Compressing Bayesian Networks: Swarm-Based Descent, Efficiency, and Posterior Accuracy

Yang Xiang and Benjamin Baird

University of Guelph, Canada

Abstract. Local models in Bayesian networks (BNs) reduce space complexity, facilitate acquisition, and can improve inference efficiency. This work focuses on Non-Impeding Noisy-AND Tree (NIN-AND Tree or NAT) models whose merits include linear complexity, being based on simple causal interactions, expressiveness, and generality. We present a swarm-based constrained gradient descent for more efficient compression of BN CPTs (conditional probability tables) into NAT models. We show empirically that multiplicatively factoring NAT-modeled BNs allows significant speed up in inference for a reasonable range of sparse BN structures. We also show that such gain in efficiency only causes reasonable approximation errors in posterior marginals in NAT-modeled real world BNs.

Keywords: Uncertainty, Bayesian Networks, Causal Independence Models

1 Introduction

Local models (modeling the CPT over an effect and its causes) in BNs, such as noisy-OR [9], noisy-MAX [3, 2], context-specific independence (CSI) [1], recursive noisy-OR [6], NIN-AND Tree or NAT [12], DeMorgan [7], cancellation model [11], and tensor-decomposition [10], reduce the space complexity of BNs, facilitate knowledge acquisition, and can improve efficiency for inference. This work focuses on NAT models [12], whose merits include linear complexity, being based on simple causal interactions (reinforcement and undermining), expressiveness (recursive mixture, multi-valued, ordinal and nominal [13]), and generality (generalizing noisy-OR, noisy-MAX [14], and DeMorgan [12]). In addition, NAT models support much more efficient inference, where two orders of magnitude speedup in lazy propagation is achieved in very sparse NAT-modeled BNs [14]. Since causal independence encoded in a NAT model is orthogonal to CSI, NAT models provide an alternative mechanism to CSI for efficient inference in BNs.

This work advances the state of the art of NAT modeling around three issues. Given a general BN, it can be compressed into a NAT-modeled BN for improved space and inference efficiency. A key step of compression is to parameterize alternative NAT models through constrained gradient descent. We investigate a swarm-based constrained gradient descent for more efficient compression.

NAT-modeling has been shown to improve inference efficiency in very sparse BNs. In this work, we apply NAT-modeling to BNs with a wider range of structural densities, and assess the impact on inference efficiency in that range.

Compression of a general BN into a NAT-modeled BN introduces approximation errors. The compression errors have been evaluated through Kullback−Leibler divergence and Euclidean distance between the target CPT (from the general BN)

and the NAT CPT [13]. In this work, we investigate the impact of compression errors on the posterior marginals from inference, which provide a more direct evaluation of approximation errors of NAT-modeling.
The remainder of the paper is organized as follows. Section 2 reviews the background on NAT modeling. The swarm-based parameterization is presented in Section 3. Evaluations on efficiency gain in inference due to NAT modeling and on posterior accuracy are reported in Sections 4 and 5, respectively.

2 Background on NAT Models

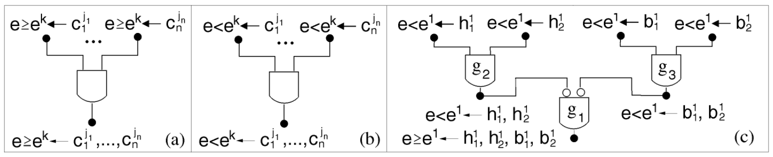
NAT models deal with *uncertain* causes, that can render their effects but do not always do so. The effect and causes in NAT models are *causal variables*.

Definition 1 *[13] A variable x that can be either inactive or be active in multiple ways, and is involved in a causal relation, is a* **causal variable** *if when all causes of the effect are inactive, the effect is inactive with certainty.*

A causal variable can be ordinal or nominal, and hence is more general than the *graded* variable, commonly assumed in noisy-OR or noisy-MAX, e.g., in [2]. The inactive value of a causal variable e is indexed as e^0, and its active values are indexed arbitrarily. In practice, some orders of indexing on active values are preferred over others. However, the semantics of NAT models does not dictate choice on such orders.
In general, we denote an effect by e and the set of all causes of e by $C = \{c_1,...,c_n\}$. The domain of e is $D_e = \{e^0,...,e^\eta\}$ $(\eta > 0)$ and the domain of c_i $(i = 1,...,n)$ is $D_i = \{c_i^0,...,c_i^{m_i}\}$ $(m_i > 0)$. An active value may be written as e^+ or c_i^+.
A causal event is a *success* or *failure* depending on whether e is rendered active at a certain range of values, is *single-causal* or *multi-causal* depending on the number of active causes, and is *simple* or *congregate* depending on the range of effect values.
A *simple single-causal success* is an event that cause c_i of value c_i^j $(j > 0)$ caused effect e to occur at value e^k $(k > 0)$, when every other cause is inactive. Denote the event probability by $P(e^k \leftarrow c_i^j) = P(e^k|c_i^j, c_z^0 : \forall z \neq i)$ $(j > 0)$. A multi-causal success involves a set $X = \{c_1,...,c_q\}$ $(q > 1)$ of active causes, where each $c_i \in X$ has the value c_i^j $(j > 0)$, when every other cause $c_m \in C \setminus X$ is inactive. A *congregate multi-causal success* is an event such that causes in X collectively caused the effect to occur at value e^k $(k > 0)$ or values of higher indexes, when every other cause is inactive. We denote the probability of the event as

$$P(e \geq e^k \leftarrow c_1^{j_1},...,c_q^{j_q}) = P(e \geq e^k|c_1^{j_1},...,c_q^{j_q}, c_z^0 : c_z \in C \setminus X) \;\; (j > 0),$$

where $X = \{c_1,...,c_q\}$ $(q > 1)$. It can also be denoted $P(e \geq e^k \leftarrow \underline{x}^+)$.
A *congregate single-causal failure* refers to an event where $e < e^k$ $(k > 0)$ when cause c_i has value c_i^j $(j > 0)$ and every other cause is inactive. We denote the probability of the event as $P(e < e^k \leftarrow c_i^j) = P(e < e^k|c_i^j, c_z^0 : \forall z \neq i) \;\; (j > 0)$.
A NAT consists of two types of NIN-AND gates, each over disjoint sets of causes $W_1,...,W_q$. An input event of a *direct* gate (Fig. 1 (a)) is $e \geq e^k \leftarrow \underline{w}_i^+$ and the output event is $e \geq e^k \leftarrow \underline{w}_1^+,...,\underline{w}_q^+$. An input of a *dual* gate (Fig. 1 (b)) is $e < e^k \leftarrow \underline{w}_i^+$ and the output event is $e < e^k \leftarrow \underline{w}_1^+,...,\underline{w}_q^+$. The probability of the output event of a gate is the product of probabilities of its input events.
Interactions among causes may be reinforcing or undermining, either between causes or groups of causes, as specified in Def. 2.



$e \geq e^k \leftarrow c_1^{j_1}$
$e \geq e^k \leftarrow c_n^{j_n}$
$e \geq e^k \leftarrow c_1^{j_1}, \ldots, c_n^{j_n}$
(a)
$e < e^k \leftarrow c_1^{j_1}$
$e < e^k \leftarrow c_n^{j_n}$
$e < e^k \leftarrow c_1^{j_1}, \ldots, c_n^{j_n}$
(b)
$e < e^1 \leftarrow h_1^1$
$e < e^1 \leftarrow h_2^1$
$e < e^1 \leftarrow b_1^1$
$e < e^1 \leftarrow b_2^1$
g_2
g_3
g_1
$e < e^1 \leftarrow h_1^1, h_2^1$
$e < e^1 \leftarrow b_1^1, b_2^1$
$e \geq e^1 \leftarrow h_1^1, h_2^1, b_1^1, b_2^1$
(c)

Fig. 1. A direct NIN-AND gate (a), a dual NIN-AND gate (b), and a NAT (c)

Definition 2 *[12] Let e^k be an active effect value, $R = \{W_1, \ldots\}$ be a partition of a set $X \subseteq C$ of causes, $R' \subset R$, and $Y = \cup_{W_i \in R'} W_i$. Sets of causes in R* **reinforce** *each other relative to e^k, iff $\forall R'$ $P(e \geq e^k \leftarrow \underline{y}^+) \leq P(e \geq e^k \leftarrow \underline{x}^+)$. They* **undermine** *each other relative to e^k, iff $\forall R'$ $P(e \geq e^k \leftarrow \underline{y}^+) > P(e \geq e^k \leftarrow \underline{x}^+)$.*

Direct gates model undermining and dual gates model reinforcing. A NAT organizes multiple gates into a tree and expresses mixtures of reinforcing and undermining recursively, as illustrated in Fig. 1 (c). A NAT specifies the interaction between each pair of c_i and c_j, denoted by the *PCI bit* $pci(c_i, c_j) \in \{u, r\}$, where u stands for undermining and r for reinforcing. The collection of PCI bits is the *PCI pattern* of the NAT. The PCI pattern for the NAT in Fig. 1 (c) is $\{pci(h_1,h_2) = r, pci(h_1,b_1) = u, pci(h_1,b_2) = u, pci(h_2,b_1) = u, pci(h_2,b_2) = u, pci(b_1,b_2) = r\}$. A NAT can be uniquely identified by its PCI pattern [16].
From the NAT in Fig. 1 (c) and probabilities of its input events, in the general form $P(e^k \leftarrow c_i^j)$ $(j,k > 0)$, called *single-causals*, $P(e \geq e^1 \leftarrow h_1^1, h_2^1, b_1^1, b_2^1)$ can be obtained. From the single-causals and all derivable NATs, the NAT CPT $P(e|h_1,h_2,b_1,b_2)$ is uniquely defined [12]. A NAT model for $|C| = n$ is specified by a NAT topology and a set of single-causals of space linear on n.
A general CPT over C and e has space exponential on n. The complexity is reduced to being linear by compressing the CPT into a NAT model, with the following main steps. (1) Extract one or more PCI patterns from the target CPT [13]. (2) Retrieve NAT structures that are compatible with the PCI patterns [15]. (3) Search for numerical parameters for each NAT structure and return the NAT structure and its parameters that best approximate the target CPT [13].

3 Parameterizing NAT Models by Swarm Descent

Parameterization (the step (3) above) is a time consuming compression step (see experiment at end of section). This section presents a novel method to improve its efficiency. To clarify key issues, we present the existing method [13] algorithmically so that the enhancement can be elaborated.
The input includes a target CPT $P_T(e|C)$ and a set Ψ of candidate NATs over C and e. The output is a NAT $Y \in \Psi$ and a set SC of single-causals associated with Y. The criterion is to select the NAT model $M = (Y, SC)$ such that its CPT P_M best approximates P_T. The difference of P_M from P_T to be minimized is average Kullback−Leibler divergence,

$$KL(P_T, P_M) = \frac{1}{Z} \sum_{i=0}^{Z-1} \sum_j P_T(i,j) log \frac{P_T(i,j)}{P_M(i,j)},$$

where i indexes conditional probability distributions (CPDs) in P_T, j indexes probabilities in each CPD, and Z counts the CPDs. This is achieved by constrained gradient descent described below. In the experimental study, average

Euclidean distance

$$ED(P_T, P_M) = \sqrt{\frac{1}{K}\sum_{i=0}^{Z-1}\sum_{j}(P_T(i,j) - P_M(i,j))^2}$$

is also obtained, where K counts probabilities in P_T.
The gradient g_M is the set of partial derivatives of $KL(P_T, P_M)$ relative to each single-causal $x \in SC$, where $g_M(x) = \partial KL(P_T, P_M)/\partial x$. In a *single-causal descent step*, a single-causal x is revised into $x - \alpha * g_M(x)$, where α is the descent scale (e.g., 0.01). A *descent step* consists of $|SC|$ single-causal descent steps, one per $x \in SC$. During a descent step, each x in the general form $P(e^k \leftarrow c_i^j)$ $(j, k > 1)$ is constrained by $x \in (0,1)$. In addition, for every c_i^j, elements of SC are collectively constrained by $\sum_{k=1}^{\eta} P(e^k \leftarrow c_i^j) < 1$. One *descent round* consists of up to $MaxStep$ (e.g., 200) descent steps, as specified in Algo. 1.

Algorithm 1 $Descent(P_T, Y, SC)$

```
1 for $step = 1$ to $MaxStep$,
2     compute gradient $g_M$ from $P_T$ and $M = (Y, SC)$;
3     if $g_M$ signifies convergence, return SC;
4     for each $x \in SC$, $x = x - \alpha * g_M(x)$, subject to relevant constraints;
5 return SC;
```

Given a NAT Y, the returning SC and the KL distance of NAT model $M = (Y, SC)$ is often dependent on the initial single-causals SC. To get the best KL distance from Y, *MaxRound* (e.g., 10) of descent rounds are performed as Algo. 2.

Algorithm 2 $MultiDescent(P_T, Y)$

```
1 BestSC = null, $BestKL = \infty$;
2 for $round = 1$ to $MaxRound$,
3     randomly initialize a set SC of single-causals;
4     $SC' = Descent(Y, SC)$;
5     compute $KL(P_T, P_M)$, where $M = (Y, SC')$;
6     if $KL(P_T, P_M) < BestKL$, then $BestSC = SC'$, $BestKL = KL(P_T, P_M)$;
7 return $(BestSC, BestKL)$;
```

Parameterization step selects the best NAT from candidates as Algo. 3.

Algorithm 3 $ParameterizeNAT(P_T, \Psi)$

```
1 BestNAT = null, BestSC = null, $BestKL = \infty$;
2 for each NAT $Y \in \Psi$,
3     $(SC, KL) = MultiDescent(P_T, Y)$;
4     if $KL < BestKL$, then $BestNAT = Y$, $BestSC = SC$, $BestKL = KL$;
5 return $(BestNAT, BestSC)$;
```

The above method is time consuming (see experiment at end of section). We observe that it conducts descent rounds independently, one for each candidate NAT and each initial SC set. In the following, we apply particle swarm optimization [5] to develop a novel swarm-based constrained gradient descent, where dependency between multiple descent rounds is introduced to improve efficiency.
A *particle* captures a descent round relative to a NAT Y and an initial set *SC* of single-causals. The descent round is divided into multiple *descent intervals*, where each interval involves $IntervalLength$ (e.g., 10) descent steps. The operation of a particle during a descent interval is specified as Algo. 4. It is similar to Algo. 1 but over a smaller number of steps. Instead of returning *SC*, it saves *SC*, a flag, gradient, and KL distance as attributes of the particle, to be used for evaluating the particle performance.

Algorithm 4 $ParticleDescent(P_T, Y, SC)$

```
1  init flag converged = $false$;
2  for $step = 1$ to $IntervalLength$,
3     compute gradient $g_M$ from $P_T$ and $M = (Y, SC)$;
4     if $g_M$ signifies convergence, converged = $true$ and break;
5     for each $x \in SC$, $x = x - \alpha * g_M(x)$, subject to relevant constraints;
6  compute $kl = KL(P_T, P_M)$, where $M = (Y, SC)$;
7  save SC, converged, $g_M$, and kl;
```

A *particle group* is a set of particles that share the same NAT Y. Performances of particles in a group G are evaluated after each *intra-group check period* consisting of a constant of *InGroupCheckPeriod* (e.g., 2) descent intervals. The evaluation is specified in Algo. 5, where each particle $R \in G$ is associated with attributes such as its α value, *converged* flag, gradient, and KL distance at the end of the last interval of the period. These attributes can be accessed by, e.g., *R.converged* and $R.kl$. For each $R.g_M$, we denote the average of its element by $R.\overline{g_M}$.
In Algo. 5 below, each particle $R \in G$ not yet converged is evaluated as *promising*, *average*, or *poor*. A particle is promising, if its average KL distance is better than the group average and it is faster in descending. A particle is poor performing, if its average KL distance is worse than the group average and it is slower in descending. Each promising particle is rendered to descend even faster, and each poor performing particle is halted, where γ and ρ are threshold scales (e.g., ≥ 1).

Algorithm 5 $InGroupCheck(G)$

```
1  $avgKL = (\sum_{R \in G} R.kl)/|G|$;
2  $avgGradient = (\sum_{R \in G} R.\overline{g_M})/|G|$;
3  for each particle $R \in G$, where $R.converged = false$,
4     if $R.kl < \gamma * avgKL$ and $R.\overline{g_M} > avgGradient$, increase $R.\alpha$;
5     else if $R.kl > \rho * avgKL$, $R.\overline{g_M} < avgGradient$, and $|G| > 1$,
6        remove R from G;
7  return G;
```

Each candidate NAT is processed by exactly one particle group. Performances of all particle groups are evaluated after each *inter-group check period* consisting of a constant of *InterGroupCheckPeriod* (e.g., 4) descent intervals. As the outcome of evaluation, poorly performing groups are terminated. The evaluation is specified in Algo. 6, where Φ denotes the set of all particle groups, and β is a threshold scale (e.g., 1.25). Lines 1 to 4 determine average group performance, and the remainder uses it to identify poorly performing groups.

Algorithm 6 *InterGroupCheck*(Φ)

```
1  $avgKL = 0$;
2  for each particle group $G \in \Phi$,
3      $G.bestKL = \min_{R \in G} R.kl$,   $avgKL = avgKL + G.bestKL$;
4  $avgKL = avgKL/|\Phi|$;
5  for each particle group $G \in \Phi$,
6      if $G.bestKL > \beta * avgKL$ and $|\Phi| > 1$, remove $G$ from $\Psi$;
7  return $\Phi$;
```

Algo. 7 is the top level algorithm for swarm-based descent. *GroupSize* defines the initial number of particles per particle group. It serves the role equivalent to *MaxRound* in Algo. 2. *MaxInterval* (e.g., 10) controls the total number of descent intervals. Lines 1 to 5 initialize all particle groups. Particles descend in lines 6 to 11, subject to intra-group and inter-group evaluations and eliminations. The remainder selects the best NAT model emerging from the descent.

Algorithm 7 *ParameterizeBySwarm*(P_T, Ψ)

```
1  create a set $\Phi$ of $|\Psi|$ particle groups;
2  for each group $G \in \Phi$,
3      assign a distinct NAT $Y \in \Psi$ to $G$ as $G.Y$;
4      create GroupSize particles in $G$;
5      for each particle $R \in G$, init $R.\alpha$ and a set of single-causals as $R.SC$;
6  for $interval = 1$ to $MaxInterval$,
7      for each group $G \in \Phi$,
8          for each particle $R \in G$, $R$ runs $ParticleDescent(P_T, G.Y, R.SC)$;
9      if (interval mod InGroupCheckPeriod) = 0,
10         for each group $G \in \Phi$, $G = InGroupCheck(G)$;
11     if $(interval \bmod InterGroupCheckPeriod) = 0$, $\Phi = InterGroupCheck(\Phi)$;
12 BestNAT = null, BestSC = null, $BestKL = \infty$;
13 for each group $G \in \Phi$,
14     for each particle $R \in G$,
15         if $R.kl < BestKL$, then $BestNAT = G.Y$, $BestSC = R.SC$, $BestKL = R.kl$;
16 return $(BestNAT, BestSC)$;
```

We observe that although *ParticleDescent* by multiple particles are executed sequentially, they are independent until intra-group evaluations. To further improve efficiency, we also investigated a parallel version of swarm-based descent, where *ParticleDescent* by each particle is run on a separate thread. The primary difference from Algo. 7 is to execute line 8 by threads. All particle threads must also be completed before line 9 is executed.

The three alternative methods for parameterization, referred to as CGD (constrained gradient descent), SSD (sequential swarm-based descent), and PSD (parallel swarm-based descent), are evaluated experimentally. Five batches of target CPTs are randomly generated with the number of causes $n = 5, 6, 7, 8, 9$, respectively. Each batch consists of 2 groups with the variable domain size bounded at $s = 3, 4$, respectively. Each group consists of 50 CPTs. Hence, the experiment consists of a total of $5 * 2 * 50 = 500$ target CPTs. Each CPT is run by each of the 3 methods, using a 6-core desktop at 3.7GHz clock speed.

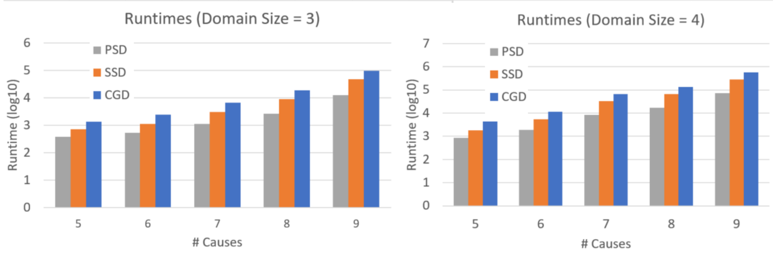


Fig. 2. Runtimes (msec) of PSD, SSD, and CGD in Log10

Fig. 2 summarizes runtime. For each method, as n grows from 5 to 9, computation cost grows by about 100 times. Change of domain sizes from 3 to 4 only increases the computational cost, but does not affect the relative efficiency of the three methods. PSD is consistently more efficient than SSD, and SSD is consistently more efficient than CGD. Furthermore, as n grows from 5 to 9, the advantage of PSD over SSD and that of SSD over CGD becomes more pronounced. At $n = 9$, PSD is about 3.1 times faster than SSD and SSD is about 3.1 times faster than CGD. This results in PSD being about one order of magnitude faster than CGD.

Fig. 3 compares the average KL distance of compressed CPTs from target CPTs by CGD and SSD (PSD has the same KL distance as SSD). Fig. 4 compares the average ED distance. As n grows from 5 to 9, the approximation errors tend to increase, but the average ED distances are between 0.23 and 0.32.

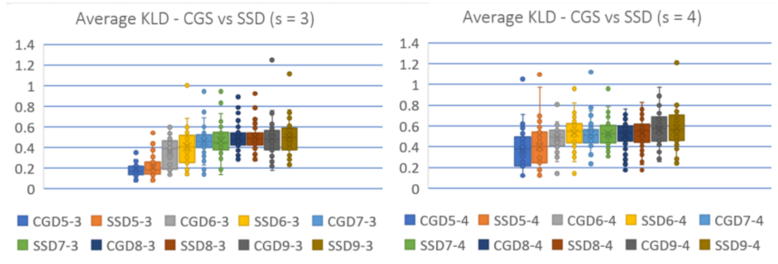


Fig. 3. Average KL distance between target and NAT CPTs from CGD and SSD

Errors by SSD are similar to those of CGD, and are more often (but not always) slightly larger. The slightly large errors can be attributed to early termination of some particles or particle groups. Although they do not perform well in the early descent rounds (and hence are terminated by SSD), they could lead to more accurate approximations later on if allowed to continue. Hence, the swarm-based descent should be viewed as a good heuristic and trade-off of slight accuracy for efficiency, rather than as a dominate strategy.

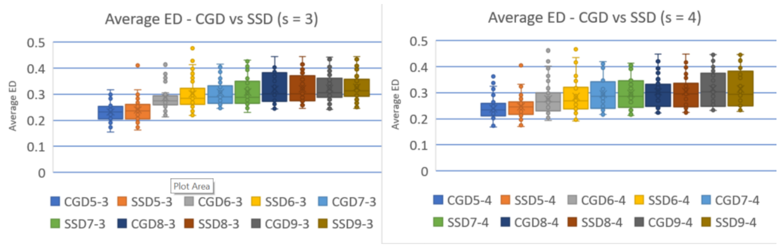


Fig. 4. Average ED distance between target and NAT CPTs from CGD and SSD

4 Impact of MF of NAT Models on Inference Efficiency

In this section, we investigate the impact of NAT modeling on efficiency of inference with BNs. We consider BNs where the CPT of every non-root variable of 2 or more parents is a NAT model, which is referred to as a *NAT-modeled* BN. For the inference method, we focus on lazy propagation (LP) [8]. In order to perform LP with NAT-modeled BNs, we compile them through multiplicative factorization (MF) [14]. In particular, the NAT model of each CPT is structured into a hybrid graph, where link potentials for undirected links and family potentials for directed families are assigned based on parameters of the NAT model. As a result, the NAT-modeled BN is converted into a Markov network and then compiled into a lazy junction tree (JT) for LP. We refer to BNs compiled as so as MF of NAT-modeled BNs (MF-BN).

For comparison, we created a *peer* BN (PR-BN) for each NAT-modeled BN, where each NAT model is expanded into a tabular CPT. The peer BN is then compiled into a lazy JT for LP. In an earlier work, it has been shown that for very sparse BNs (5% more links than singly connected), MF-BNs support up to 2 orders of magnitude speedup in LP than peer BNs [14]. The question of our interest is whether the speedup in LP extends to less sparse NAT-modeled BNs. We provide positive empirical evidence below.

We simulated NAT-modeled BNs with 100 variables per BN. The maximum number of parents per variable in each BN is bounded at $n = 8, 10, 12$, respectively. The uniform domain size of all variables is controlled at $m = 3, 4$, respectively. The structural density of BNs is controlled by adding $w = 5, 10, 15, 20, 25, 30, 35, 40, 45, 50\%$ of links to a singly connected network, respectively. Hence, there are a total of $3 \times 2 \times 10 = 60$ distinct (n, m, w) combinations. For each combination, we simulated 10 BNs. This amounts to a total of 600 NAT-modeled BNs.

For each NAT-modeled BN, we perform LP with its MF-BN and PR-BN, and compare runtimes from a laptop of 2.8 GHz clock speed. Fig. 5 shows MF-BN runtimes, that grow significantly as BN structures become denser (we omit analysis based on tree-width growth due to space). On the other hand, little growth is observed as n grows from 8 to 12. We attribute this to MF of NAT models, which breaks large BN CPTs into smaller MF factors.

Fig. 6 shows ratios of PR-BN runtimes versus MF-BN runtimes. MF-BNs have superior efficiency for very sparse structures ($w = 5\%$), and the superiority grows as n and m grow. This is consistent with findings in [14]. For $n = 10, 12$, MF-BNs run faster than PR-BNs by up to 2 orders of magnitude (984 times faster in one case). The superiority decreases as w grows (denser structures), but persists up to $w = 20\%$ for $n = 10$, and up to $w = 30\%$ for $n = 12$. Hence, the advantage of MF of NAT-modeling extends to a wider range of sparse structures than reported

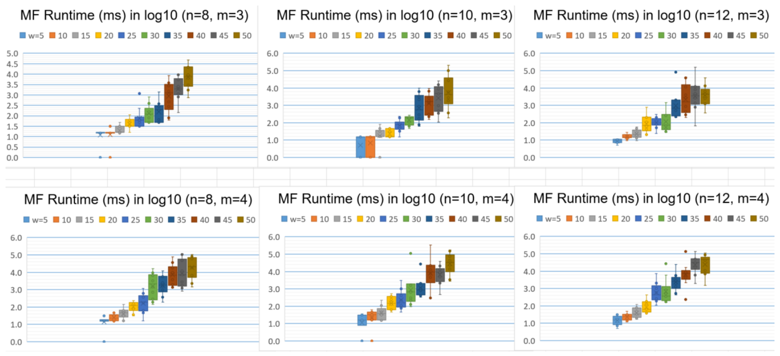


Fig. 5. Runtimes for MF-BNs

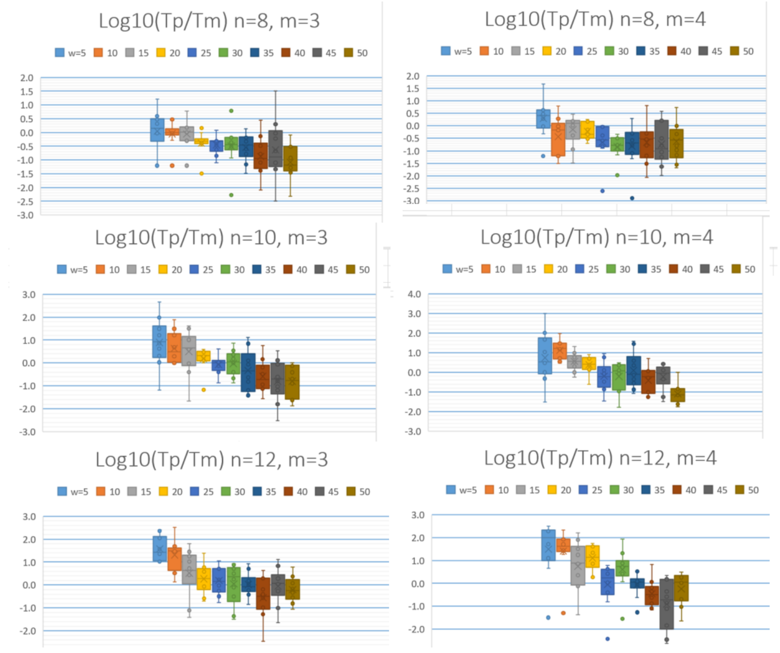


Fig. 6. Ratios of PR-BN runtimes (T_P) versus MF-BN runtimes (T_M)

[14]. Between $w = 25$ and 35% for $n = 10$ and $w = 35$ and 45% for $n = 12$, either MF-BNs or PR-BNs may run faster depending on particular BNs.

5 Posterior Accuracy After NAT-Modeling

CPT compression errors are assessed in Section 3, among others. This section investigates the degree in which they translate into errors of posterior marginals in inference. Eight real world discrete BNs are selected from the well-known *bn-learn* repository. The selection criterion is that the BN must contain a sufficient number of variables whose CPTs are suitable for NAT-modeling. More specifically, the following variables are not suited for NAT-modeling.

1. Root: Its CPT is trivial (not conditional).
2. Single parent: Its tabular CPT is not exponentially spaced.
3. Some CPDs in the CPT are partially deterministic: The causes are not uncertain causes as assumed by NAT models.

4. The variable has a large domain (e.g., of size 20) but a few parents (e.g, 4). As analyzed in [14], MF of NAT models are not more efficient than tabular CPTs for such variables.

Table 1 lists selected BNs. In their NAT-modeled versions, variables unsuited for NAT-modeling keep original CPTs. CPTs of remaining variables in each BN (percentage in 4th column) are NAT-modeled and subject to compression errors. These errors translate into errors in posterior marginals during inference. The objective of the experiment is to access the magnitude of errors in posterior marginals, in relation to CPT compression errors.

Table 1. Real world BNs (Andes$^-$: Andes with 3 isolated nodes removed)

Network	# Nodes	# NAT CPTs	% NAT CPTs	# links	w
Alarm	37	16	43.2	46	28
Andes$^-$	220	106	48.2	338	54
Barley	48	28	58.3	84	79
Child	20	6	30.0	25	32
Hepar2	70	33	47.1	123	78
Insurance	27	8	29.6	52	100
Win95pts	76	8	10.5	112	49
Munin	1041	11	1.1	1397	34

For each BN, we perform LP on the original version (RW-BN) as well as the NAT-modeled version (NT-BN), conditioned on the same observation over 10% of randomly selected variables. For each pair of RW-BN and NT-BN, 10 runs of LP are performed with distinct observations. This amounts to a total of $8*2*10 = 160$ LP runs.

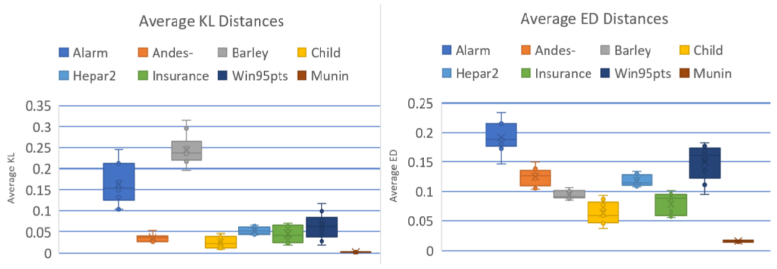


Fig. 7. Average KL and ED distances from LP runs

Average KL and ED distances (Section 3) are shown in Fig. 7. We observe that both distances are reasonably small. Hence, the posterior marginals are reasonably accurate, even though a significant percentage of CPTs in each BN (between 30 and 50% for most BNs) are NAT-modeled. The posterior errors are generally smaller than CPT compression errors (Figs. 3 and 4). That is, CPT compression errors are attenuated, rather than amplified, by the inference.

Fig. 8 (left) summarizes the NT-BN runtime. Ratios of RW-BN runtimes over NT-BN runtimes are summarized in Fig. 8 (right). NT-BNs for Win95pts and Munin run LP faster than RW-BNs, which can be attributed to their lower density levels (*w* in Table 1). For other BNs, their density levels are generally higher than the threshold level observed in Section 4, and their NT-BNs run LP slower than RW-BNs.

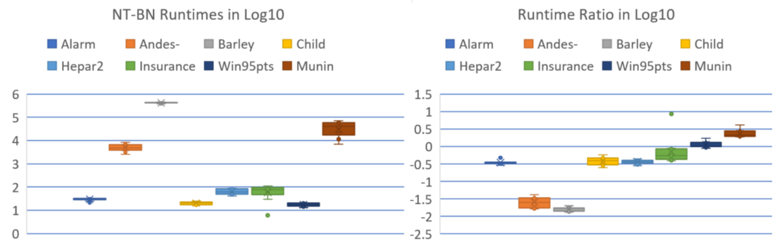


Fig. 8. Left: NT-BN runtimes; Right: Runtime ratios (RW-BN over NT-BN)

In summary, the result demonstrates that NAT-modeling maintains reasonable posterior accuracy in inference (while reducing BN CPT space from being exponential to being linear). As MF of NAT-modeled BNs are more efficient in inference when BN structures are sparse, a promising direction for future work is to learn sparse NAT-modeled BNs directly from data (rather than building a dense BN and then compressing it).

6 Conclusion

Contributions of this work are on local modeling in BNs for more efficient inference, with focus on NAT models. First, we proposed swarm-based constrained gradient descent that allows one order of magnitude faster search and parameterization of NAT-models. Second, our experimental study shows that by multiplicatively factorizing NAT-modeled BNs, LP efficiency can be improved for a range of sparse BN structures. In particular, up to 2 orders of magnitude efficiency gain can be obtained when BN structural densities are up to about 30% more links beyond being singly-connected. Finally, our experimental study of NAT-modeling real world BNs for LP inference demonstrated reasonable accuracy in posterior marginals that tend to be more accurate than CPT compression.

A number of directions for future research can be envisioned. We have NAT-modeled real world BNs by keeping deterministic CPTs. They may be approximated by NAT models or other compact local models, e.g., algebraic decision diagrams [4], and then integrated with NAT models in a same BN. The range of structural densities where MF of NAT modeled BNs show superior LP performance suggests direct learning of NAT modeled BNs constrained to that density range. Finally, MF of NAT models is linear in number of causes but exponential in the effect domain size. Methods other than MF without such limitation may be explored.

Acknowledgement

Financial support from NSERC Discovery Grant to first author is acknowledged.

References

1. C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In E. Horvitz and F. Jensen, editors, *Proc. 12th Conf. on Uncertainty in Artificial Intelligence*, pages 115–123, 1996.
2. F.J. Diez. Parameter adjustment in Bayes networks: The generalized noisy OR-gate. In D. Heckerman and A. Mamdani, editors, *Proc. 9th Conf. on Uncertainty in Artificial Intelligence*, pages 99–105. Morgan Kaufmann, 1993.
3. M. Henrion. Some practical issues in constructing belief networks. In L.N. Kanal, T.S. Levitt, and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence 3*, pages 161–173. Elsevier Science Publishers, 1989.
4. J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proc. 15th Conf. on Uncertainty in Artificial Intelligence*, pages 279–288, 1999.
5. J. Kennedy and R.C. Eberhart. Particle swarm optimization. In *Proc. IEEE Inter. Conf. on Neural Networks*, pages 1942–1948, Piscataway, NJ, 1995.
6. J.F. Lemmer and D.E. Gossink. Recursive noisy OR - a rule for estimating complex probabilistic interactions. *IEEE Trans. on System, Man and Cybernetics, Part B*, 34(6):2252–2261, Dec 2004.
7. P.P. Maaskant and M.J. Druzdzel. An independence of causal interactions model for opposing influences. In M. Jaeger and T.D. Nielsen, editors, *Proc. 4th European Workshop on Probabilistic Graphical Models*, pages 185–192, Hirtshals, Denmark, 2008.
8. A.L. Madsen and F.V. Jensen. Lazy propagation: A junction tree inference algorithm based on lazy evaluation. *Artificial Intelligence*, 113(1-2):203–245, 1999.
9. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
10. J. Vomlel and P. Tichavsky. An approximate tensor-based inference method applied to the game of Minesweeper. In *Proc. 7th European Workshop on Probabilistic Graphical Models, Springer LNAI 8745*, pages 535–550, 2012.
11. S. Woudenberg, L.C. van der Gaag, and C. Rademaker. An intercausal cancellation model for Bayesian-network engineering. *Inter. J. Approximate Reasoning*, 63:3247, 2015.
12. Y. Xiang. Non-impeding noisy-AND tree causal models over multi-valued variables. *International J. Approximate Reasoning*, 53(7):988–1002, 2012.
13. Y. Xiang and Q. Jiang. NAT model based compression of Bayesian network CPTs over multi-valued variables. *Computational Intelligence (online DOI: 10.1111/coin.12126; paper version in press)*, 2017.
14. Y. Xiang and Y. Jin. Efficient probabilistic inference in Bayesian networks with multi-valued NIN-AND tree local models. *Int. J. Approximate Reasoning*, 87:67–89, 2017.
15. Y. Xiang and Q. Liu. Compression of Bayesian networks with NIN-AND tree modeling. In L.C. vander Gaag and A.J. Feelders, editors, *Probabilistic Graphical Models, LNAI 8754*, pages 551–566. Springer, 2014.
16. Y. Xiang and M. Truong. Acquisition of causal models for local distributions in Bayesian networks. *IEEE Trans. Cybernetics*, 44(9):1591–1604, 2014.