transform.

Definition 2.3 (Discrete Fourier Transform)**.** *The discrete Fourier transform $\mathcal{F}$ of a discrete probability distribution F_X defined over integers $[0, M-1]$ is:*

$$\hat{F}_X(x) = \mathcal{F}\{F_X\}(x) = \sum_{k=0}^{M-1} F_X(k) \cdot e^{-i2\pi xk/M} \tag{11}$$

$\hat{F}_X(x)$ is also defined over M values and is called the Fourier transform of F_X and its domain is called the frequency domain. The inverse Fourier transform is defined as:

$$F_X(k) = \mathcal{F}^{-1}\{\hat{F}_X\}(k) = \frac{1}{M} \sum_{x=0}^{M-1} \hat{F}_X(x) \cdot e^{i2\pi xk/M} \tag{12}$$

The original domain is referred to as the time domain.

Theorem 2.2 (The convolution theorem, (4.3.49 in Proakis (2001)))**.** *For two discrete probability distributions $F_X(t)$ and $F_Y(t)$, it can be shown that:*

$$(F_X * F_Y)(k) = \mathcal{F}^{-1}(\mathcal{F}\{F_X\} \cdot \mathcal{F}\{F_Y\})(k) \tag{13}$$

where $\mathcal{F}\{F_X\} \cdot \mathcal{F}\{F_Y\}$ denotes the pointwise multiplication of the two frequency distributions.

Corollary 2.2.1 (Symmetry)**.** *For i.i.d random variables $X = \{X_1, X_2, \ldots, X_N\}$ and their sum $Z = \sum X_i$, it is easy to show that:*

$$P(Z = k) = \mathcal{F}^{-1}(\mathcal{F}\{F_X\}^N)(k)$$
$$\text{where } F_X(k) = P(X_i = k) \tag{14}$$

Theorem 2.3 (Time Complexity)**.** *For i.i.d random variables $X = \{X_1, X_2, \ldots, X_N\}$ where each variable has a domain size $|D|$ and $Z = \sum X_i$, computing $P(Z = k)$ using Eq. 14 will take time $O(N^2|D|^2)$. For non-i.i.d variables, the time complexity is $O(N^3|D|^2)$.*

Proof. See supplemental materials □

This computation can be accelerated using the **Fast Fourier Transform** (FFT). The FFT is a collection of divide and conquer algorithms (e.g., the Cooley-Tukey algorithm (Cooley and Tukey, 1965)) that reduces the time required for computing the Fourier transform from $O(|D|^2)$ to $O(|D| \log |D|)$ where $|D|$ is the domain size of the discrete distribution being transformed.

Theorem 2.4 (FFT Time Complexity)**.** *For i.i.d random variables $X = \{X_1, X_2, \ldots, X_N\}$ where each variable has a domain size $|D|$ and their sum $Z = \sum X_i$, computing $P(Z = k)$ using the FFT will take time $O(N|D| \log(N|D|))$. For non-i.i.d variables, the time complexity is $O(N^2|D| \log(N|D|))$.*

Proof. See supplemental materials □

In practice, even if X_i is not defined over consecutive integers, it is still possible to calculate the sum distribution of Z using the FFT, however the time complexity is better expressed using the range of Z:

$$R = \max(Z) - \min(Z)$$
$$\text{Time} = O(R \log R) \tag{15}$$

3 APPLICATION TO BAYESIAN NETWORKS

3.1 FFT Reduction

Up to this point, we have been working with sums of random variables. Now, we will apply the FFT theory towards the reduction of variables in a Bayesian Network.

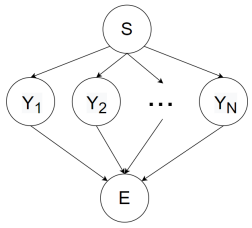

Figure 2: Symmetric Bayesian Network with i.i.d paths between source node S and sink node E

Consider the Bayesian Network in fig. 2 which has 3 types of nodes.

- Source node S take a value in some domain D_S
- A set of N *i.i.d* nodes Y_i that take values in some domain D_Y

- Sink node E takes a value in a subset of the natural numbers $D_E \subset \mathbb{N}$

Let $U : D_Y \to \mathbb{N}$ be some cost function. We assume:

$$P(E=e \mid \{Y_i=y_i, \forall i\})) = \begin{cases} 1 & \text{if } e = \sum_i^N U[y_i] \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

The size of this function expressed as conditional probability table is exponential in N. However, using FFT theory, we can reduce the size of this CPT from $O(|D_E||D_Y|^N)$ to $O(|D_S||D_E|)$ by eliminating $Y_i, \ldots, Y_N$.

Theorem 3.1 (FFT Reduction)**.** *Let $B = \{X, D, F\}$ with $X = \{S, Y_1, \ldots, Y_N, E\}$ be a source-sink network with N i.i.d paths as in Fig. 2. The network can be transformed into $\{X', D', F'\}$ such that $X' = \{S, E\}$ reducing the CPT for E from size $O(|D_E||D_Y|^N)$ to size $O(|D_S||D_E|)$ in $O(|D_S|R\log R)$ time where R is the numerical range of random variable E.*

Proof. Our goal is to define a new CPT for E of size $O(|D_S||D_E|)$ where each entry is:

$$P(E=e \mid S=s),\ \forall s \in D_S, e \in D_E \tag{17}$$

We use the FFT reduction to eliminate all *i.i.d* nodes Y_i in between the S and E nodes. For each value of (s, e), we proceed as follows:

1. To calculate $P(\mathbf{U}[Y_i] = j|S = s)$, we transform the distribution on $Y's$ from the domain D_Y to a distribution on the natural numbers domain $\mathbb{N}$. This takes $O(|D_Y|)$ time as we simply iterate through $\mathbf{U}[y], \forall y \in D_Y$.
2. Let:

$$Z = \textstyle\sum_i \mathbf{U}[Y_i] \tag{18}$$

Let F_u be the discrete probability distribution of $U[Y_i]$:

$$F_u(j) = P(U[Y_i]=j \mid S=s) \tag{19}$$

Using the convolution theorem:

$$\begin{aligned} P(E=e \mid S=s) &= P(Z=e \mid S=s) \\ &= \mathcal{F}^{-1}\{\mathcal{F}\{F_u\}^N\}(e) \end{aligned} \tag{20}$$

which takes $O(R\log R)$ where R is Z's range:

$$R = N(\max \mathbf{U} - \min \mathbf{U}) \tag{21}$$

We perform the above steps for each value of $s \in D_S$ giving a final time complexity of:

$$\begin{aligned} &O(|D_S|R\log R) \\ &R = N(\max \mathbf{U} - \min \mathbf{U}) \end{aligned} \tag{22}$$

to reduce the CPT to $O(|D_E|\,|D_S|)$. □

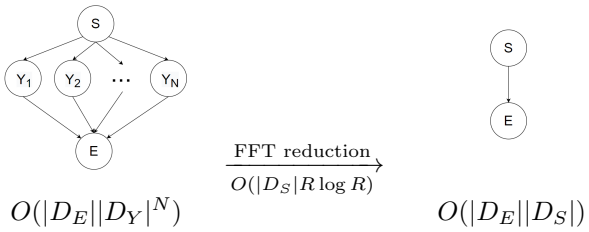

Figure 3: FFT Reduction

Corollary 3.1.1 (Reduction on general CI Networks)**.** *For any set of random variables $X = \{X_1, \ldots, X_N\}$ (not necessarily identically distributed) and corresponding sum $Z = \sum_i X_i$, we can calculate all probabilities $P(Z = k)$ in $O(N^2|D|\log(N|D|))$ time.*

Proof. See supplemental materials. □

3.2 Algorithm CI-Elim-Bel with FFT

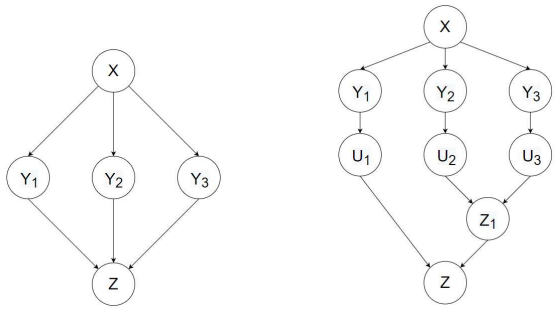

Figure 4: Five node source-sink Bayesian Network (a) and its decomposition graph (b)

In this section we will show how the use of FFT for processing summation can be incorporated into the bucket-elimination algorithm *ci-elim-bel*. We will illustrate this with an example. Consider the five node Bayesian Network in Fig. 4 where all Y_i's are i.i.d and are defined by $P(Y_i|X) = P(Y_k|X),\ \forall k.$ and assume $Z = \sum_i Y_i$. Like in (Rish and Dechter, 1998), we can perform *ci-elim-bel* on this network using the decomposition graph in Fig. 4 (See supplemental Fig. 1 for the equation describing the full computation). The elimination steps are as follows for ordering $o = \{\{u_1, z_1\}, Y_1, \{u_2, u_3\}, Y_2, Y_3\}$:

1. bucket $Y_3 : h^{Y_3}(X, u_3) = \sum_{Y_3} P(Y_3|X)P(u_3|Y_3)$
2. bucket $Y_2 : h^{Y_2}(X, u_2) = \sum_{Y_2} P(Y_2|X)P(u_2|Y_2)$
3. bucket $\{u_2, u_3\} : h^{\{u_2,u_3\}}(X, z_1) = \sum_{\{u_2,u_3|z_1=u_2+u_3\}} h^{Y_2}(X, u_2)h^{Y_3}(X, u_3)$

4. bucket $Y_1 : h^{Y_1}(X, u_1) = \sum_{Y_1} P(Y_1|X)P(u_1|Y_3)$
5. bucket $\{u_1, z_1\} : P(Z|X) = \sum_{\{u_1,z_1|z=u_1+z_1\}} h^{y_1}(X, u_1)h^{\{u_1,u_2\}}(X, z_1)$

where h^{Y_i} denotes intermediate functions. The largest operation occurs in bucket $\{u_1, z_1\}$ which has a complexity of $|D_X||D_Z||D_Y|$.

Observe that the calculations performed in the buckets $\{u_2, u_3\}$ and $\{u_1, z_1\}$ are equivalent to multiplying the coefficients of the corresponding generating polynomials. For example, consider the following three polynomials:

$$\begin{aligned}\mathbb{P}_{h^{Y_2}} &= \textstyle\sum_i h^{Y_2}(X, u_2 = i)x^i \\ \mathbb{P}_{h^{Y_3}} &= \textstyle\sum_i h^{Y_3}(X, u_3 = i)x^i \\ \mathbb{P}_{h^{\{u_2,u_3\}}} &= \textstyle\sum_i h^{\{u_2,u_3\}}(X, z_1 = i)x^i \end{aligned} \tag{23}$$

We have that:

$$h^{\{u_2,u_3\}}(X, z_1) = \sum_{\{u_2,u_3|z_1=u_2+u_3\}} h^{Y_2}(X, u_2)h^{Y_3}(X, u_3)$$

is exactly the equation for the calculation of the coefficients in the polynomial product:

$$\mathbb{P}_{h^{\{u_2,u_3\}}} = \mathbb{P}_{h^{Y_2}} \times \mathbb{P}_{h^{Y_3}} \tag{24}$$

It follows then that for a given number N of Y_i nodes, variable elimination using a decomposition graph takes the same amount of time as the generating function polynomial multiplication. However, as we have shown earlier using the FFT reduction is computationally faster compared to the generating function approach yielding potentially a speedup compared to CI-based bucket elimination.

In general, suppose in the variable elimination calculation we have the following sequence of buckets in the *ci-elim-bel* algorithm for a temporal decomposition of N, Y_i nodes:

- bucket $\{u_1, u_2\} : h^{\{u_1,u_2\}}(X, z_1) = \sum_{\{u_1,u_2|z_1=u_1+u_2\}} h^{Y_1}(X, u_1)h^{Y_2}(X, u_2)$
- bucket $\{u_3, z_1\} : h^{\{u_3,z_1\}}(X, z_2) = \sum_{\{u_3,z_1|z_2=u_3+z_1\}} h^{Y_3}(X, u_3)h^{\{u_1,u_2\}}(X, z_1)$
- bucket $\{u_4, z_2\} : h^{\{u_4,z_2\}}(X, z_3) = \sum_{\{u_4,z_2|z_3=u_4+z_2\}} h^{Y_4}(X, u_4)h^{\{u_3,z_1\}}(X, z_2)$
- $\vdots$
- bucket $\{u_N, z_{N-2}\} : P(Z|X) = \sum_{\{u_N,z_{N-2}|z=u_N+z_{N-2}\}} h^{Y_N}(X, u_N) \; h^{\{u_{N-1},Z_{N-3}\}}(X, z_{N-2})$

	i.i.d $Y_i's$	non-*i.i.d* $Y_i's$
BE	$O(N\|D_X\|R_Z^2)$	$O(N\|D_X\|R_Z^2)$
FFT	$O(\|D_X\|R_Z \log R_Z)$	$O(N\|D_X\|R_Z \log R_Z)$

Table 1: Time complexity for computation of temporal decomposition buckets. Top row (Bucket Elimination), Bottom row (FFT)

Computationally, the bucket computations will take $O(N^2|D_U|^2|D_X|)$ time. Observe that if the Y_i's are identically distributed conditioned on X, we have that

$$h^{Y_i}(X, u_i) = h^{Y_j}(X, u_j), \quad \forall i, j \in [1, N] \tag{25}$$

Theorem 3.2 (FFT bucket elimination)**.** *Given a sequence of N temporal decomposition buckets in ci-elim-bel for N i.i.d Y_i's that conditionally depend on variable X with domain size $|D_X|$ and sum to variable Z, it is possible to compute the sequence of buckets in $O(|D_X|R_Z \log R_Z)$ time where R_Z is the numerical range of the domain of the final bucket.*

Proof. We perform the following FFT reduction:

1. Define functions F_{Y_i}
$$F_{Y_i}(X, k) = h^{Y_i}(X, u_i = k)$$
2. Perform the FFT: $(O(|D_X|R_Z \log R_Z))$
$$\hat{F}_{Y_i}(X, t) = \mathcal{F}\{F_{Y_i}(X)\}(t)$$
3. Exponentiate the function $\hat{F}_{Y_i}$: $(O(R_Z))$
$$\hat{F}_{Y_i}(X, t)^N$$
4. Perform the inverse FFT: $(O(|D_X|R_Z \log R_Z))$
$$P(Z|X) = \mathcal{F}^{-1}\{\mathcal{F}\{F_{Y_i}(X)\}^N\}(Z)$$

Which in total takes $O(|D_X|R_Z \log R_Z)$ time. □

If the Y_i's are not identically distributed, we will need to perform an FFT for each F_{Y_i} and replace step 3 with pointwise multiplications which in total will take $O(NR_Z)$ time bringing the total time to $O(N|D_X|R_Z \log R_Z)$. A comparison of the different methods can be seen in Table 1. For comparison purposes, we can approximately substitute $N|D_X| \approx R_Z$.

Extending *ci-elim-bel*. Using *ci-elim-bel* as the baseline, we can replace the computation of + in addition based networks with the FFT to create the *ci-elim-FFT* algorithm (see Algorithm 2). The FFT operation to compute h^{Z_i} is performed with respect

Algorithm 2: ci-elim-FFT
Input: A Bayesian network B, evidence e
Output: $P(x_1|e)$

Generate a decomposition network from B
Generate ordering $o = \{Z_1, \ldots, Z_n\}$ with $Z_1 = \{x_1\}$ using B'
for $i = n \to 1$ **do** `// create buckets`
  $\forall x \in Z_i$, put all network functions with x as highest ordered variable in $bucket_i$
end
for $i = n \to 1$ **do** `// process buckets`
  `//` $h_1, \ldots, h_m$ `are functions in` $bucket_i$
  if $Z_i = \{u_l; u_k\}, u = u_l + u_k$ **then**
    $h^{u_l} = \prod_{j, u_l \in h_j} h_j$;
    $h^{u_k} = \prod_{j, u_k \in h_j} h_j$;
    $h^{Z_i} = \mathcal{F}^{-1}\{\mathcal{F}\{h^{u_l}\} \cdot \mathcal{F}\{h^{u_k}\}\}$
  else
    use regular ci-elim-bel to compute h^{Z_i}
  Put h^{Z_i} in the highest bucket that mentions h^{Z_i}'s variable.
end
Return αh^{x_1}, (α is a normalizing constant).

to the additive variables y_l and y_k. This means that we evaluate:

$$h^{Z_i} = \mathcal{F}^{-1}\{\mathcal{F}\{h^{y_l}\} \cdot \mathcal{F}\{h^{y_k}\}\} \tag{26}$$

for every value of the variables of h^{Z_i} except y, y_l, y_k.

4 EXPERIMENTAL EVALUATION

We evaluate the effectiveness of both approaches proposed for accelerating inference in Bayesian Networks. Specifically we evaluate the FFT reduction technique [as in section 3.1] for network transformation on a selection of common substructures that may be present in larger networks. We also evaluate the performance of *ci-elim-FFT* compared to *ci-elim-bel* on general two layer additive networks also known as $k - n$ networks. Rish (1999) showed that *ci-elim-bel* can speed up inference exponentially on $k - n$ networks when $k > 2n$. We test the scaling efficiency of *ci-elim-FFT* in cases where $k > 2n$ where one might want to use *ci-elim-bel*.

4.1 Experimental Setup

We evaluate all inference tasks on a 64-bit machine with an Intel i7-10870H 2.2 GHz CPU and 32 GB of RAM. The models are written in python with the library pomegranate and bucket elimination is performed using the library's pgmpy's built-in variable elimination. The FFT is computed using the library numpy. We also compare python implementations

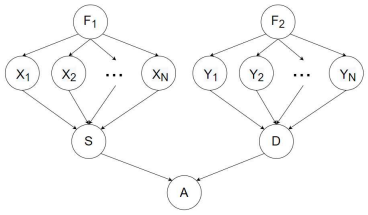

Figure 5: Two branch supply-demand network

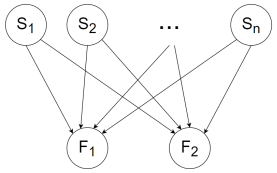

Figure 6: Two layer additive network

(not using pgmpy) of *ci-elim-FFT* with *ci-elim-bel* where the only difference is the method for computing buckets with two hidden variables.

4.2 FFT reduction evaluation

We evaluate the FFT reduction for accelerating inference on a selection of networks with casually independent summation nodes. For each network scenario, we test three different approaches:

1. vanilla bucket elimination (Naive)
2. bucket elimination on a temporal network decomposition as in Bibartiu et al. (2019) (Temporal)
3. FFT reduction [as in Section 3.1] followed by bucket elimination (FFT)

We test each approach for finding the marginal distribution of a specific variable in three scenarios:

1. $P(K = k)$ in an n-parent convergent network (Fig. 1) found in the distributed resource applications described in Bibartiu et al. (2019)
2. $P(E = e)$ in an n-path source-sink graph (Fig. 2) which can be found as substructures of the Bayesian Networks for evolutionary games in Hsiao et al. (2021)
3. $P(A = a)$ in a supply and demand model for distributed resource production and consumption shown in Fig. 5. This consists of two n-parent convergent networks that feed into a boolean-valued node that is True if supply is larger or equal to than demand and False otherwise.

For each network class, we evaluate end-to-end inference time (construction time and inference time) for increasing numbers of casually independent nodes (S_i

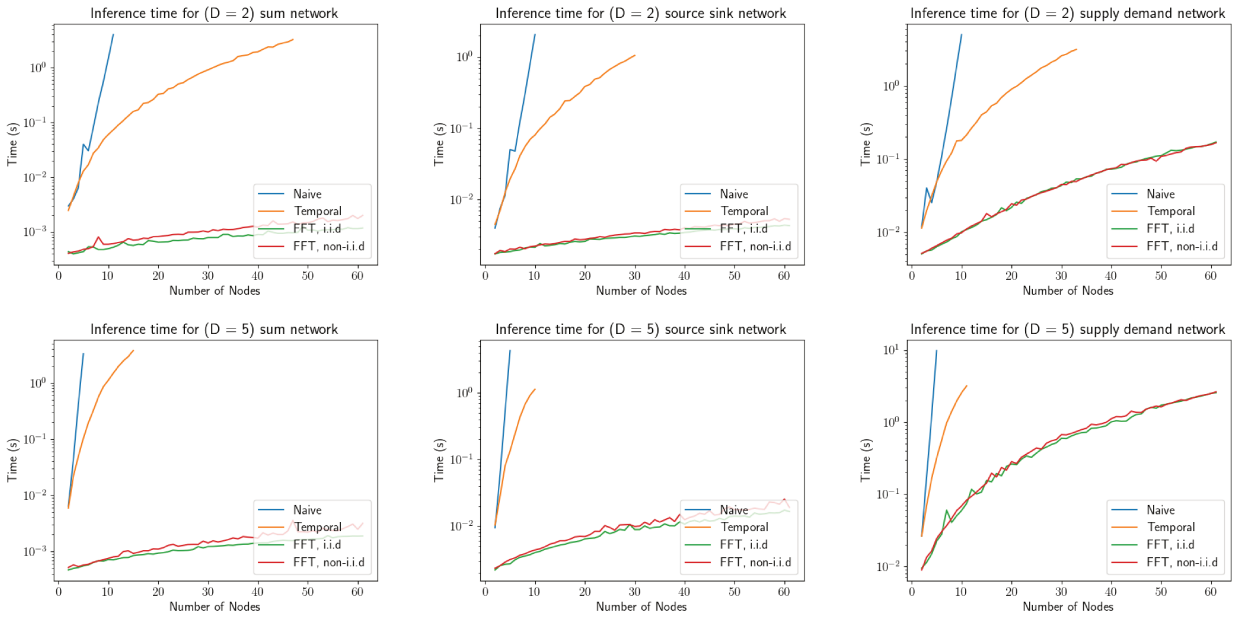

Figure 7: Inference times for three scenarios controlled for domain sizes (D = 2, D = 5)

in Fig. 1, Y_i in Fig. 2, and X_i, Y_i in Fig. 5). We also evaluate inference time for the FFT approach on networks with *i.i.d* intermediate nodes vs. non-*i.i.d* nodes.

4.3 Results of FFT reduction

In the three scenarios tested, as seen in Fig. 7, the FFT reduction method obtains a significant computational advantage over the naive approach as well as inference on the temporal decomposition. In Fig. 7, it is easy to see how each method scales relative to their theoretical complexity. The naive bucket elimination approach scales exponentially while the temporal decomposition method scales at a cubic rate. In comparison, employing the FFT reduction to remove *i.i.d* nodes before performing bucket elimination significantly decreases the computational cost of subsequent inference.

Surprisingly, there is not much of a difference between networks with *i.i.d* nodes versus non-*i.i.d* nodes for the FFT approach, suggesting that the inference time for the reduced network overshadows the complexity of performing the FFT reduction.

4.4 Evaluating ci-elim-FFT

We compare the computational efficiency of *ci-elim-bel* with *ci-elim-FFT* on 2-layer additive networks (Fig. 6). This is a generalization of the Binary 2-layer Noisy-Or (BN2O) networks commonly used in medical diagnosis. Instead of binary variables, we assume that the sources can be stochastically transformed into integer-valued variables which sum to an integer valued result. These types of networks can be used to model numerical medical findings such as body temperature or blood test results where separate causes (diseases and medications) have additive effects on the observed findings. We measure the efficiency of finding

$$P(S_1|F_1 = e_1, F_2 = e_2) \tag{27}$$

given findings (e_1, e_2). We control for the number of causes ($\{S_1, \ldots, S_N\}$ in Fig. 6) as well as the domain size of each variable.

Fig. 8 demonstrates *ci-elim-FFT*'s clear advantage over *ci-elim-bel*. This advantage increases both as the

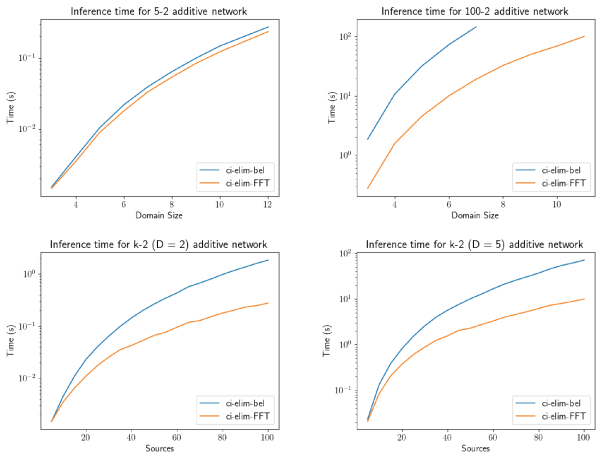

Figure 8: Inference times for two layer additive networks controlling for domain size (top) and number of source nodes (bottom)

number of source nodes increases and as well as when the domain size of variables increases.

5 RELATED WORK AND LIMITATIONS

Our work is not the first work which has attempted to apply the Discrete Fourier Transform (DFT) to exact probabilistic inference on Bayesian Networks. The DFT can be thought of as a special case of the tensor decomposition approach first described in Savicky and Vomlel (2007). In Plajner and Vomlel (2021), the DFT was applied as a special case of the tensor decomposition towards test score prediction on a subset of the source-sink type networks that we discussed in section 3. However, in their approach, a naive application of the DFT was used that relies on matrix multiplication which is equivalent to computing Eq. 11 directly. Due to this, we believe that our FFT based approach can provide a method for lowering the computational cost of test score prediction even further beyond the existing DFT approach using matrix multiplication. Furthermore, our work also improves upon prior work by directly integrating the FFT into a general bucket elimination algorithm.

The FFT itself has also been applied towards the calculation of random variable sums in Beyene (2001), but not directly towards the similar problems as found in summation type nodes in Bayesian Networks or combined with algorithms such bucket elimination.

Conceptually, the problem of accelerating the computation of summation-based CPTs is also similar to the problem addressed through the use of counting factors in the literature on lifted variable elimination (Taghipour et al., 2013). However, there are a few key differences. Most notably, while lifting is said to work on general models, in practice several restrictions must be satisfied for lifting to provide a computational advantage. In particular, it is necessary for the factors present in the equation to be identical. This allows work to be exponentiated out through symmetry and is equivalent to the case of $i.i.d$ variables in our work. If the factors are not identical, then it is necessary to perform grounding, defaulting to basic variable elimination.

While our FFT method provides the greatest computational advantage when all variables are $i.i.d$, we note in corollary 3.3.1 and in the paragraph right after theorem 3.2, that the method can still be applied even when the variables are not $i.i.d$, providing theoretical speedup (as shown in Table 1) and an empirical speedup as observed (in Figure 7, 8). Importantly, CI-elim-FFT can be applied to any Bayesian Network that include summation nodes, and not just to the four types of networks demonstrated in the empirical results. For general networks, the only requirement for ci-elim-FFT to provide a speedup over a temporal decomposition/ci-elim-bel is the presence of summation type nodes.

It may be interesting to incorporate our FFT method as a type of lifted operator for counting factors over asymmetric variables, extending ci-elim-FFT into an analagous lifted version. We leave this problem for future research in this area.

6 CONCLUSION

We have presented an efficient method for reducing the size of summation-based Conditional Probability Tables (CPTs) in Bayesian Networks having *causal independence* (CI). We also have shown how to apply this reduction directly towards the acceleration of Bucket Elimination. We have provided experimental results showing the FFT reduction's advantage for inference on a selection of common sub-networks found in Bayesian Networks for modeling distributed resource. We have developed an extension to *ci-elim-bel* called *ci-elim-FFT* and provided empirical results that demonstrate its scaling advantages.

Acknowledgements

This work supported in part by NSF grant IIS-2008516 and AFOSR grant 1010GWA357. The information in this paper does not necessarily reflect the position or policy of the funders, and no official endorsement should be inferred.

References

Bassamzadeh, N. and Ghanem, R. (2017). Multiscale stochastic prediction of electricity demand in smart grids using bayesian networks. *Applied energy*, 193:369–380.

Beyene, J. (2001). *Uses of the fast Fourier transform (FFT) in exact statistical inference*. PhD thesis, National Library of Canada= Bibliothèque nationale du Canada.

Bibartiu, O., Dürr, F., Rothermel, K., Ottenwälder, B., and Grau, A. (2019). Towards scalable k-out-of-n models for assessing the reliability of large-scale function-as-a-service systems with bayesian networks. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 514–516. IEEE.

Cai, B., Liu, Y., Fan, Q., Zhang, Y., Liu, Z., Yu, S., and Ji, R. (2014). Multi-source information fusion

based fault diagnosis of ground-source heat pump using bayesian network. *Applied energy*, 114:1–9.

Cooley, J. W. and Tukey, J. W. (1965). An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301.

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

Dechter, R. (2013). Reasoning with probabilistic and deterministic graphical models: Exact algorithms. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 7(3):1–191.

Graham, R. L., Knuth, D. E., Patashnik, O., and Liu, S. (1989). Concrete mathematics: a foundation for computer science. *Computers in Physics*, 3(5):106–107.

Heckerman, D. and Breese, J. S. (1994). A new look at causal independence. In *Uncertainty Proceedings 1994*, pages 286–292. Elsevier.

Hsiao, V., Pan, X., Nau, D., and Dechter, R. (2021). Approximating spatial evolutionary games using bayesian networks. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1533–1535.

Jiang, Y., Zhang, H., Song, X., Jiao, X., Hung, W. N., Gu, M., and Sun, J. (2012). Bayesian-network-based reliability analysis of plc systems. *IEEE transactions on industrial electronics*, 60(11):5325–5336.

Plajner, M. and Vomlel, J. (2021). Bayesian networks for the test score prediction: A case study on a math graduation exam. In *European Conference on Symbolic and Quantitative Approaches with Uncertainty*, pages 255–267. Springer.

Portinale, L. and Bobbio, A. (2013). Bayesian networks for dependability analysis: an application to digital control reliability. *arXiv preprint arXiv:1301.6734*.

Proakis, J. G. (2001). *Digital signal processing: principles algorithms and applications*. Pearson Education India.

Rish, I. (1999). *Efficient reasoning in graphical models. PhD thesis*. University of California, Irvine.

Rish, I. and Dechter, R. (1998). On the impact of causal independence. Technical report, AAAI SS-98-03. Dept. Information and Computer Science, UCI.

Savicky, P. and Vomlel, J. (2007). Exploiting tensor rank-one decomposition in probabilistic inference. *Kybernetika*, 43(5):747–764.

Taghipour, N., Fierens, D., Davis, J., and Blockeel, H. (2013). Lifted variable elimination: Decoupling the operators from the constraint language. *Journal of Artificial Intelligence Research*, 47:393–439.

Zhang, N. L. and Poole, D. (1996). Exploiting causal independence in bayesian network inference. *Journal of Artificial Intelligence Research*, 5:301–328.

Fast Fourier Transform Reductions for Bayesian Network Inference: Supplementary Materials

A Supplemental Proofs

A.1 Tutorial for Generating Function Multiplication in Proposition 1

Algorithm 3: Multiply Method 1
Input: Two generating polynomials $\mathbb{P}_{X_1}, \mathbb{P}_{X_2}$
Output: The product: $\mathbb{P}_{X_1+X_2}$

let $M_1 =$ size of $\mathbb{P}_{X_1}$, let $M_2 =$ size of $\mathbb{P}_{X_2}$;
let $M = M_1 + M_2$ // For just 2 polynomials this is $2|D|$;
for $i = 0 \rightarrow M$ **do** `// outer loop`
  initialize $(\mathbb{P}_{x_1+x_2})_i = 0$;
  for $j = 0 \rightarrow i$ **do** `// inner loop`
    if $j \in \{0, M_1 - 1\}$ *and* $i - j \in \{0, M_2 - 1\}$ **then**
      $(P_{x_1+x_2})_i \mathrel{+}= (P_{x_1})_j \cdot (P_{x_2})_{i-j}$
  end
end
return $P_{x_1+x_2}$

Algorithm 4: Multiply Method 2
Input: Two generating polynomials $\mathbb{P}_{X_1}, \mathbb{P}_{X_2}$
Output: The product: $\mathbb{P}_{X_1+X_2}$

let $M_1 =$ size of $\mathbb{P}_{X_1}$, let $M_2 =$ size of $\mathbb{P}_{X_2}$;
for $i = 0 \rightarrow M_1 + M_2$ **do** `// outer loop`
  initialize $(\mathbb{P}_{x_1+x_2})_i = 0$;
end
for $i = 0 \rightarrow M_1$ **do** `// outer loop`
  for $j = 0 \rightarrow M_2$ **do** `// inner loop`
    $(P_{x_1+x_2})_{i+j} \mathrel{+}= (P_{x_1})_i \cdot (P_{x_2})_j$
  end
end
return $P_{x_1+x_2}$

Algorithm 5: Multiply N Polynomials
Input: N generating polynomials $\mathbb{P}_{X_1}, \mathbb{P}_{X_2}, \ldots, \mathbb{P}_{X_N}$
Output: The product: $\mathbb{P}_Z = \mathbb{P}_{X_1+X_2+\ldots+X_N}$

initialize $\mathbb{P}_Z = P_{X_1}$;
for $2 \rightarrow N$ **do**
  $P_Z = Multiply(P_Z, P_{X_k})$ // call either Method 1 or Method 2
end
return P_Z

Consider two generating polynomials $\mathbb{P}_{X_1}, \mathbb{P}_{X_2}$, the first with M_1 coefficients and the second with M_2 coefficients, the product is computed as:

$$(\mathbb{P}_{X_1} \times \mathbb{P}_{X_2})_k = \sum_{j=0}^{k} (\mathbb{P}_{X_1})_k (\mathbb{P}_{X_2})_{k-j} \tag{28}$$

where subscripts denote the coefficient of the corresponding polynomial. Explicitly, this computation can be computed using either Method 1 in Algorithm 3 (directly using Eq. (8)) or using Method 2 (using a pair-wise product) in Algorithm 4. In either case, this computation is quadratic in the size of the polynomials (number of coefficients). For this analysis, we define time complexity as the number of floating point multiplication operations performed. These occur only when we multiply two coefficients together.

For two polynomials, if $M_1 = M_2 = |D|$, the time complexity is:

- In the case of Method 1, the outer loops iterates for M steps and the inner loops iterates up to the intermediate counter i. The inner multiplication is only evaluated if $j \in \{0, M_1 - 1\}$ and $i - j \in \{0, M_2 - 1\}$. As a result, only $M_1 \cdot M_2$ products are ever evaluated: $O(M_1 \cdot M_2) = O(|D|^2)$
- In the case of Method 2, the outer loops iterates for M_1 steps and the inner loops iterates up to M_2: $O(M_1 \cdot M_2) = O(|D|^2)$

Note that neither method calculates any multiplication term (of which there are $M_1 \cdot M_2$ terms) more than once so the true number of pair-wise multiplications using either method is the same.

Consider the multiplication of N polynomials using Algorithm 5:

$$\mathbb{P}_Z = \prod_{j=1}^{N} \mathbb{P}_{X_j} = (\ldots(\mathbb{P}_{X_1} \times \mathbb{P}_{X_2}) \times \ldots \times \mathbb{P}_{X_N}) \tag{29}$$

Algorithm 5 computes this quantity sequentially and produces intermediate polynomials:

- $(\mathbb{P}_{X_1} \times \mathbb{P}_{X_2})$
- $(\ldots(\mathbb{P}_{X_1} \times \mathbb{P}_{X_2}) \times \mathbb{P}_{X_3})$
- $(\ldots(\mathbb{P}_{X_1} \times \mathbb{P}_{X_2}) \times \ldots \times \mathbb{P}_{X_i})$

Given two polynomials $\mathbb{P}_{X_1}, \mathbb{P}_{X_2}$ with sizes M_1, M_2, the size of their product is at most $M_1 + M_2$. Consequently, the size of intermediate polynomials $(\ldots(\mathbb{P}_{X_1} \times \mathbb{P}_{X_2}) \times \ldots \times \mathbb{P}_{X_i})$ is at most $i|D|$ (bounded by $N|D|$). Each call on multiply is called on two polynomials:

1. Intermediate polynomial: $(\ldots(\mathbb{P}_{X_1} \times \mathbb{P}_{X_2}) \times \ldots \times \mathbb{P}_{X_i})$. Size: $i|D|$
2. Next polynomial $\mathbb{P}_{X_i+1}$. Size: $|D|$

The complexity of Multiply on these two polynomials is $i|D| \cdot |D|$ (there are $i|D| \cdot |D|$ pairwise products to evaluate). There are $N - 2$ calls to Multiply. Therefore the total time for computing $\mathbb{P}_Z$ is:

$$\sum_{i=2}^{N-1} i|D| \cdot |D| = O(|D|^2 \cdot N^2) \tag{30}$$

which is Eq. (9) in the paper.

A.2 Proof of Theorem 2.3

Proof. When computing the discrete Fourier transform, the size of the distributions before and after the transforms are applied must be the same. It is necessary to pad the starting distributions F_X with zeros up to the size of the target sum Z which is $O(N|D|)$. The direct computation of the Fourier Transforms is quadratic in $N|D|$. Sequentially, the computation consists of:

- Fourier transform $\mathcal{F}$: $O(\big(N|D|\big)^2)$
- Exponentiation $\mathcal{F}\{F_X\}^N$: $O(N|D|)$
- Inverse transform $\mathcal{F}^{-1}$: $O(\big(N|D|\big)^2)$

which in total is $O(N^2|D|^2)$.

In the case of non-$i.i.d$ variables, we perform a Fourier Transform for each distribution P_{X_i}, $O(N\big(N|D|\big)^2)$ and perform point-wise multiplications $O(N^2|D|)$ in the frequency domain bringing the total time to $O(N^3|D|^2)$. □

A.3 Proof of Theorem 2.4

Proof. The proof is the same as the previous theorem with the Fourier transform computed instead using the FFT in $O(N|D|\log N|D|)$ time. Consequently the time required to calculate the distribution over the sum of N $i.i.d$ variables defined over consecutive integers is reduced to:

$$\text{Time}(i.i.d) = O(N|D|\log(N|D|)). \tag{31}$$

and for non-$i.i.d$ variables:

$$\text{Time}(\text{non-}i.i.d) = O(N^2|D|\log(N|D|)). \tag{32}$$

□

A.4 Proof of Corollary 3.1.1

Proof. Simply apply the process for FFT reduction without conditioning on values of a source node S:

1. FFT for each X_i: $O(N \cdot N|D|\log(N|D|))$
2. N point-wise multiplications in Fourier Domain $O(N^2|D|)$
3. Inverse FFT for Z: $O(N|D|\log(N|D|))$

Adding these up results in $O(N^2|D|\log(N|D|))$ complexity. □

$$
\begin{aligned}
P(Z|X) &= \sum_{Y_1}\sum_{Y_2}\sum_{Y_3} P(Y_1|X)P(Y_2|X)P(Y_3|X)P(Z|Y_1,Y_2,Y_3) \\
&= \sum_{Y_1,Y_2,Y_3} P(Y_1|X)P(Y_2|X)P(Y_3|X) \sum_{\{u_1,u_2,u_3,z=u_1+u_2+u_3\}} P(u_1|Y_1)P(u_1|Y_2)P(u_1|Y_3) \\
&= \sum_{\{u_1,z_1|z=u_1+z_1\}} \sum_{Y_1} P(Y_1|X)P(u_1|Y_1) \sum_{\{u_2,u_3|z_1=u_2+u_3\}} \sum_{Y_2} P(Y_2|X)P(u_2|Y_2) \sum_{Y_3} P(Y_3|X)P(u_3|Y_3) \qquad (33)
\end{aligned}
$$

Figure 9: Equation for computing bucket elimination on Five node source-sink Bayesian Network