Workload-aware Materialization for Efficient Variable Elimination on Bayesian Networks

Cigdem Aslay
Aarhus University
Aarhus, Denmark
cigdem@cs.au.dk

Martino Ciaperoni
Aalto University
Espoo, Finland
martino.ciaperoni@aalto.fi

Aristides Gionis
KTH Royal Institute of Technology
Stockholm, Sweden
argioni@kth.se

Michael Mathioudakis
University of Helsinki
Helsinki, Finland
michael.mathioudakis@helsinki.fi

***Abstract*—Bayesian networks are general, well-studied probabilistic models that capture dependencies among a set of variables. *Variable Elimination* is a fundamental algorithm for probabilistic inference over Bayesian networks. In this paper, we propose a novel materialization method, which can lead to significant efficiency gains when processing inference queries using the Variable Elimination algorithm. In particular, we address the problem of choosing a set of intermediate results to precompute and materialize, so as to maximize the expected efficiency gain over a given query workload. For the problem we consider, we provide an optimal polynomial-time algorithm and discuss alternative methods. We validate our technique using real-world Bayesian networks. Our experimental results confirm that a modest amount of materialization can lead to significant improvements in the running time of queries, with an average gain of 70%, and reaching up to a gain of 99%, for a uniform workload of queries. Moreover, in comparison with existing junction tree methods that also rely on materialization, our approach achieves competitive efficiency during inference using significantly lighter materialization.**

***Index Terms*—probabilistic inference, materialization**

I. Introduction

Research in *machine learning* has led to powerful methods for building probabilistic models which are used to carry out general predictive tasks [1]. In a typical setting, a model is trained offline from available data and it is subsequently employed online for processing probabilistic inference queries. For example, a joint-probability model trained offline over the attributes of a relational database can be employed by the DBMS at query time to produce selectivity estimates for each query, for the purpose of query optimization [2]. For efficiency in such a setting, it is important to consider computational costs not only for training the model, but also for performing inference at query time. In this paper, we focus on the efficiency of inference in Bayesian networks.

Bayesian networks are probabilistic models that capture the joint distribution of a set of variables through local dependencies between small groups of variables [3]. They have an intuitive representation in terms of a directed acyclic graph (each directed edge between two variables represents a dependency of the child on the parent variable) and probabilities can be evaluated with compact sum-of-products computations. Being intuitive and modular makes Bayesian networks suitable for general settings where one wishes to represent the joint distribution of a large number of variables, e.g., the column attributes of a large relational table as the one shown in Fig. 1. Nevertheless, inference in Bayesian networks is $\mathbf{NP}$-hard and one cannot preclude the possibility that the evaluation of some queries becomes expensive for cases of interest.

How can we mitigate this risk? Our approach is to consider the anticipated workload of probabilistic queries, and materialize the results of selected computations, so as to enable fast answers to costly inference queries. Moreover, we choose to work with a specific inference algorithm for Bayesian networks, namely variable elimination [5], [6]. While there exist several inference algorithms for Bayesian networks (e.g., junction trees, arithmetic circuits, and others [7]–[11]), variable elimination is arguably the most straightforward algorithm for inference with Bayesian networks and is used as subroutine in other algorithms (e.g., junction trees [7]). Since, to the best of our knowledge, this is the first work on *workload-aware materialization* for inference in Bayesian networks, we opt to use variable elimination as the inference algorithm, and demonstrate results to motivate the same approach for other inference algorithms as future work.

Our key observations are the following: first, the execution of variable elimination involves intermediate results, in the form of relational tables, some of which might be very costly to compute every time a query requires them; second, the same intermediate tables are used for the evaluation of many different queries. Therefore, we set to precompute and materialize the intermediate tables that bring the largest computational benefit, i.e., those that are involved in the evaluation of many expensive queries. The problem formulation is general enough to accommodate arbitrary query workloads and takes as input a budget constraint on the number or space requirements of the materialized tables (Section III).

Our contributions are the following:

- An exact polynomial-time algorithm to choose a materialization for the variable elimination algorithm under cardinality constraints; the choice is optimal for a fixed ordering of variables. (Section IV)
- A greedy approximation algorithm. (Section IV)
- A pseudo-polynomial algorithm for the problem under a budget constraint on the space used for materialization and an extension to deal with redundant variables. (Section V)
- Experiments over real data, showing that modest mate-

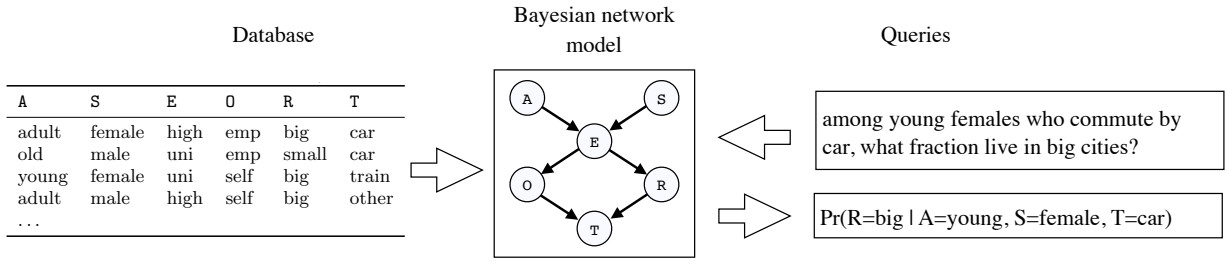


Fig. 1. The Bayesian network learned from the `survey` dataset [4]. The database consists of a single table with variables. `A` (age) , `S` (sex) , `E` (education level) , `O` (occupation) , `R` (size of residence city) , and `T` (means of transportation) The corresponding Bayesian network model can be used to answer probabilistic queries over the data, such as the query shown above.

rialization can significantly improve the running time of variable elimination, with an average gain of 70%, and reaching up to a gain of 99%, over a uniform workload of queries. In addition, for large datasets and query sizes, our approach achieves competitive inference speeds using much lighter materialization compared with junction tree methods, which also rely on materialization. (Section VI)

Proofs of lemmas and theorems, omitted due to page limit, can be found in an extended version of the paper [12].

II. Related work

For exact inference on Bayesian networks, i.e., exact computation of marginal and conditional probabilities, the conceptually simplest algorithm is VARIABLE ELIMINATION [5], [6]. Another common algorithm for exact inference is the *junction tree* algorithm [7], [8]. Obtaining the junction tree of a Bayesian network requires building the triangulated moral graph of the Bayesian network, extracting the maximal cliques of that graph, and assigning each clique to one node of the junction tree. The junction tree is then "calibrated" via a message-passing procedure that precomputes and materializes potentials that correspond to joint probability distributions of the variables belonging to each of its nodes. This way, a query that involves variables in the same node can be answered directly by marginalizing those variables from the potential of that node. For queries whose query variables reside in more than one node, referred to as "out-of-clique" queries, additional message passing is performed between the nodes containing the query variables. To improve the efficiency of handling out-of-clique queries, Kanagal and Deshpande [13] proposed to materialize additional probability distributions arising from a hierarchical partitioning of the calibrated junction tree. We compare experimentally our proposal with the aforementioned junction tree algorithms [8], [13].

Additionally, there have been several data structures proposed for efficient exact inference with graphical models in the knowledge-compilation literature [9]–[11], [13]–[15]. Darwiche [9] showed that every Bayesian network can be interpreted as an exponentially-sized multi-linear function whose evaluation and differentiation solves the exact inference problem. Accordingly, several techniques were proposed to efficiently factor these multi-linear functions into more compact representations, such as arithmetic circuits and sum-product networks, by taking advantage of possible sparsity in conditional probabilities and context-specific independence for a given query [10], [11], [15]. We note that optimizing for a given workload is orthogonal to these efforts: our cost-benefit framework could be extended to account for other concise representations of the conditional-probability tables used by inference algorithms. As this paper is the first work to address budget-constrained and workload-aware materialization for query evaluation on Bayesian networks, we opt to work with the variable-elimination algorithm [5] over tabular-factor representations due to its conceptual simplicity.

Model-based inference finds application in many settings, including tasks related to data management. For example, models like Bayesian networks can naturally be used for selectivity estimation, as explained by Getoor et al. [2] and Tzoumas et al. [16], and today there is renewed interest in approximate query processing (AQP) based on probabilistic approaches [17]–[21]. For tasks such as selectivity estimation and AQP, where inference queries are repeatedly executed over a learned model, it is important to consider the efficiency of online inference and the role that careful materialization can play to increase efficiency. Our work can be viewed as a step in this direction. Note, however, that we focus on the efficiency of VARIABLE ELIMINATION for Bayesian network inference and not on any specific application such as AQP.

III. Setting and problem statement

In this section we formally define the materialization problem we study. We first provide an overview of Bayesian-network inference and VARIABLE ELIMINATION algorithm, which is central to our contribution.

A Bayesian network $\mathcal{N}$ is a directed acyclic graph (DAG), where nodes represent variables and edges represent dependencies among variables. Each node is associated with a table quantifying the probability that the node takes a particular value, conditioned on the values of its parent nodes. For instance, if a node associated with a ν-ary variable a has ℓ parents, all of which are ν-ary variables, the associated probability distribution for a is a table with $\nu^{\ell+1}$ entries.

A key property of Bayesian networks is that each variable is conditionally independent of all its non-descendants given

the value of its parents. Due to this property, the joint probability of all the variables can be factorized into marginal and conditional probabilities associated with the nodes of the network. Using the network of Fig. 1 as our running example, the joint probability of all the variables is given by

$$\begin{aligned}\Pr(\mathtt{A},\mathtt{S},\mathtt{E},\mathtt{O},\mathtt{R},\mathtt{T}) = \quad & \Pr(\mathtt{T} \mid \mathtt{O},\mathtt{R}) \Pr(\mathtt{O} \mid \mathtt{E}) \Pr(\mathtt{R} \mid \mathtt{E}) \\ & \Pr(\mathtt{E} \mid \mathtt{A},\mathtt{S}) \Pr(\mathtt{A}) \Pr(\mathtt{S}).\end{aligned} \tag{1}$$

Each factor on the right-hand side of (1) is part of the specification of the Bayesian network and represents the marginal and conditional probabilities of its variables.

In what follows, we assume we are given a Bayesian network $\mathcal{N}$ that is discrete, i.e., all variables are categorical; numerical variables can be discretized in categorical intervals.

Querying the Bayesian network. We consider the task of answering probabilistic queries over a Bayesian network $\mathcal{N}$. Specifically, we consider queries of the form

$$q = \Pr(X_q, Y_q = \mathbf{y}_q), \tag{2}$$

where $X_q \subseteq X$ is a set of free variables and $Y_q \subseteq X$ is a set of bound variables with corresponding values $\mathbf{y}_q$. Notice that free variables X_q indicate that the query requests a probability for *each* of their possible values. In the example of Fig. 1, $\Pr(\mathtt{S} = \text{female}, \mathtt{E} = \text{uni})$ and $\Pr(\mathtt{T}, \mathtt{A} = \text{young})$ are instances of such queries.

We denote by $Z_q = X \setminus (X_q \cup Y_q)$ the set of variables that do not appear in the query q. The variables in the set Z_q are those that need to be *summed out* in order to compute the query q. Specifically, query q is computed via the summation

$$\Pr(X_q, Y_q = \mathbf{y}_q) = \sum_{Z_q} \Pr(X_q, Y_q = \mathbf{y}_q, Z_q)\,. \tag{3}$$

The answer to the query $\Pr(X_q, Y_q = \mathbf{y}_q)$ is a table indexed by the combinations of values of variables in X_q. Note that conditional probabilities of the form $\Pr(X_q \mid Y_q = \mathbf{y}_q)$ can be computed from the corresponding joint probabilities by

$$\Pr(X_q \mid Y_q = \mathbf{y}_q) = \frac{\Pr(X_q, Y_q = \mathbf{y}_q)}{\sum_{X_q} \Pr(X_q, Y_q = \mathbf{y}_q)}. \tag{4}$$

In addition, for variables with discrete numerical domain, *range-queries* of the form $\Pr(X_q, \mathbf{y}_\ell \leq Y_q \leq \mathbf{y}_u)$ can be computed from the corresponding joint probabilities by

$$\Pr(X_q, \mathbf{y}_\ell \leq Y_q \leq \mathbf{y}_u) = \sum_{\mathbf{y}_\ell \leq \mathbf{y}_q \leq \mathbf{y}_u} \Pr(X_q, Y_q = \mathbf{y}_q), \tag{5}$$

where '$\leq$' here denotes an element-wise operator over variable vectors. So without loss of generality, we focus on queries of type (2).

Answering queries. A query q can be computed by *brute-force* approach as follows. First, compute the joint probability of all the variables in the network via a natural join over the tables of the factors of (1). Let H be the resulting table, indexed by the combination of values of all variables. Then, select those entries of H that satisfy the corresponding equality condition $Y_q = \mathbf{y}_q$. Finally, for each $a \in Z_q$, compute a sum over each *group* of values of a (i.e., "sum out" a). The table that results from this process is the answer to query q.

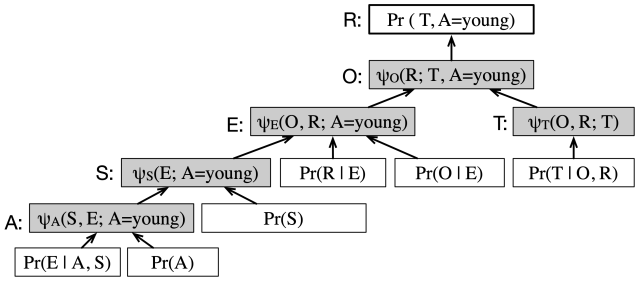


Fig. 2. The elimination tree T for the query $q = \Pr(\mathtt{T}, \mathtt{A} = \text{young})$ and order of variables $\sigma = (\mathtt{A}, \mathtt{S}, \mathtt{T}, \mathtt{E}, \mathtt{O}, \mathtt{R})$, for the dataset and Bayesian network shown in Fig. 1.

Variable elimination. The VARIABLE ELIMINATION algorithm proposed by Zhang et al. [5] improves upon the brute-force approach by taking advantage of the factorization of the joint probability, hence, avoiding the computation of H. Given a total order σ on variables, the VARIABLE ELIMINATION algorithm computes the quary q by summing out the variables in Z_q, one at a time, from the intermediate tables obtained via natural join of the *relevant* factors.

For example, consider the Bayesian network of Fig. 1, the query $q = \Pr(\mathtt{T}, \mathtt{A} = \text{young})$, and the order $\sigma = \langle \mathtt{A}, \mathtt{S}, \mathtt{T}, \mathtt{E}, \mathtt{O}, \mathtt{R} \rangle$. Note that for this query q, we have $X_q = \{\mathtt{T}\}$, $Y_q = \{\mathtt{A}\}$, and $Z_q = \{\mathtt{S}, \mathtt{E}, \mathtt{O}, \mathtt{R}\}$. The first variable in σ is $\mathtt{A} \in Y_q$. The VARIABLE ELIMINATION algorithm considers only the tables of those factors that include variable $\mathtt{A}$ and select the rows that satisfy the equality condition $(\mathtt{A} = \text{young})$; then it performs the natural join over the resulting tables. This computation corresponds to the following two equations:

$$\psi_{\mathtt{A}}(\mathtt{S}, \mathtt{E}; \mathtt{A} = \text{young}) = \Pr(\mathtt{E} \mid \mathtt{A} = \text{young}, \mathtt{S}) \Pr(\mathtt{A} = \text{young}),$$

$$\begin{aligned}\Pr(\mathtt{A} = \text{young}, \mathtt{S}, \mathtt{E}, \mathtt{O}, \mathtt{R}, \mathtt{T}) = \ & \psi_{\mathtt{A}}(\mathtt{S}, \mathtt{E}; \mathtt{A} = \text{young}) \Pr(\mathtt{S}) \\ & \Pr(\mathtt{T} \mid \mathtt{O}, \mathtt{R}) \Pr(\mathtt{O} \mid \mathtt{E}) \Pr(\mathtt{R} \mid \mathtt{E}).\end{aligned}$$

Note that the computation involves only two factors that contain the variable $\mathtt{A}$, resulting in a table $\psi_{\mathtt{A}}(\mathtt{S}, \mathtt{E}; \mathtt{A} = \text{young})$, which is indexed by $\mathtt{S}$ and $\mathtt{E}$, and replaces these factors in the factorization of the joint probability.

The algorithm then proceeds with the next variables in the elimination order σ, each time considering the current set of relevant factors in the factorization, until there are no more variables left to eliminate. The answer to the query is given by the final remaining factor upon all the variables in σ are processed this way.

Elimination tree. The VARIABLE ELIMINATION algorithm gives rise to a graph, like the one shown in Fig. 2 for the example of Fig. 1. Each node is associated with a factor and there is a directed edge between two factors if one is used for the computation of the other. In particular, each *leaf node* corresponds a factor, which is a conditional probability table in the Bayesian network. In our running example, these are the

factors that appear in (1). Each *internal node* corresponds to a factor that is computed from its children (i.e., the factors of its incoming edges), and is computed during the execution of the VARIABLE ELIMINATION algorithm. Moreover, each internal node corresponds to one variable. The last factor computed is the answer to the query.

Notice that the graph constructed in this manner is either a tree or a forest. It is not difficult to see that *the elimination graph is a tree if and only if the corresponding Bayesian network is a weakly connected* DAG. To simplify our discussion, we will focus on connected Bayesian networks and deal with an *elimination tree* T for each query. All results can be directly extended to the case of forests.

We note that the exact form of the factor for each internal node of T depends on the query. For the elimination tree in Fig. 2, we have factor $\psi_{\mathtt{A}}(\mathtt{S},\mathtt{E};\mathtt{A}=\text{young})$ on the node that corresponds to variable A. However, if the query contained variable A as a free variable rather than bound to value $(\mathtt{A}=\text{young})$, then the same node in T would contain a factor $\psi_{\mathtt{A}}(\mathtt{S},\mathtt{E};\mathtt{A})$. And if the query did not contain variable A, then the same node in T would contain a factor $\psi_{\mathtt{A}}(\mathtt{S},\mathtt{E})$. On the other hand, the structure of the tree, the factors that correspond to leaf nodes and the variables that index the variables of the factors that correspond to internal nodes are query independent.

Materialization of factors. Materializing factor tables for internal nodes of the elimination tree T can speed up the computation of queries that require those factors. As we saw earlier, factors are computed in a sequence of steps, one for each variable. Each step involves the natural join over other factor tables, followed by: (i) either variable summation (to sum-out variables Z_q); or (ii) row selection (for variables Y_q); or (iii) no operation (for variables X_q). In what follows, *we focus on materializing factors that involve only variable summation*, the first out of these three types of operations. Materializing such factors is often useful for multiple queries q and sufficient to make the case for the materialization of factors that lead to the highest performance gains over a given query workload. Dealing with the materialization of general factors is a straightforward but somewhat tedious extension which is left for future work.

To formalize our discussion, we introduce some notation. Given a node $u \in V$ in an elimination tree T, we write T_u to denote the subtree of T that is rooted at node u. We also write X_u to denote the subset of variables of X that are associated with the nodes of T_u. Finally, we write A_u to denote the set of ancestors of u in T, that is, all nodes between u and the root of the tree T, excluding u.

Computing a factor for a query q incurs a computational cost. We distinguish two notions of cost: first, if the children factors of a node u in the elimination tree T are given as input, computing u incurs a *partial* cost of computing the factor of u from its children; second, starting from the factors associated with conditional probability tables in the Bayesian network, the *total* cost of computing a node includes the partial costs of computing all intermediate factors, from the *leaf* nodes to u. Formally, we have the following definitions.

Definition 1 (Partial-Cost)**.** *The partial cost $c(u)$ of a node $u \in V$ in the elimination tree $T = (V, E)$ is the computational cost required to compute the corresponding factor given the factors of its children nodes.*

Definition 2 (Total-Cost)**.** *The total cost of a node $u \in V$ in the elimination tree $T = (V, E)$ is the total cost of computing the factor at node u, i.e.,*

$$b(u) = \sum_{x \in T_u} c(x)\,, \tag{6}$$

where $c(x)$ is the partial cost of node x.

When we say that we materialize a node $u \in V$, we mean that we precompute and store the factor that is the result of summing out all variables below it on T. *When is a materialized factor useful for a query q?* Intuitively, it is useful if it is one of the factors computed during the evaluation of q, in which case we save the total cost of computing it from scratch, provided that there is no other materialized factor that could be used in its place, with greater savings in cost. The following definition of usefulness formalizes this intuition.

Definition 3 (Usefulness)**.** *Let $q = \Pr(X_q, Y_q = \mathbf{y}_q)$ be a query, and $R \subseteq V$ a set of nodes of the elimination tree T that are materialized. We say that a node $u \in V$ is useful for the query q with respect to the set of nodes R, if (i) $u \in R$; (ii) $X_u \subseteq Z_q$; and (iii) there is no other node $v \in A_u$ for which conditions (i) and (ii) hold.*

To indicate that a node u is *useful* for the query q with respect to a set of nodes R with materialized factors, we use the indicator function $\delta_q(u; R)$. That is, $\delta_q(u; R) = 1$ if node $u \in V$ is useful for the query q with respect to the set of nodes R, and $\delta_q(u; R) = 0$ otherwise. When a materialized node is useful for a query q, it saves us the total cost of computing it from scratch. Considering a query workload, where different queries appear with different probabilities, we define the benefit of a set of materialized nodes R as the total cost we save in expectation.

Definition 4 (Benefit)**.** *Consider an elimination tree $T = (V, E)$, a set of nodes $R \subseteq V$, and query probabilities $\Pr(q)$ for the set of all possible queries q. The benefit $B(R)$ of the node set R is defined as:*

$$\begin{aligned} B(R) &= \sum_{q} \Pr(q) \sum_{u \in R} \delta_q(u; R)\, b(u) \\ &= \sum_{u \in R} \Pr(\delta_q(u; R) = 1)\, b(u) \\ &= \sum_{u \in R} \mathrm{E}[\delta_q(u; R)]\, b(u)\,. \end{aligned} \tag{7}$$

Problem definition. We can now define formally the problem we consider: for a space budget K, our goal is to select a set of factors to materialize to achieve optimal benefit.

Problem 1. *Given a Bayesian network $\mathcal{N}$, an elimination tree $T = (V, E)$ for answering probability queries over $\mathcal{N}$, and budget K, select a set of nodes $R \subseteq V$ to materialize, whose total size is at most K, so as to optimize $B(R)$.*

For simplicity, we also consider a version of the problem where we are given a total budget k on the number of nodes that we can materialize. We first present algorithms for Problem 2 in Section IV, and discuss how to address the more general Problem 1 in Section V.

Problem 2. *Given a Bayesian network $\mathcal{N}$, an elimination tree $T = (V, E)$ for answering probability queries over $\mathcal{N}$, and an integer k, select at most k nodes $R \subseteq V$ to materialize so as to optimize $B(R)$.*

Note that the factors to materialize are chosen under the assumption that the queries will follow a given probability distribution $\mathcal{P} = \{\Pr(q)\}$. If the queries that are eventually encountered follow a different distribution $\mathcal{P}'$, then query answering may be faster or slower on average than under $\mathcal{P}$; this depends on whether $\mathcal{P}'$ assigns higher or lower probability to queries that are favorable or not for the chosen materialization. Notice, however, that even when the materialization has been optimized for a different workload, the runtime cost of queries can never be worse than the runtime without materialization. The worst case arises when the chosen materialization is useful for no query with non-zero probability in $\mathcal{P}'$. In such a case the benefit of the materialization is zero.

IV. Algorithms

In this section we present our algorithms for Problem 2: Section IV-A presents an exact polynomial-time dynamic-programming algorithm; Section IV-B presents a greedy algorithm, which yields improved time complexity but provides only an approximate solution, yet with quality guarantee.

A. Dynamic programming

We discuss our dynamic-programming algorithm in three steps. First, we introduce the notion of *partial benefit* that allows us to explore partial solutions for the problem. Second, we demonstrate the optimal-substructure property of the problem, and third, we present the algorithm.

Partial benefit. In Definition 4 we defined the (total) *benefit* of a subset of nodes $R \subseteq V$ (i.e., a potential solution) for the whole elimination tree T. Here we define the *partial benefit* of a subset of nodes R for a subtree T_u of a given node u of the elimination tree T.

Definition 5 (partial benefit)**.** *Consider an elimination tree $T = (V, E)$, a subset of nodes $R \subseteq V$, and probabilities $\Pr(q)$ for the set of all possible queries q. The partial benefit $B_u(R)$ of the node set R at a given node $u \in V$ is*

$$B_u(R) = \sum_{v \in R \cap T_u} \mathrm{E}[\delta_q(v; R)]\, b(v)\,. \tag{8}$$

The following lemma states that, given a set of nodes R, and a node $u \in R$, the probability that u is useful for a random query with respect to R depends only on the lowest ancestor of u in R.

Lemma 1. *Consider an elimination tree $T = (V, E)$ and a set $R \subseteq V$ of nodes. Let $u, v \in R$ such that $v \in A_u$ and $path(u, v) \cap R = \emptyset$. Then we have:*

$$\mathrm{E}[\delta_q(u; R)] = \mathrm{E}[\delta_q(u; v)]\,, \tag{9}$$

where the expectation is taken over a distribution of queries q.

Building on Lemma 1, we arrive to Lemma 2 below, which states that the partial benefit $B_u(R)$ of a node-set R at a node u depends only on (i) the nodes of T_u that are included in R, and (ii) the lowest ancestor v of u in R, and therefore it does not depend on what other nodes "above" v are included in R.

Lemma 2. *Consider an elimination tree $T = (V, E)$ and a node $u \in V$. Let $v \in A_u$ be an ancestor of u. Consider two sets of nodes R and R' for which (i) $v \in R$ and $v \in R'$; (ii) $T_u \cap R = T_u \cap R'$; and (iii) $path(u, v) \cap R = path(u, v) \cap R' = \emptyset$. Then, we have: $B_u(R) = B_u(R')$.*

Let ϵ be a special node, which we will use to denote that no ancestor of a node u is included in a solution R. We define $\bar{A}_u = A_u \cup \{\epsilon\}$ as the *extended set of ancestors* of u, which adds ϵ into A_u. Notice that $path(u, \epsilon)$ corresponds to the set of ancestors of u including the root r, i.e., $path(u, \epsilon) = A_u$.

Optimal substructure. Next, we present the optimal-substructure property for Problem 2. Lemma 3 states that the subset of nodes of an optimal solution that fall in a given subtree depends only on the nodes of the subtree and the lowest ancestor of the subtree that is included in the optimal solution.

Lemma 3 (Optimal Substructure)**.** *Given an elimination tree $T = (V, E)$ and an integer budget k, let R^* denote the optimal solution to Problem 2. Consider a node $u \in V$ and let $v \in \bar{A}_u$ be the lowest ancestor of u that is included in R^*. Let $R^*_u = T_u \cap R^*$ denote the set of nodes in the optimal solution that reside in T_u and let $\kappa^*_u = |T_u \cap R^*|$. Then,*

$$R^*_u = \operatorname*{arg\,max}_{\substack{R_u \subseteq T_u \\ |R_u| = \kappa^*_u}} \{B_u(R_u \cup \{v\})\}\,. \tag{10}$$

The following lemma provides a bottom-up approach to combine partial solutions computed on subtrees. We note that in the rest of the section, we present our results on *binary trees*. This assumption is made *without any loss of generality* as any d-ary tree can be converted into a binary tree by introducing dummy nodes; furthermore, by assigning appropriate cost to dummy nodes, we can ensure that they will not be selected by the algorithm.

Lemma 4 (Additivity)**.** *Consider an elimination tree $T = (V, E)$, a node $u \in V$, and a set R_u of nodes in T_u. Let*

$r(u)$ and $\ell(u)$ be the right and left children of u, and let $R_{r(u)} = T_{r(u)} \cap R_u$ and $R_{\ell(u)} = T_{\ell(u)} \cap R_u$. Then, for $v \in \bar{A}_u$:

$$B_u(R_u \cup \{v\}) = \begin{cases} B_u(\{u,v\}) + B_{r(u)}\big(R_{r(u)} \cup \{u\}\big) \\ \qquad + B_{\ell(u)}\big(R_{\ell(u)} \cup \{u\}\big), & \text{if } u \in R_u \\ B_{r(u)}\big(R_{r(u)} \cup \{v\}\big) + B_{\ell(u)}\big(R_{\ell(u)} \cup \{v\}\big), & \text{otherwise.} \end{cases} \tag{11}$$

Dynamic programming. Finally, we discuss how to use the structural properties shown above in order to devise the dynamic-programming algorithm. We first define the data structures that we use. Consider a node u in the elimination tree, a node $v \in \bar{A}_u$, and an integer κ between 1 and $\min\{k, |T_u|\}$. We define $F(u, \kappa, v)$ to be the optimal value of partial benefit $B_u(R)$ over all sets of nodes R that satisfy the following three conditions: (i) $|T_u \cap R| \leq \kappa$; (ii) $v \in R$; and (iii) $path(u, v) \cap R = \emptyset$. Condition (i) states that the node set R has at most κ nodes in the subtree T_u; condition (ii) states that node v is contained in R; and condition (iii) states that no other node between u and v is contained in R, i.e., node v is the lowest ancestor of u in R.

For all u, v, κ, and sets R that satisfy conditions (i)–(iii) we also define $F^+(u, \kappa, v)$ and $F^-(u, \kappa, v)$ to denote the optimal partial benefit $B_u(R)$ for the cases when $u \in R$ and $u \notin R$, respectively. Hence, we have

$$F(u, \kappa, v) = \max\left\{F^+(u, \kappa, v), F^-(u, \kappa, v)\right\}. \tag{12}$$

We assume that the special node ϵ belongs in all solution sets R but does not count towards the size of R. Notice that $F(u, \kappa, \epsilon)$ is the optimal partial benefit $B_u(R)$ for all sets R that have at most κ nodes in T_u and no ancestor of u belongs to R. We now show how to compute $F(u, \kappa, v)$ for all $u \in V$, $\kappa \in \{1, \ldots, \min\{k, |T_u|\}\}$, and $v \in \bar{A}_u$ by a bottom-up dynamic-programming algorithm:

1. If u is a leaf of the elimination tree then
$$\begin{aligned} F^-(u, 1, v) &= 0, \text{ for all } v \in \bar{A}_u, \\ F^+(u, 1, v) &= -\infty, \text{ for all } v \in \bar{A}_u. \end{aligned} \tag{13}$$
This initialization enforces leaf nodes not to be selected, as they correspond to factors that define the Bayesian network and are part of the input.
2. If u is not a leaf of the elimination tree then
$$F^+(u, \kappa, v) = B_u(\{u, v\}) + \max_{\kappa_\ell + \kappa_r = \kappa - 1} \left\{F(\ell(u), \kappa_\ell, u) + F(r(u), \kappa_r, u)\right\}, \tag{14}$$
and
$$F^-(u, \kappa, v) = \max_{\kappa_\ell + \kappa_r = \kappa} \left\{F(\ell(u), \kappa_\ell, v) + F(r(u), \kappa_r, v)\right\}.$$

The value of the optimal solution is returned by $F(r, k, \epsilon) = \max\{F^+(r, k, \epsilon), F^-(r, k, \epsilon)\}$, where r is the root of the elimination tree. To compute the entries of the table $F(u, \kappa, v)$, for all $u \in V$, $\kappa \in \{1, \ldots, \min\{k, |T_u|\}\}$, and $v \in \bar{A}_u$, we proceed in a bottom-up fashion. For each node u, once all entries for the nodes in the subtree of u have been computed, we compute $F(u, \kappa, v)$, for all $\kappa \in \{1, \ldots, \min\{k, |T_u|\}\}$, and all $v \in \bar{A}_u$. Hence, computing each entry $F(u, \kappa, v)$, requires only entries that are already computed. Once all the entries $F(u, \kappa, v)$ are computed, we construct the optimal solution by backtracking – specifically, invoking the subroutine ConstructSolution(r, k, ϵ); pseudocode as Algorithm 1.

Algorithm 1 ConstructSolution(u, κ, v)

```
if F(u, κ, v) = F⁺(u, κ, v) then
    print u
    if κ = 1 then
        return
    (κ*_ℓ, κ*_r) ← argmax_{κ_ℓ + κ_r = κ − 1} F(ℓ(u), κ_ℓ, u) + F(r(u), κ_r, u)
    ConstructSolution(ℓ(u), κ*_ℓ, u)
    ConstructSolution(r(u), κ*_r, u)
else
    (κ*_ℓ, κ*_r) ← argmax_{κ_ℓ + κ_r = κ} F(ℓ(u), κ_ℓ, v) + F(r(u), κ_r, v)
    ConstructSolution(ℓ(u), κ*_ℓ, v)
    ConstructSolution(r(u), κ*_r, v)
```

Theorem 1. *The dynamic-programming algorithm described above computes correctly the optimal solution R^*.*

We note that optimality holds for the elimination order σ which, as explained in Section III is given as an input to the problem. Notice that for each node u the computation of the entries $F(u, \kappa, v)$ requires the computation of partial benefit values $B_u(\{u, v\})$ for pairs of nodes (u, v), which in turn, require access to or computation of values $\mathrm{E}[\delta_q(u; v)]$. As Lemma 5 below shows, the latter quantity can be computed from $\mathrm{E}[\delta_q(u; \emptyset)]$ and $\mathrm{E}[\delta_q(v; \emptyset)]$, for all $u \in V$ and $v \in A_u$. In practice, it is reasonable to consider a setting where one has used historical query logs to learn empirical values for $\mathrm{E}[\delta_q(u; \emptyset)]$ and thus for $\mathrm{E}[\delta_q(u; v)]$.

Lemma 5. *Let $u \in V$ be a given node in an elimination tree T and let $v \in A_u$ denote an ancestor of u. Then,*

$$\mathrm{E}[\delta_q(u; v)] = \mathrm{E}[\delta_q(u; \emptyset)] - \mathrm{E}[\delta_q(v; \emptyset)]. \tag{15}$$

Finally, the running time of the algorithm can be easily derived by the time needed to compute all entries of the dynamic-programming table. We note that the efficiency of the algorithm, as analyzed in Theorem 2, makes it practical to update the materialized factors whenever the input Bayesian network is updated.

Theorem 2. *The running time of the dynamic-programming algorithm is $\mathcal{O}\big(nhk^2\big)$, where n is the number of nodes in the elimination tree, h is its height, and k is the number of nodes to materialize.*

B. Greedy algorithm

The benefit function is non-negative, non-decreasing, and submodular. Hence, it follows from the classic result of Nemhauser et al. [22] that the greedy algorithm for Problem 2 offers a $(1 - 1/e)$ approximation guarantee. Technical details and proofs are in the extended version of the paper [12].

V. Extensions

A. Space budget constraints

The algorithms presented in the previous section address Problem 2, where budget k specifies the number of nodes to materialize. A more practical scenario is Problem 1, where a budget K is given on the total space required to materialize the selected nodes. In this case, for each node u of the elimination tree T the space s_u required to materialize the probability table at node u is given as input. Both algorithms, dynamic-programming and greedy, are extended to address this more general version of the problem. The extension is fairly standard, and thus we describe it here only briefly.

For the dynamic-programming algorithm the idea is to create an entry $F(u, \kappa, v)$ for nodes u and v, and with index κ taking values from 1 to $\min\{K, S_u\}$, where S_u is the total space required to materialize the probability tables of all nodes in T_u. We then evaluate $F(u, \kappa, v)$ as the maximum benefit over all possible values κ_ℓ and κ_r such that $\kappa_\ell + \kappa_r = \kappa - s_u$, where s_u is the space required to materialize node u.

The modified algorithm provides the exact solution in $\mathcal{O}\big(nhK^2\big)$ running time. Note, however, that unlike the previous case (Problem 2) where k is bounded by n, the value of K is not bounded by n. As the running time is polynomial in the *value* of K, which is specified by $\mathcal{O}(\log K)$ bits, it follows that the algorithm is *pseudo-polynomial*. However, the technique can be used to obtain a fully-polynomial approximation scheme (FPTAS) by rounding all space values into a set of smaller granularity and executing the dynamic programming algorithm using these rounded values.

For the greedy algorithm, in each iterative step we select to materialize the node u that maximizes the *normalized marginal gain* $\left(B(R \cup \{u\}) - B(R)\right)/s_u$. The modified greedy algorithm has the same running time and approximation guarantee $\left(1 - \frac{1}{e}\right)$ [23].

B. Accounting for redundant variables

So far we have considered a *fixed* elimination tree T and elimination order σ. The elimination tree T specifies the order in which sum-of-product operations are performed, with one summation for *every variable* in the Bayesian network $\mathcal{N}$. One can observe, however, that it is *not* necessary to involve *every variable* in the evaluation of a query: previous work [5], [8], [24] provides methods to determine the variables that are *redundant* for the evaluation of a query q allowing to perform computations based on a "shrunk" Bayesian network.

Accordingly, one can devise a *redundancy-aware scheme* by materializing different probability tables for a set of "shrunk" Bayesian networks obtained through removal of redundant variables. Here, we provide a brief discussion for it and provide the details in the extended version of the paper [12]. Let $\mathcal{L}$ be the set that contains these shrunk Bayesian networks and the input Bayesian network. The set $\mathcal{L}$ can be represented as a lattice with edges between each network and its maximal subnetworks in $\mathcal{L}$. When a query arrives, it is efficiently mapped to one network in the lattice from which its value can be computed exactly. In this scheme, optimization considerations involve the choice of networks to include in the lattice, as well as the materialized factors for each network.

VI. Experiments

Our empirical evaluation has two parts. In the first part, we evaluate the benefits of materialization for variable elimination. As explained in Section III, we consider materializing only factors resulting from joins and variable summations. In the second part, we compare our approach with two junction tree-based inference algorithms, which also rely on materialization. In what follows, we first describe the experimental setup (Sec. VI-A) and then the results (Sec. VI-B).

A. Setup

Datasets. We use real-world Bayesian networks (see Table I for statistics). Column "parameters" refers to the number of entries of the factors that define the corresponding Bayesian network. Pathfinder [25] is used in an expert system that assists surgical pathologists with the diagnosis of lymph-node diseases. Diabetes [27] models insulin dose adjustment. Mildew is used to predict the necessary amount of fungicides against mildew in wheat. Link [28] models the linkage between a gene associated with a rare heart disease (the human LQT syndrome) and a genetic marker gene. Munin [29] is used in an expert electromyography assistant. Munin#1 and Munin#2 are two subnetworks of Munin. Andes [26] is used in an intelligent tutoring system that teaches Newtonian physics to students. The TpcH Bayesian networks were learned from TpcH data, following Tzoumas et al. [16]. When necessary due to the page limit, we show results only on a subset of datasets, implying that the results are similar on the rest. The full plots are placed in the extended version [12] of the paper. All datasets are publicly available online.[1]

Elimination order. As explained in Section III, elimination trees are determined by the given variable-elimination order. However, finding the optimal order is **NP**-hard [30], and several heuristics have been proposed. Among these heuristics, greedy algorithms perform well in practice [31]. Given a

[1] See https://github.com/aslayci/qtm for the TpcH datasets and http://www.bnlearn.com/bnrepository/ [4] for the rest.

TABLE I
Statistics of Bayesian networks.

Network	nodes	edges	parameters	avg. degree
Mildew	35	46	547 K	2.63
Pathfinder [25]	109	195	98 K	2.96
Munin#1	186	273	19 K	2.94
Andes [26]	220	338	2.3 K	3.03
Diabetes [27]	413	602	461 K	2.92
Link [28]	714	1 125	20 K	3.11
Munin#2 [29]	1 003	1 244	84 K	2.94
Munin [29]	1 041	1 397	98 K	2.68
TpcH#1	17	17	1.5 K	2.00
TpcH#2	31	31	7.4 K	2.00
TpcH#3	38	39	355 K	2.05
TpcH#4	35	37	27 K	2.11

Bayesian network $\mathcal{N}$, a greedy algorithm begins by initializing a graph $\mathcal{H}$ from the "moralization" of $\mathcal{N}$, i.e., by connecting the parents of each node and dropping the direction of the edges. Then at the i-th iteration, a node that minimizes a heuristic cost function is selected as the i-th variable in the ordering. The selected variable is then removed from $\mathcal{H}$ and undirected edges are introduced between all its neighbors in $\mathcal{H}$. In this paper, we consider heuristics where the cost of a node is: *min-neighbors* (MN): the number of neighbors it has in $\mathcal{H}$; *min-weight* (MW): the product of domain cardinalities of its neighbors in $\mathcal{H}$; *min-fill* (MF): the number of edges that need to be added to $\mathcal{H}$ due to its removal; and *weighted-min-fill* (WMF): the sum of the weights of the edges that need to be added to $\mathcal{H}$ due to its removal, where the weight of an edge is the product of the domain cardinalities of its endpoints [30].

Table II shows statistics for the factors in the elimination trees created under orders generated by the aforementioned heuristics. For each Bayesian network, we select the elimination order that induces the smallest average parameter size and use the maximum parameter size as a tie-breaker. In separate experiments, we have confirmed that this heuristic works well, i.e., it chooses an elimination order with the best or close-to-the-best performance and avoids very costly orders [12]. Table III reports statistics of the elimination trees obtained from the chosen elimination order for each dataset.

Query workload. The problem we consider assumes a query workload, i.e., a probability distribution $\Pr(q)$ of queries q. In practice, it is reasonable to consider a setting where one has access to a historical query log. In the absence of such a log for the networks of Table I, we consider queries $q = \Pr(X_q, Y_q = \mathbf{y}_q)$, where $Y_q = \emptyset$ and all variables are either free (X_q) or summed-out (Z_q). Note that the setting $Y_q = \emptyset$ is a worst-case scenario that leads to computationally intensive queries, since, by not selecting any subsets of rows associated with $Y_q = \mathbf{y}_q$ we need to generate larger factors. We consider two workload schemes. In the first scheme we consider *uniform* workloads, where each variable has equal probability to be a member of X_q. For each dataset, we generate a total of 250 random queries, with 50 queries for each query size r_q, i.e., $r_q = |X_q| \in [1, 5]$. In the second scheme we consider *skewed* workloads, where the variables appearing earlier in the elimination ordering are more likely to appear among the summed-out variables Z_q. Specifically, a variable that appears ℓ levels above another in T is ℓ times more likely to be placed among the free variables X_q. We do not show results for workload distributions with opposite skew: since we focus on materialized tables that involve only summed-out variables, it is easy to see that having free variables close to the leaves of the elimination tree T would usually render them not useful.

Cost values. To solve Problem 2, we must assign partial cost values $c(u)$ to the nodes u of elimination trees. Cost must represent the running time of computing the corresponding factor from its children in the elimination tree. Following the time-complexity analysis of Koller et al. [30] for tabular-factor representations, we estimate $c(u)$ to be proportional to the cost of the corresponding natural join operation. We implement joins using the one-dimensional representation of factor tables described by Murphy [32]. For this implementation, the cost of the natural join operation is twice the resulting size of join, which can be calculated from the sizes of the joined tables without actually performing the join. We confirm empirically that the cost estimates align almost perfectly with the corresponding execution times (Pearson $\rho \geq 0.99$).

TABLE II
PARAMETER SIZE OF FACTORS CREATED WITH DIFFERENT ELIMINATION ORDERS (K: THOUSAND, M: MILLION, T: TRILLION).

	MN		MF		WMF		MW	
Network	avg	max	avg	max	avg	max	avg	max
MILDEW	15 K	170 K	**10 K**	**170 K**	57 K	1 M	966 K	19 M
PATHFINDER	570	16 K	**568**	**16 K**	643	16 K	> 1 T	> 1 T
MUNIN#1	749 K	59 M	375 K	39 M	**367 K**	**39 M**	> 1 T	> 1 T
ANDES	2 K	131 K	**1.4 K**	**66 K**	**1.4 K**	**66 K**	> 1 T	> 1 T
DIABETES	9 K	194 K	**4 K**	**194 K**	325 K	33 M	> 1 T	> 1 T
LINK	109 K	17 M	**31 K**	**4 M**	633 K	268 M	> 1 T	> 1 T
MUNIN#2	40 K	31 M	**1.7 K**	**168 K**	1.8 K	168 K	> 1 T	> 1 T
MUNIN	9.5 K	588 K	5 K	392 K	**3 K**	**112 K**	> 1 T	> 1 T
TPCH#1	144	48	144	48	144	48	**144**	**39**
TPCH#2	400	60	400	60	400	60	**280**	**39**
TPCH#3	937 K	30 K	937 K	30 K	112 K	6 K	**4 K**	**300**
TPCH#4	1 K	230	1 K	230	1 K	195	**1 K**	**176**

Algorithms. We compare with other inference algorithms:

(i) Junction tree (JT). Following Lauritzen et al. [8] the tree is calibrated by precomputing and materializing the joint probability distributions for each of its nodes. As discussed in Section II, this supports efficient evaluation of all the queries in which the query variables belong to the same tree node.

(ii) Indexed junction tree (IND). We implement the approach of Kanagal and Deshpande [13], which makes use of a hierarchical index built on the calibrated junction tree. As discussed in Section II, this index contains additional (materialized) joint probability distributions for speeding up "out-of-clique" queries. Index construction requires to specify a parameter denoting the maximum possible size for a potential to be materialized. We set this parameter in terms of total number of entries in a potential by considering candidate values $\{250, 10^3, 10^5\}$ and selecting the one resulting in smallest

TABLE III
STATISTICS OF ELIMINATION TREES.

Tree	nodes	height	max. # children
MILDEW (MF)	70	17	3
PATHFINDER (MF)	218	12	54
MUNIN#1 (WMF)	372	23	7
ANDES (MF)	440	38	5
DIABETES (MF)	826	77	4
LINK (MF)	1 428	56	15
MUNIN#2 (MF)	2 006	23	8
MUNIN (WMF)	2 082	24	8
TPCH#1 (MW)	34	8	3
TPCH#2 (MW)	62	11	5
TPCH#3 (MW)	76	13	5
TPCH#4 (MW)	70	11	4

median query processing cost for each dataset under uniform workload. We compute the cost of these algorithms as done for variable elimination following Koller et al. [30] and confirm that the cost estimates align perfectly with the execution times (Pearson $\rho \geq 0.98$).

Execution system. Experiments were executed on a 64-bit SUSE Linux Enterprise Server with Intel Xeon 2.90 GHz CPU and 264 GB memory. Our implementation is online.[2]

B. Results

Improvement over Variable Elimination. We first report the performance gains that materialization brings to variable elimination. Results for the uniform scheme are shown in Fig. 3. Each plot corresponds to one dataset, with the x-axis showing the number of factors that are materialized (budget k) and the y-axis showing the cost savings in query time, expressed as a percentage of the query time when no materialization is used. The reported savings are averages over the query workload and each bar within each plot corresponds to a different query size r_q. The numbers on the bars indicate the percentage of cost savings relative to the materialization of all the factors in the tree in the uniform-workload scheme.

We observe in Fig. 3 that, consistently in all datasets, a small number of materialized factors can achieve cost savings almost as high as in the case of materializing all factors. This result is expected as the submodularity property of the benefit function implies a diminishing-returns behavior. This result is also desirable as it shows that we can achieve significant benefit by materializing only a small number of factors. Another observation, common to all datasets, is that, as the number r_q of variables in a query increases, the savings from materialization decrease. Given our choice of limiting the materialization operation to factors that involve only joins and variable summation, this trend is expected: with higher r_q, the probability that a materialized factor does not contain any free variable in its subtree decreases, limiting the benefit.

The datasets where we observe considerably small savings are MUNIN#1, ANDES, LINK, and DIABETES. In all these datasets except DIABETES, we find that a small number of factors contribute to the largest part of the computational cost when $k = 0$, due to their large number of entries: we observe that 5 out of 372 factors in MUNIN#1 and 6 out of 1428 factors in LINK contribute to almost 90% of the computational cost, while for ANDES, 5 out of 440 factors contribute to 75%. This suggests that the same computational burden of creating these large factor tables carries to the case of $k > 0$ whenever none of the materialized factors are useful. We indeed observe that the average cost savings, among the queries in which the variables associated to these large factors are summed out, is greater than 90% in these datasets. On the other hand, for DIABETES, we find that the number of entries in the factor tables are almost uniformly distributed, however, the structure of the elimination tree has large chain components, as reflected by its larger height relative to other similar-sized trees, which makes it rare for any chosen factor to be useful. In Table IV we report the average query-processing times when no materialization is used (i.e., k = 0) under the uniform-workload scheme. We observe that the running time increases with the number r_q of free variables in a query. This is expected as free variables lead to larger factor tables during variable elimination. We omit the results per query size for the skewed-workload scheme and simply note that we observed similar trends for it, although with even higher cost savings. Results are provided in the extended version [12].

[2]https://github.com/aslayci/qtm

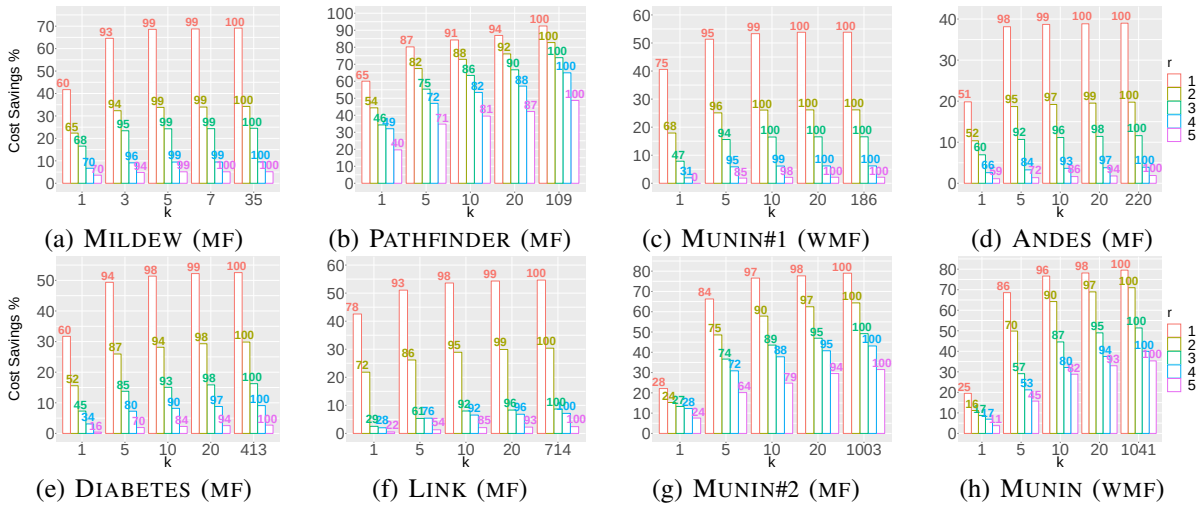
TABLE IV
AVERAGE QUERY PROCESSING TIMES IN SECONDS IN UNIFORM-WORKLOAD SCHEME, WHEN NO MATERIALIZATION IS USED.

Network	$r_q = 1$	$r_q = 2$	$r_q = 3$	$r_q = 4$	$r_q = 5$	all
MILDEW	11.2	33.5	97.5	122.7	177.3	77.0
PATHFINDER	0.2	0.2	0.2	0.3	0.4	0.2
MUNIN#1	319.4	408.4	474.7	656.6	767.2	438.2
ANDES	1.1	1.6	2.6	4.8	7.6	3.5
DIABETES	18.0	95.7	332.3	621.0	801.9	162.3
LINK	119.4	215.8	313.8	391.1	532.0	287.1
MUNIN#2	7.1	11.6	14.5	12.9	25.8	14.4
MUNIN	15.1	16.7	25.8	30.3	38.5	25.3
TPCH#1	0.1	0.1	0.1	0.2	2.1	0.5
TPCH#2	0.1	0.1	0.6	2.9	15.5	3.8
TPCH#3	3.7	5.9	37.3	81.9	183.7	50.9
TPCH#4	0.2	0.7	2.9	25.5	28.2	8.7

We report the overall comparison between the uniform- and skewed-workload schemes in Fig. 4. For MILDEW, average savings are slightly higher than 10% under the uniform-workload scheme: when we drill down to savings specific to query size, as provided in Fig. 3, we see that for queries of size 1, 2, and 3, the savings are around 70%, 35%, and 25% respectively, and have a sharp decrease to under 10% for queries of size 4 and 5, resulting to an average around 10% over all queries. We remind that MILDEW has only 35 variables, which translate to a small elimination tree, making it especially hard to find useful factors for large value of r_q under the uniform workload. Savings improve significantly to an average of 50% in the skewed-workload scheme for MILDEW. On the other hand, for DIABETES, average savings under both workload schemes remain around 10%, due to the elimination tree having large chain components, limiting the extent we can exploit materialization. For the rest of the datasets, PATHFINDER, MUNIN#2, and MUNIN, we observe relatively high savings under both workload schemes, where average savings for $k = 20$ is 70% for PATHFINDER and 50% for the other two Bayesian networks. Overall, we observe that it is possible to obtain up to 99% savings with a small number of materialized factors in both schemes for each dataset.

Comparison with junction tree algorithms. We begin by comparing the computational costs for inference, i.e., query answering. Results for the uniform scheme are shown in Fig. 5, where variable elimination with different levels of materialization is indicated by VE-k. The reported costs are averages over the query workload per query size r_q.

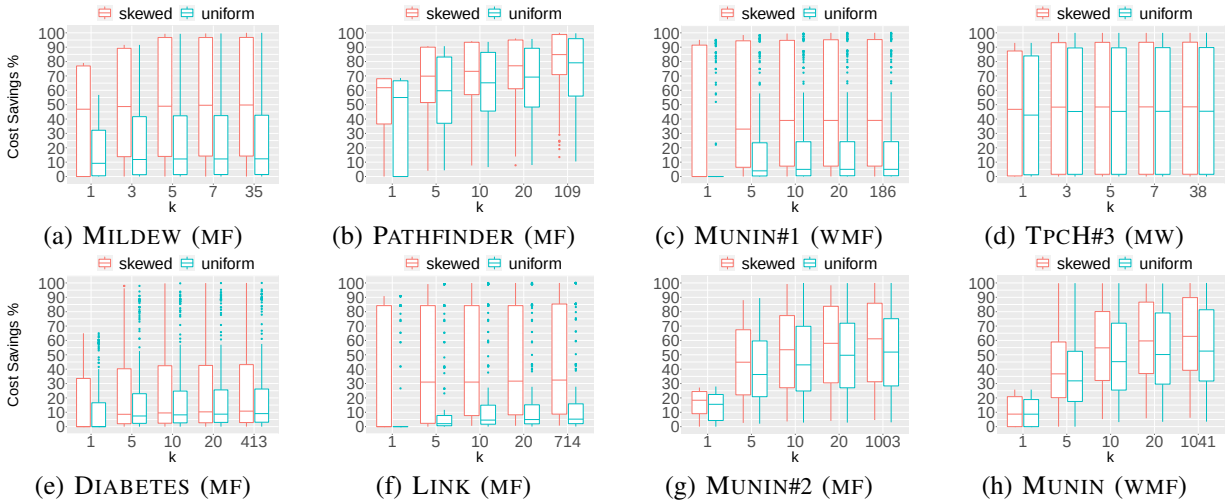
Fig. 5 demonstrates that, for all the datasets, our proposed



(a) MILDEW (MF) (b) PATHFINDER (MF) (c) MUNIN#1 (WMF) (d) ANDES (MF)

(e) DIABETES (MF) (f) LINK (MF) (g) MUNIN#2 (MF) (h) MUNIN (WMF)

Fig. 3. Cost savings per query size r_q in uniform-workload scheme. x-axis: number of materialized factors (budget k). y-axis: cost savings in query running time compared to no materialization. Numbers on the bars: the percentage of cost savings relative to the materialization of all factors.



(a) MILDEW (MF) (b) PATHFINDER (MF) (c) MUNIN#1 (WMF) (d) TPCH#3 (MW)

(e) DIABETES (MF) (f) LINK (MF) (g) MUNIN#2 (MF) (h) MUNIN (WMF)

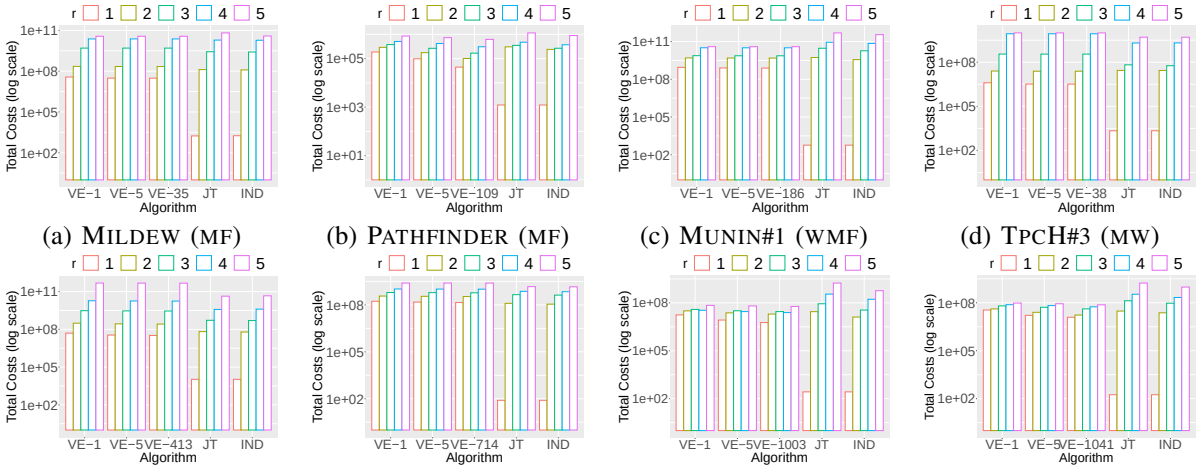
Fig. 4. Cost savings for uniform and skewed workloads. x-axis: budget k. y-axis: cost savings in query running time compared to no materialization.

variable elimination with different levels of materialization is competitive with the junction tree algorithms (JT and IND) for $r_q > 1$, even with a very small materialization budget of $k = 1$. When $r_q = 1$, i.e., when there is only one query variable, JT and IND perform significantly better. This is expected, since the distribution of any single variable is readily available from the materialized junction tree. However for $r_q > 1$, VE-k has comparable performance to JT and IND – and, in fact, VE-k significantly outperforms the other algorithms in half of the datasets, including the largest Bayesian networks MUNIN#2 and MUNIN. This is because in this case it is unlikely for the junction tree to have the joint distribution of $r_q > 2$ variables readily available. Specifically, it is unlikely for the junction tree to contain a materialized joint distribution of exactly the r_q variables included in a given query. In such cases, the junction tree algorithm JT essentially performs variable elimination over the junction tree.

We omit the results for the skewed-workload scheme as the overall trend is similar, with slightly better performance for VE-k. They are made available in the extended version [12]. Instead, we present an aggregate comparison of the algorithms in the uniform and skewed-workload schemes in Fig. 6.

Note that JT and IND are quite sensitive to the query variables, as indicated by the high variability in the costs for answering different queries. This is expected, as their performance depends on how far apart the query variables are located on the junction tree. On the other hand, VE-k is more robust under both workload schemes.

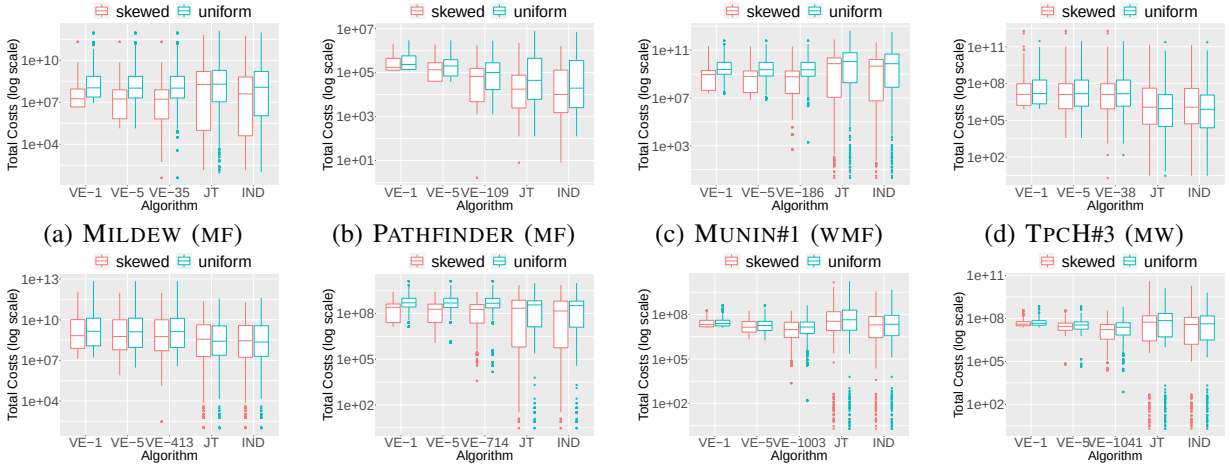
Additionally, we report the computational costs for the offline materialization phase in Table V. In particular, we report the disk space occupied by the materialized structures, along with the running time required for materialization. For our method, we report the results for VE-n, i.e., for the maximum possible budget $k = n$, where all the factors are materialized.



(a) MILDEW (MF) (b) PATHFINDER (MF) (c) MUNIN#1 (WMF) (d) TPCH#3 (MW)

(e) DIABETES (MF) (f) LINK (MF) (g) MUNIN#2 (MF) (h) MUNIN (WMF)

Fig. 5. Total costs per query size r_q in uniform-workload scheme for different algorithms.



(a) MILDEW (MF) (b) PATHFINDER (MF) (c) MUNIN#1 (WMF) (d) TPCH#3 (MW)

(e) DIABETES (MF) (f) LINK (MF) (g) MUNIN#2 (MF) (h) MUNIN (WMF)

Fig. 6. Comparison of total costs under uniform and skewed workloads for different algorithms.

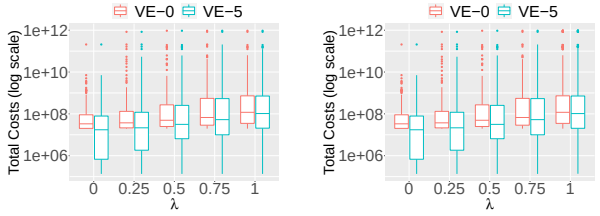
For JT, the costs concern the precomputation of the junction tree (the "calibration" phase). For IND, the costs concern both the junction tree and the additional index.

Table V demonstrates that VE-k significantly outperforms JT and IND both in terms of precomputation time and materialization volume. In fact, for MUNIN#1 and TPCH#3, the junction tree algorithms did not terminate after a two-day-long execution (NA entries in Table V). We conclude that, for settings that include large Bayesian networks and modest to large query sizes, a small level of materialization for variable elimination offers a significant advantage over junction tree based algorithms, as it provides efficient inference (Fig. 5-6), as well as faster and lighter precomputation (Table V).

TABLE V
MATERIALIZATION PHASE STATISTICS.

	Disk Space (MB)			Time (seconds)		
Network	VE-n	JT	IND	VE-n	JT	IND
MILDEW	1.7	373	1 354	5	18 360	18 360
PATHFINDER	< 1	17	23	< 1	302	305
MUNIN#1	317	NA	NA	270	NA	NA
ANDES	4.1	70	78	2	3 682	3 686
DIABETES	15	945	3 286	2	41 228	41 247
LINK	245	3 735	3 824	100	98 533	98 647
MUNIN#2	9	480	573	8	21 348	21 635
MUNIN	14	2 866	2 972	16	110 342	110 645
TPCH#1	< 1	< 1	< 1	0.01	0.306	0.322
TPCH#2	< 1	< 1	< 1	0.02	1.866	1.882
TPCH#3	< 1	NA	NA	0.02	NA	NA
TPCH#4	< 1	4.7	6.9	0.02	106	107

Robustness. We experiment with settings where materialization is optimized for a training query workload $\mathcal{P}$, but the test queries are drawn from a different workload $\mathcal{P}'$. Specifically, we set $\mathcal{P}$ to be either the uniform or the skewed workload; and $\mathcal{P}'$ to consist of queries from the uniform workload in proportion λ and from the skewed workload in proportion $(1-\lambda)$, with varying $\lambda \in [0, 1]$. The results are shown for one Bayesian network (MILDEW) in Fig. 7. In both cases, perfor-



(a) $\mathcal{P}$: Skewed (b) $\mathcal{P}$: Uniform

Fig. 7. Robustness with MILDEW. We explore how performance of variable elimination (with and without materialization) changes when the query workload changes from skewed ($\lambda = 0$) to uniform ($\lambda = 1$).

mance decreases smoothly for increasingly uniform workload $\mathcal{P}'$ (larger values of λ). This happens because in both cases the materialized factors that benefit the queries are the ones that dominate the skewed workload, i.e., queries with variables near the root of the elimination tree.

VII. CONCLUSIONS

In this paper, we studied the problem of materializing intermediate relational tables created during the evaluation of queries over a Bayesian network using variable elimination. We presented efficient algorithms to choose the optimal tables under a budget constraint for a given query workload and variable-elimination order. Our experiments show that appropriate materialization can offer significant inference speed-up and competitive advantages over junction-tree algorithms.

Our work contributes to the growing research on data-management issues in machine-learning systems [33]. One direction for future work is to adapt our framework to other concise factor representations, such as arithmetic circuits [10], [15] and sum-product networks [11] or junction trees [13]. Moreover, in the context of specific applications such as AQP, it is crucial to investigate the maintenance of the materialized factors in the presence of updates. Finally, while the proposed method is shown to be quite robust with respect to changes of the query distribution, it will be valuable to implement a drift-detection mechanism, which automatically prompts an update in the materialization strategy.

VIII. ACKNOWLEDGMENTS

This work has been supported by the MLDB project, funded by the Academy of Finland (decisions 322046 and 325117); the EC H2020 RIA project SoBigData (871042); and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

REFERENCES

[1] C. Bishop, "Model-based machine learning," *Philosophical Transactions of the Royal Society A*, vol. 371, no. 1984, 2013.
[2] L. Getoor, B. Taskar, and D. Koller, "Selectivity estimation using probabilistic models," in *SIGMOD*, 2001.
[3] J. Pearl, *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Elsevier, 2014.
[4] M. Scutari and J.-B. Denis, *Bayesian networks with examples in R*. CRC press, 2014.
[5] N. Zhang and D. Poole, "A simple approach to Bayesian network computations," in *Canadian Conference on Artificial Intelligence*, 1994.
[6] ——, "Exploiting causal independence in Bayesian network inference," *Journal of Artificial Intelligence Research*, vol. 5, 1996.
[7] F. V. Jensen, *An introduction to Bayesian networks*. Springer-Verlag, 1996.
[8] S. Lauritzen and D. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems," *Journal of the Royal Statistical Society*, vol. 50, no. 2, 1988.
[9] A. Darwiche, "A differential approach to inference in Bayesian networks," *Journal of the ACM*, vol. 50, no. 3, 2003.
[10] M. Chavira and A. Darwiche, "Compiling Bayesian networks using variable elimination," in *IJCAI*, 2007.
[11] H. Poon and P. Domingos, "Sum-product networks: A new deep architecture," in *ICCV Workshops*, 2011.
[12] C. Aslay, M. Ciaperoni, A. Gionis, and M. Mathioudakis. (2021) Query the model: precomputations for efficient inference with Bayesian networks. 1904.00079.pdf. [Online]. Available: https://arxiv.org/pdf/
[13] B. Kanagal and A. Deshpande, "Indexing correlated probabilistic databases," in *SIGMOD*, 2009.
[14] R. Dechter, "Bucket elimination: A unifying framework for reasoning," *Artificial Intelligence*, vol. 113, no. 1-2, 1999.
[15] M. Chavira and A. Darwiche, "Compiling Bayesian networks with local structure," in *IJCAI*, 2005.
[16] K. Tzoumas, A. Deshpande, and C. S. Jensen, "Efficiently adapting graphical models for selectivity estimation," *The VLDB Journal*, vol. 22, no. 1, pp. 3–27, 2013.
[17] S. Chaudhuri, B. Ding, and S. Kandula, "Approximate query processing: No silver bullet," in *SIGMOD*, 2017.
[18] T. Kraska, "Approximate query processing for interactive data science," in *SIGMOD*, 2017.
[19] Q. Ma and P. Triantafillou, "Dbest: Revisiting approximate query processing engines with machine learning models," in *SIGMOD*, 2019.
[20] B. Mozafari, "Approximate query engines: Commercial challenges and research opportunities," in *SIGMOD*, 2017.
[21] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and C. Binnig, "Deepdb: Learn from data, not from queries!" *Proceedings of the VLDB Endowment*, 2020.
[22] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions I," *Mathematical programming*, vol. 14, no. 1, 1978.
[23] M. Sviridenko, "A note on maximizing a submodular set function subject to a knapsack constraint," *Operations Research Letters*, vol. 32, 2004.
[24] D. Geiger, T. Verma, and J. Pearl, "d-separation: From theorems to algorithms," in *Machine Intelligence and Pattern Recognition*, 1990, vol. 10, pp. 139–148.
[25] D. Heckerman, E. Horvitz, and B. Nathwani, "Toward normative expert systems; Part I; The pathfinder project," *Methods of information in medicine*, vol. 31, no. 2, 1992.
[26] C. Conati, A. Gertner, K. VanLehn, and M. Druzdzel, "On-line student modeling for coached problem solving using Bayesian networks," in *User Modeling*, 1997.
[27] S. Andreassen, R. Hovorka, J. Benn, K. G. Olesen, and E. R. Carson, "A model-based approach to insulin adjustment," in *AIME 91*, 1991.
[28] C. Jensen and A. Kong, "Blocking Gibbs sampling for linkage analysis in large pedigrees with many loops," *The American Journal of Human Genetics*, vol. 65, no. 3, 1999.
[29] S. Andreassen *et al.*, "Munin: An expert emg assistant," in *Computer-aided electromyography and expert systems*, 1989.
[30] D. Koller, N. Friedman, and F. Bach, *Probabilistic graphical models: Principles and techniques*. MIT press, 2009.
[31] M. Fishelson and D. Geiger, "Optimizing exact genetic linkage computations," *Journal of Computational Biology*, vol. 11, no. 2-3, 2004.
[32] K. Murphy, "Fast manipulation of multi-dimensional arrays in Matlab," Technical report, MIT AI Lab, Tech. Rep., 2002.
[33] S. Hasani, S. Thirumuruganathan, A. Asudeh, N. Koudas, and G. Das, "Efficient construction of approximate ad-hoc ml models through materialization and reuse," *Proceedings of the VLDB Endowment*, 2018.