Workload-Aware Materialization of Junction Trees

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

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

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 popular probabilistic models that capture the conditional dependencies among a set of variables. Inference in Bayesian networks is a fundamental task for answering probabilistic queries over a subset of variables in the data. However, exact inference in Bayesian networks is **NP**-hard, which has prompted the development of many practical inference methods.

In this paper, we focus on improving the performance of the junction-tree algorithm, a well-known method for exact inference in Bayesian networks. In particular, we seek to leverage information in the workload of probabilistic queries to obtain an optimal workload-aware materialization of junction trees, with the aim to accelerate the processing of inference queries. We devise an optimal pseudo-polynomial algorithm to tackle this problem and discuss approximation schemes. Compared to state-of-the-art approaches for efficient processing of inference queries via junction trees, our methods are the first to exploit the information provided in query workloads. Our experimentation on several real-world Bayesian networks confirms the effectiveness of our techniques in speeding-up query processing.

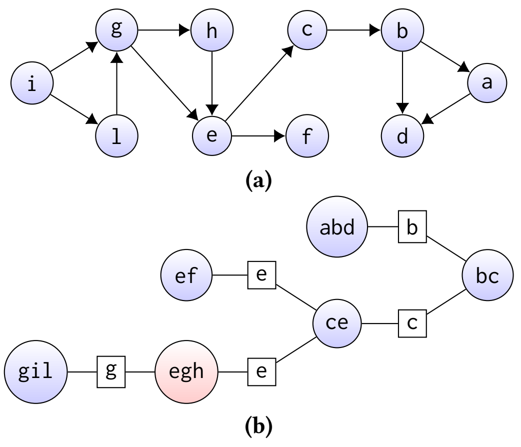


Figure 1: Example: (a) a simple Bayesian network, and (b) corresponding junction tree. An in-clique query $q = \{g, h\}$ can be answered via marginalization from the joint probability distribution associated with the clique node in red.

1 INTRODUCTION

Bayesian networks are probabilistic graphical models that represent a set of variables and their conditional dependencies via a directed acyclic graph. They are powerful models for answering probabilistic queries over variables in the data, and making predictions about the likelihood of subsets of variables when other variables are observed. Inference in Bayesian networks is a fundamental task with applications in a variety of domains, including machine learning [5] and probabilistic database management [15]. However, the problem of exact inference in Bayesian networks is **NP**-hard [27]. This challenge has prompted a large volume of literature that aims to develop practical inference algorithms, both exact [7, 8, 13, 14, 25, 28, 34] and approximate [18].

A state-of-the-art method for exact inference in Bayesian networks is the junction-tree algorithm [25], which enables simultaneous execution of a large class of inference queries. Specifically, the main idea behind the junction-tree algorithm is to convert the Bayesian network into a tree, called "junction tree", and precompute a collection of joint probability distributions for selected subsets of variables, referred to as *cliques* of the tree. This precomputation allows to answer inference queries that involve variables captured by a clique, as demonstrated in Figure 1. However, it does not allow for direct answer of queries that are not captured by a single clique. Such *out-of-clique* inference queries are instead answered via a message-passing algorithm over the junction tree, which often demands a large amount of computations. For instance, in a Bayesian network fully specified by approximately 10^3 parameters, it can take more than a minute to complete the message passing procedure for some queries in our experiments. To reduce the computational burden of out-of-clique queries, Kanagal and Deshpande [21] proposed to materialize suitable joint probability distributions, each corresponding to a partition of the junction tree. In this paper, we follow the same approach and take it one step further by considering a query workload. As in the work of Kanagal and Deshpande, we materialize joint probability distributions corresponding to partitions of the junction tree. Departing from their approach, however, we choose the distributions to materialize in a *workload-aware* manner, and not only based on the structure of the junction tree. Leveraging the information that is available in a query workload allows our method to be optimized for specific settings of interest: in principle, for different query workloads, our method will choose different distributions to materialize, leading to optimal workload processing time. To the best of our knowledge, we are the first to address the task of identifying the *optimal workload-aware* materialization to speed-up the processing of arbitrary inference queries over junction trees.

More concretely, we make the following contributions.

- We define the problem of *workload-aware materialization* of junction trees as a novel optimization problem (Section 3). The general problem of optimally materializing multiple shortcut potentials given a space budget (MOSP) uses, as a subtask, the problem of materializing a single optimal shortcut potential (SOSP). Thus, we define and solve this simpler problem first as a special case, before discussing the general case.
- We prove that both problems are **NP**-hard (Section 4.1).
- We propose PEANUT (Sections 4.2–4.5), a method to optimize query processing based on junction trees through materialization. The offline component of PEANUT utilizes pseudo-polynomial dynamic-programming algorithms to find the optimal materialization for a given workload and under a budget constraint. The online component of PEANUT identifies the materialized distributions to exploit during the processing of a query, translating to reduced message-passing cost, and hence, improved query-answering time.
- In addition to the pseudo-polynomial algorithms, we provide an approximate strongly-polynomial algorithm (Section 4.4) to improve the efficiency of the offline component of PEANUT.
- Our optimal workload-aware materialization algorithm selects *disjoint* shortcut potentials. This constraint makes the problem tractable, but in practice, it leads to solutions that do not fully utilize the available space. To overcome this limitation, we also propose PEANUT+ (Section 4.6), which, based on a simple but effective greedy heuristic, permits overlapping shortcut potentials to be materialized, and leads to better utilization of the available budget. PEANUT+ is the best performing method in practice, and the method of choice.
- We evaluate PEANUT and PEANUT+ empirically over commonly-used benchmarks (Section 5) and show that the proposed materialization, for large enough budget, can save on average between 20% and 40% of query-processing cost, while typically offering an improvement of two orders of magnitude over the previous work of Kanagal and Deshpande [21].

2 RELATED WORK

Recent years have witnessed an increasing amount of uncertain and correlated data generated in a variety of application areas. Uncertain data are collected into probabilistic databases, which can be efficiently represented as probabilistic graphical models [15]. Thus, the problem of querying large probabilistic databases can be formulated as an inference problem in probabilistic graphical models. Although the junction tree algorithm and our materialization can be used for inference on different graphical models, such as Markov random fields, in this work we restrict our attention to Bayesian networks, which are also useful in a variety of tasks beyond querying probabilistic databases. For instance, Getoor et al. [16] resort to Bayesian networks to provide selectivity estimates for typical relational queries. Lately, significant progress has been made towards exploring the connection between data management and graphical models. Khamis et al. [1, 22] propose *FAQ*, a unifying framework for a class of problems sharing the same algebraic structure, which encompasses the processing of the queries considered in our work. Khamis et al. develop a dynamic programming algorithm for the general *FAQ* problem and introduce a notion of *FAQ*-width to characterize its complexity. In 2019, Schleich et al. [31] propose *LMFAO*, an in-memory optimized execution engine for multiple queries over a relational database modelled as a junction tree. *LMFAO* identifies, for each query, the direction of the message passing which leads to the lowest computational cost. A similar optimization will be investigated for PEANUT in future work. Nevertheless, their work does not consider materialization of additional probability distributions not directly captured by the junction tree structure, which is the focus of our work. The conceptually simplest algorithm for exact inference over Bayesian networks is variable elimination [34, 35]. In our recent work, we have developed workload-aware materialization techniques for the variable-elimination inference method [4]. Execution of variable elimination typically involves the computation of marginal-distribution tables, which have to be re-computed every time inference is performed. The junction-tree algorithm [19, 25], closely intertwined with variable elimination, attempts to turn this observation into its advantage by precomputing and materializing several distributions for different subsets of variables. The advantage of such precomputation is that, if an inference query involves only variables within a materialized distribution, as it is the case for all single-variable queries, then the query can be answered directly via marginalization from that distribution. It should be noted that, although the junction-tree algorithm allows to perform inference on arbitrary Bayesian networks, if no restriction is posed to the tree structure, inference may become infeasible. In particular, the feasibility of the method depends on the junction-tree *treewidth*, which is defined as the maximum number of variables in a materialized distribution minus 1. For inference queries that cannot be directly answered from the tree through marginalization, a message-passing algorithm needs to be performed, which may be extremely computationally expensive. To alleviate this issue, Kanagal and Deshpande [21] propose a disk-based hierarchical index structure for the junction tree. They also find that, by precomputing and materializing additional joint probability distributions, it is possible to prune a considerable amount of computations at query time. Our approach also relies on extending the materialization of the junction tree, but in a workload-aware manner.

Finally, note that our approach is based on a dynamic-programming optimization framework. The general idea of framing the selection of a materialization as an optimization problem to be tackled by means of algorithmic techniques was first introduced by Chaudhuri et al. [6].

3 PROBLEM FORMULATION

In this section, we formally define the optimization problems we study. To make the paper self-contained and introduce the necessary notation, we begin with a brief overview of the junction-tree algorithm in Section 3.1. For a more elaborate presentation of the junction-tree algorithm, we refer the reader to the classic papers of Jensen et al. [19] and Lauritzen and Spiegelhalter [25]. After describing the junction-tree algorithm, we proceed with the formal definition of the optimization problems in Section 3.2

3.1 Background on junction trees

The junction-tree algorithm performs exact inference on Bayesian networks using the *junction tree* data structure. In what follows, we provide a brief description of Bayesian networks, the junction-tree structure, and the junction-tree inference algorithm.

Bayesian networks. Bayesian networks are probabilistic graphical models that represent the joint distribution of a set of variables. More formally, a Bayesian network $\mathcal{N}$ is a directed acyclic graph, where nodes represent variables and directed edges represent

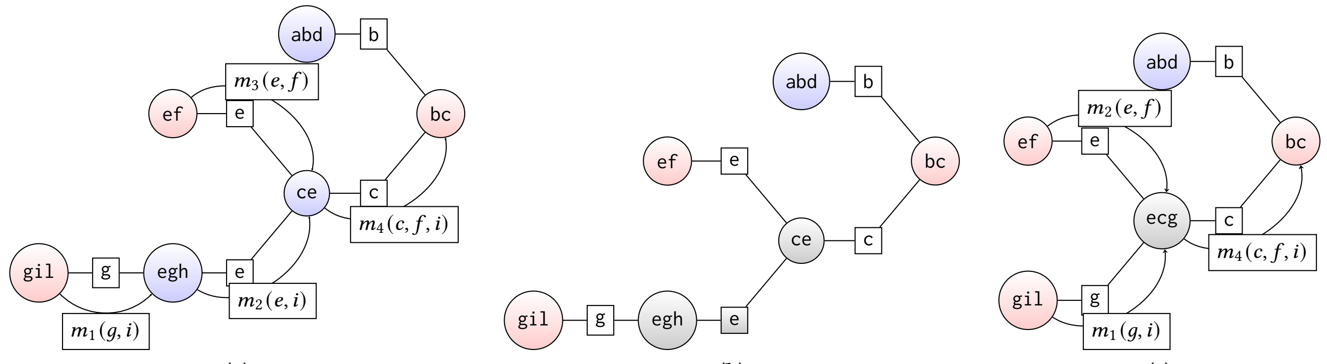


Figure 2: Example: message passing to answer out-of-clique query $q = \{b, i, f\}$, In (a), no materialization is used. In (b), the subtree T_S associated with a shortcut potential S is coloured in grey. In (c), materialization of S is used.

variable dependencies. A Bayesian network allows to express the joint probability of all variables as a product of a finite number of factors, so that each factor corresponds to the conditional probability of a variable given the value of its parents.

In our presentation, we assume that all variables are categorical. In practice, numerical variables can be handled as categorical via discretization into categorical intervals.

Junction trees. Given a Bayesian network $\mathcal{N}$ over a set of variables X, a junction tree $T = (V, E)$ is built from the Bayesian network in five steps: (1) moralization, (2) triangulation, (3) clique-graph formation, (4) junction-tree extraction, and (5) junction-tree calibration. The result of this procedure is the *junction tree* of the Bayesian network $\mathcal{N}$. The nodes of T are referred to as *clique nodes* and the edges as *separators*. Each *clique node* is assigned a set of Bayesian network factors, and their product is used to initialize the *clique potential*, which is a table mapping each configuration of the associated variables to a non-negative real value. In what follows, we will refer to the variables associated with a clique node u or a separator (u, v) as the *scope* of the clique node or the separator, and we will denote them with X_u and $X_{u,v}$, respectively. In the last step of the junction tree construction, the junction tree T is *calibrated* via the *Hugin* algorithm [12]. In short, this means that the clique potentials of the junction tree are normalized to coincide with the joint distributions of the related variables. By construction, a junction tree satisfies the *running-intersection property*, which states that if $a \in X_u$ and $a \in X_v$ for two nodes u and v in T, then $a \in X_w$ for all nodes w that lie on the path from u to v, which we denote by $path_T(u, v)$. This property is crucial for inference (described later in this section). Moreover, one node $r \in V$ of the junction tree T is selected and marked as *pivot* (root), and is the node towards which all messages are sent. The choice of the pivot node affects the order and the size of the set of messages sent to process an inference query. Selecting a pivot node that is optimal for all possible queries is not possible. For our purposes, we consider an arbitrary node to be the pivot. For future work, however, it would be interesting to study the problem of finding the materialization that is optimal across all pivot selections. The junction-tree algorithm is used to answer inference queries on the Bayesian network using the junction-tree structure defined above. An inference query $q \subseteq X$ is defined by a subset of variables in X and asks for the joint or conditional probability distribution of the variables appearing in q. In what follows, we focus only on joint-probability queries, as conditional probabilities can be obtained from the answer to joint-probability queries. Inference queries can be separated into two types, *in-clique* and *out-of-clique* queries. In-clique queries correspond to cases where all the variables of the query q are associated with the same clique node of the junction tree. The junction-tree algorithm answers in-clique queries by directly marginalizing the joint probability of the variables corresponding to the clique node. For example, in Figure 1, the query $q = \{g, h\}$ is answered by marginalizing the calibrated potential of the clique node egh. Out-of-clique queries correspond to cases where not all variables are associated with the same clique node. For out-of-clique queries, a *message-passing* procedure is invoked over a Steiner tree T_q, the smallest tree that connects all the clique nodes containing the query variables. When performing message passing, we always consider the direction of the messages induced by the pivot r of T. As a consequence, the pivot (root) node r_q of T_q (i.e., the clique node towards which all messages are sent) is defined to be the clique node closest to the pivot r of the junction tree and therefore, $r_q = r$ if r is contained in the Steiner tree T_q. Messages are sent towards the pivot r_q in discrete steps from one node to the next, starting from the leaves of T_q. The message sent from a given clique node u to the next is the product of its own potential with the messages it receives from other nodes. Once the pivot r_q has received messages from all its neighbours in T_q, it computes the answer to the query q by summing out all the non-query variables.[1] An example of the described message-passing procedure is given in Figure 2(a). The query variables $q = \{b, i, f\}$ are not all included in the same clique. Thus, the Steiner tree T_q that contains the cliques having all the query variables within their scopes (highlighted in red), is extracted. The pivot of both the junction tree and the Steiner tree is bc, and hence the leaves gil and ef begin the message passing. After message $m_4(c, f, i)$ is sent to the pivot, we have a potential corresponding to the joint probability distribution of variables $\{b, c, f, i\}$ from which we sum out variable c to obtain the query answer, namely the joint probability distribution of variables b, i and f.

3.2 Problem statement

At a high level, our objective is to minimize the expected running time for queries drawn from the same distribution as of queries in

[1]In a practical implementation, an equivalent but more efficient approach is to sum-out variables as soon as possible — i.e., marginalization is performed for each message before it is sent to the next clique node, to compute the marginal distribution only of the variables belonging to the query and the separator over which the message is sent, since all the other variables are redundant and would later be summed out.

a given workload. The approach we take is to materialize *shortcut potentials*, i.e., additional data structures introduced by Kanagal and Deshpande [21]. Depending on the query, shortcut potentials may be utilized by the junction-tree algorithm to skip part of the computations. Naturally, this approach introduces a trade-off between the benefit in reduced running time, on one hand, and the cost of storing the materialized data structures, on the other. In what follows, we define shortcut potentials, we quantify the cost and benefit of their materialization, and finally, we formally define the optimization problems we consider.

Shortcut potentials. To speed-up out-of-clique query processing in junction trees, Kanagal and Deshpande [21] propose to store certain joint probability distributions, the so-called *shortcut potentials*, which allow to substantially reduce the amount of computations. A shortcut potential S is identified by a subtree $T_S \subseteq T$ of the junction tree with vertices $V(S)$ and is defined as the joint distribution of all variables in the scope of those separators that "cut" the subtree T_S from the junction tree T. This set of separators is henceforth denoted as $cut(S)$. Moreover, as for any other subtree, the pivot (root) r_S of T_S is defined to be the clique node closest to the pivot r of the junction tree T— and therefore, $r_S = r$ if r is contained in $V(S)$. For example, in Figure 2(b), the highlighted shortcut potential is identified by the subtree consisting of clique nodes egh and ce; and defined as the joint distribution of the separator variables c, e, g. Shortcut potentials allow to reduce the size of the Steiner tree T_q extracted to compute the answer of q, thus, decreasing the associated message-passing cost. For example, consider the processing of query $q = \{\mathrm{b}, \mathrm{i}, \mathrm{f}\}$ on the junction tree in Figure 2(a). By utilizing the shortcut potential S that corresponds to the highlighted subtree T_S in Figure 2(b), we perform message passing on a smaller but valid Steiner tree and avoid the computation of $m_2(\mathrm{e}, \mathrm{i})$, as shown in Figure 2(c).

Cost of a shortcut potential. The cost or weight of a shortcut potential captures the amount of storage space it takes when materialized. Formally, we define the cost as the size $\mu(S)$ of the probability-distribution table S. This quantity is given by the product of the cardinalities of all the variables in the shortcut-potential scope X_S. The cost is upper bounded by the product of the cardinalities of the variables in the separators $(u, v) \in cut(S)$.

Benefit of a shortcut potential. Intuitively, the notion of benefit is meant to capture the expected savings in running time achieved by the materialization of a shortcut potential. For a shortcut potential to have a positive benefit, it has to be *useful* (i.e., to be possible to utilize it by the junction-tree algorithm) for some queries in the query workload. Therefore, before we quantify the notion of *benefit*, let us define the notion of *usefulness*. In plain words, a shortcut potential S is useful for a query when replacing the nodes of $V(S)$ with the shortcut potential S allows the junction-tree algorithm to compute the same result for query q but more efficiently. The precise conditions under which this is possible depend on whether or not the shortcut potential subtree includes the pivot of the Steiner tree, i.e., $r_q \in V(S)$. On the one hand, if $r_q \in V(S)$,we need at least a path from the leaves of T_q to r that passes through a separator in $cut(S)$. On the other hand, if $r_q \notin V(S)$, we need at least two separators of $cut(S)$ in the same path between any leaf of T_q and r. In both cases, moreover, no query variables should be left out of T_q when $V(S)$ is replaced with S. Formally, we have the following definition.

Definition 3.1 (Usefulness). Consider a query q with associated Steiner tree T_q and a shortcut potential S. We say that S is *useful* for q if *one* of the following conditions holds:

(i) $r_q \notin V(S)$ and there are at least two separators $\{(u, v), (w, z)\} \in cut(S)$ in the path between any leaf of T_q and r_q;
(ii) or $r_q \in V(S)$ and there is at least one separator $(u, v) \in cut(S)$ in the path between any leaf of T_q and r_q;

and in addition replacing $V(S)$ in T_q with S results in a Steiner tree that still contains all the query variables while leading to a lower message-passing cost. We define:

$$\delta_S(q) = \begin{cases} 1 & \text{if } S \text{ is useful for } q, \\ 0 & \text{otherwise.} \end{cases}$$

To quantify the benefit of materializing a shortcut potential, we propose a definition of *shortcut-potential benefit*, which relies on the definition of usefulness. Intuitively, the benefit of a shortcut potential for a query should reflect the amount of message-passing operations in T_S that are now avoided thanks to the shortcut potential S. Such amount can be computed based on the size of the clique potentials and the cardinality of the query variables. This motivates the following definition.

Definition 3.2. The benefit of a shortcut potential S with respect to a query q is defined as

$$B(S, q) = \delta_S(q) \sum_{v \in V(S)} \mu(v) \prod_{w \in X_{T_v} \cap q} \alpha(w),$$

In the above definition, $\mu(v)$ denotes the size of the clique potential of a clique node v, $\alpha(w)$ denotes the cardinality of a variable w, and X_{T_v} is union of the scopes of all the cliques in the subtree rooted at clique v. The benefit of a shortcut potential S with respect to a query log Q should take into account not only the benefit with respect to individual queries, but also query probabilities to guarantee that queries that are most likely to occur will be assigned a higher weight in the computation of benefit.

Definition 3.3 (Benefit). The benefit of a shortcut potential S with respect to a query log Q is defined as

$$B(S, Q) = \sum_{q \in Q} Pr_Q(q)\, B(S, q).$$

Here, $Pr_Q(q)$ is the probability of query q being drawn from the query log Q, which in practice can be estimated from the frequency $f(q)$ of the available queries.

Materializing a single optimal shortcut potential. We build on the work of Kanagal and Deshpande [21] to further improve the efficiency of query processing using junction trees. We use a simple idea: take the anticipated query workload into account. One may have a precise idea of the anticipated query workload — for example when a large historical query log Q is available, from which relative frequencies of different queries are known, possibly indicating that some queries are far more likely than others. Alternatively, there may be uncertainty about the anticipated workload. Even in the extreme case in which no historical query log is available, one may optimize materialization for an "uninformative" (e.g., uniform) distribution of queries. In all cases, the available information about possible queries can guide the selection of the shortcut potentials to materialize. Indeed, it is expected that some regions of the junction tree will be relevant for far more queries than others — or associated with a higher volume of computation (i.e., larger messages in the junction-tree algorithm). Previous work, however, is agnostic to queries. Therefore, we introduce here the problem of choosing, for a given r_S, a shortcut potential of optimal benefit with respect to a given query workload, under a user-specified budget constraint. Formally:

Problem 1 (SOSP: Single Optimal Shortcut Potential). *Consider a junction tree $T = (V, E)$, with pivot r, a query log Q, and space budget K. We are asked to find a single shortcut potential S, with pivot $r_S \in V$, such that the benefit $B(S, Q)$ is maximized, subject to the constraint $\mu(S) \leq K$.*

In simple words, Problem 1 seeks the shortcut potential that would avoid the maximum volume of message-passing operations for query log Q, under the constraint that it can be materialized within a given space budget.[2] The solution to Problem 1 is a shortcut potential rooted at r_S. Clearly, our optimization problem relies on the simple assumption that the workload is stationary.

Materializing multiple optimal shortcut potentials. In practice, it would be desirable to materialize not just one, but any number of shortcut potentials within a given space budget. We consider this problem while restricting our attention to *non-overlapping* shortcut potentials, i.e., node-disjoint subtrees T_S of T. For this problem, as shown in the next section, we can show that a dynamic-programming algorithm provides the exact solution. More formally, we consider the following problem.

Problem 2 (MOSP: Multiple Optimal Shortcut Potentials). *Consider a junction tree $T = (V, E)$ with pivot r, a query log Q, and space budget K. We are asked to find a set of node-disjoint shortcut potentials $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$, for some $k \geq 1$, such that the total benefit*

$$\sum_{i=1}^{k} B(S_i, Q)$$

is maximized, subject to the constraints $\sum_{i=1}^{k} \mu(S_i) \leq K$ and $V(S_i) \cap V(S_j) = \emptyset$ for all $i \neq j$ with $i, j = 1, \ldots, k$.

Note that the previous work of Kanagal and Deshpande [21] considers materializing a hierarchical set of shortcut potentials such that some shortcut potential subtrees are nested into others. While the shortcut potentials we select are optimal only under the assumption of exclusion of overlaps and stationarity of the query workload distribution, the workload-aware nature of our approach is sufficient to lead to superior empirical performance.

4 COMPLEXITY AND ALGORITHMS

In this section, we first prove that problems SOSP and MOSP, defined in the previous section, are both **NP**-hard. We then present PEANUT, a method to optimize inference based on junction trees. PEANUT is composed of an *offline* and an *online* component.

The offline component tackles the SOSP and MOSP problems; the corresponding algorithms are described in Sections 4.2 and 4.3, respectively. Both algorithms are based on dynamic programming, they run in *pseudo-polynomial time*, and provide an optimal solution. Hereafter, it is convenient to consider T as rooted at the pivot $r \in V$. The algorithm for the MOSP problem, named BUDP, follows a standard bottom-up traversing scheme, while the algorithm for the SOSP problem, named LRDP, visits the nodes of the tree in a "left-to-right" order. Both algorithms require two dynamic-programming passes, one to compute the benefit of the optimal solution, and one to obtain the set of separators that are part of the optimal solution. Additionally, in Section 4.4, we present a *strongly-polynomial algorithm* that provides a trade-off between execution time and solution quality. The online component of PEANUT follows the standard junction-tree-based inference approach and it is described in Section 4.5. Finally, in Section 4.6 we describe PEANUT+, a practical extension of PEANUT.

4.1 Problem complexity

We now establish the complexity of the problems defined in Section 3.2.

All proofs are omitted due to space constraints, but are available in the extended version of the paper [10].

We start with the problem of selecting a single optimal shortcut potential (SOSP).

Theorem 4.1. *The problem of selecting a single optimal shortcut potential* (SOSP) *is* **NP***-hard.*

Theorem 4.1 is proven via a reduction from the *tree-knapsack problem* (TKP).

A similar complexity result holds for the more general problem of finding multiple node-disjoint optimal shortcut potentials (MOSP).

Theorem 4.2. *The problem of finding multiple node-disjoint optimal shortcut potentials* (MOSP) *is* **NP***-hard.*

Theorem 4.2 is proven via a reduction from the 0–1 knapsack problem (KP).

4.2 Materializing a single shortcut potential

Despite the hardness of the two problems, we now show that they are not strongly **NP**-hard, and so it is possible to obtain an optimal solution in *pseudo-polynomial time*. We start with the SOSP problem. The algorithm we design follows the *left-to-right* (or depth-first) dynamic programming approach [9, 20, 30], which we appropriately name LRDP. The procedure is illustrated in Algorithm 1. We assume that the nodes of the tree are labeled from 0 (the pivot r_S) to n in a depth-first manner and "from left to right." Consider any node v. Let S_v be the shortcut potential such that $V(S_v)$ includes all the nodes in the path $path(\pi_v, r)$, where π_v is the parent of v. In what follows, for simplicity of notation, we refer to the benefit and cost of the shortcut potential S_v as $b^Q(v)$ and $c(v)$, respectively. The LRDP algorithm has two building blocks: a *forward* and a *backward* step. The forward step computes the benefit $b^Q(v)$ and the cost $c(v)$, for each node v, visiting nodes in a depth-first fashion. This is the only step in which the query-log is used. The backward step is performed every time a leaf node u of the junction tree T is reached, while there are no nodes with larger label to visit in the forward step. Let us denote the leaf nodes of T by $leaves(T)$. To perform the backward step at node v, we consider all cost values $c \in [K]$. For each value c, we find the optimal combination of the children of π_v with label v or lower (i.e., the children that have currently been visited by the algorithm) that maximize the benefit under the constraint that the total cost does not exceed c. Let Γ denote the optimal combination and $b^*(\pi_v)$ the corresponding benefit. To complete the backward step, we compare $b^*(\pi_v)$ with $b^Q(v)$ and $b^Q(\pi_v)$.

[2] We considered also an alternative objective function that subtracts $\mu(S)$. from the benefit $B(S, q)$. This small correction in the objective accounts for the number of message-passing operations performed by the junction-tree algorithm with materialization. However: (*i*) this would make the optimization problems less tractable; and (*ii*) the cost $\mu(S)$ is already taken into account in the budget constraint. For these reasons, we opted to optimize the benefit as defined in the text. Nevertheless, in our experiments, we do compare the exact computational cost of algorithms.

Algorithm 1 Left-to-right dynamic programming (LRDP) for the single optimal shortcut potential (SOSP) problem

```
Input: junction tree $T$, budget $K$, query log $Q$, pivot of $S$ $r_S$
 1: for $c \leftarrow 0$ to $K$ do
 2:     $I[r_S, c] \leftarrow 0,\ P[r_s, c] \leftarrow -\infty$
 3: for $v \leftarrow 1$ to $n$ do
 4:     $P \leftarrow$ ForwardMove($v, P, K$)
 5:     if $v \in leaves(T)$ then
 6:         while ($v \neq r_S$) and
                    (there is no $i \in Children(v)$ s.t. $i > v$) do
 7:             $I, P \leftarrow$ BackwardMove($v, I, P, K$)
 8:             $v \leftarrow \pi_v$
    return $I, P$

 9: procedure ForwardMove($v$, $P$, $K$)
10:     for $c \leftarrow 0$ to $K$ do
11:         if $c > c(v)$ then
12:             $P[v, c] \leftarrow b^Q(v)$
        return $P$

13: procedure BackwardMove($v$, $I$, $P$, $K$)
14:     for $c \leftarrow 0$ to $K$ do
15:         $currChildren(\pi_v) \leftarrow \{u \in Children(\pi_v) : u \leq v\}$
16:         $\Gamma'_c = \{\Gamma : \Gamma \subseteq currChildren(\pi_v) \wedge \sum_{child \in \Gamma} c(child) \leq c\}$
17:         $b^*(\pi_v)[c] \leftarrow \max_{\Gamma_c \in \Gamma'_c} \sum_{child \in \Gamma_c} P[c(child), child]$
18:         $\Gamma \leftarrow \arg\max_{\Gamma_c \in \Gamma'_c} \sum_{child \in \Gamma_c} P[c(child), child]$
19:         $\eta \leftarrow \max\{P[v, c], P[\pi_v, c], b^*(\pi_v)[c]\}$
20:         if $\eta = P[v, c]$ then
21:             for $child \in currChildren(\pi_v)$ do
22:                 $I[child, c] \leftarrow 0$
23:             $I[v, c] \leftarrow 1$
24:         if $\eta = b^*(\pi_v)[c]$ then
25:             for $child \in currChildren(\pi_v)$ do
26:                 $I[child, c] \leftarrow 0$
27:                 if $child \in \Gamma$ then
28:                     $I[child, c] \leftarrow 1$
29:         if $\eta = P[\pi_v, c]$ then
30:             for $child \in currChildren(\pi_v)$ do
31:                 $I[child, c] \leftarrow 0$
32:         $P[\pi_v, c] \leftarrow \eta$
        return $I, P$
```

Algorithm 2 Reconstruct solution found by LRDP algorithm

```
Input: junction tree $T$, budget $K$, matrix $I$ and $r_S$ from
       Algorithm 1
 1: for $c \leftarrow 0$ to $K$ do
 2:     $toVisit \leftarrow \{Children(r_S)\}$
 3:     $currPath \leftarrow \emptyset$
 4:     $S[r_S, c] \leftarrow \emptyset$
 5:     while $toVisit.size > 0$ do
 6:         $v \leftarrow toVisit.pop()$
 7:         if ($v \in leaves(T)$) and ($I[v, c] = 1$) then
 8:             $S[r_S, c] \leftarrow S[r_S, c] \cup (v, \pi_v)$
 9:             $currPath \leftarrow \emptyset$
10:         if ($currPath.size > 0$) and ($I[v, c] = 0$) and
                (there is no $child \in Children(\pi_v)$ s.t. $child > v$)
    do
11:             $S[r_S, c] \leftarrow S[r_S, c] \cup \{(\pi_v, p_{\pi_v})\}$
12:             $currPath \leftarrow \emptyset$
13:         if $I[v, c] = 1$ then
14:             $currPath \leftarrow currPath \cup \{v\}$
15:             $toVisit \leftarrow toVisit \cup Children(v)$
    return $S$
```

Based on this comparison, we keep track of the nodes that contribute to the optimal solution and of the optimal benefit value. We tabulate the results of the computation using two matrices, P and I. The former stores benefit values, whereas the latter acts as an indicator and is used to reconstruct the solution. In particular, we set $I[v, c] = 1$ for each node v such that the separators (v, π_v) are part of the $cut(S)$ associated with the optimal shortcut potential of cost c. We also set $I[u, c] = 1$ for each node u in the path between v and r. Similarly, $P[v, c]$ stores the current optimal benefit for a shortcut potential when the algorithm visits node v. After the last backward step is carried out, the LRDP algorithm has computed the optimal shortcut potential for all cost values $c \in [K]$. Once the optimal single shortcut-potential benefit has been computed, it is rather simple to reconstruct the solution. The reconstruction procedure is described in Algorithm 2. Two auxiliary sets are used, *currPath* and *toVisit*. The main idea is that all nodes that identify the optimal shortcut-potential subtree are marked by the matrix I, as discussed above. Specifically, all nodes in a path between the root r_S of S and a node i for which the separator (i, π_i) is part of the set $cut(S)$ identifying the optimal shortcut potential are assigned value 1 in the indicator matrix. Thus, it is necessary to retrieve the bottom-most node marked in I with 1 for each such path. The reconstruction procedure traverses the tree top-down, visiting all the nodes indicated by the indicator matrix I in a depth-first ("left-to-right") manner. This order is respected because the *pop* operation in Line 6 returns the node with the smallest DFS label. The execution of Algorithm 2 gives the set of separators whose joint probability distribution is the optimal shortcut potential. Algorithms 1 and 2 can be executed with each internal node of the tree T being considered as root r_S, so as to obtain the single optimal shortcut potential $S[r_S, c]$ rooted at r_S and of cost c as well as its associated optimal benefit $P(S[r_S, c])$, for each $r_S \in V$ and each cost value $c \in [K]$. This is used in developing the solution for the MOSP problem, as we see shortly. Without delving into details, it is worth noting that when executing the algorithm for all $r_S \in V$, it is possible to share computations among different roots.

Time complexity. The running time of LRDP is pseudo-polynomial; more precisely $O(nK^2)$. The quadratic term is due to the computation of $b^*(\pi_v)$ in the backward step, which is performed once for each clique node $v \neq r$. A more fine-grained expression of the time complexity for LRDP would include the complexity for computing the benefit values $b^Q(v)$, but in practice this additional complexity is dominated by the backward step.

4.3 Materializing multiple shortcut potentials

We now show how to obtain an optimal packing for MOSP, i.e., k node-disjoint shortcut potentials. Recall that k is not specified in the input, but rather it is optimized by the algorithm. In other words, we find the optimal set of non-overlapping shortcut potentials leading to the largest total benefit while satisfying the budget constraint. We stress that the shortcut potentials we retrieve are not optimal in the general sense, but are optimal under the constraints introduced in our problem formulation. As a preprocessing step, we first execute Algorithms 1 and 2 with each node of the tree being considered a root. As a result we compute

Algorithm 3 Bottom-up dynamic programming (BUDP) for the multiple optimal shortcut potentials (MOSP) problem

```
Input: junction tree $T$, budget $K$, $P$ from Algorithm 1,
       $S$ from Algorithm 2
for each $v \in V$ do
    for $c \leftarrow 0$ to $K$ do
        $I[v, c] \leftarrow 0$
for each $v \in V$ do
    if $v \notin leaves(T)$ then
        for $c \leftarrow 0$ to $K$ do                                ▷ case (i)
            $\Gamma_c \leftarrow \{c(child)$ for all $child \in Children(v)$ s.t.
                    $\sum_{child \in Children(v)} c(child) \le c\}$
            $\phi_1 \leftarrow \sum_{child \in Children(v), c(child) \in \Gamma_c} P(S[child, c(child)])$
            $H_1[v, c] \leftarrow \max_{\Gamma_c} \{\phi_1\}$
            for each $child \in Children(v)$ do
                $W_1[child, c] \leftarrow \arg\max_{c(child)} \{\phi_1\}$
        for $c \leftarrow 0$ to $K$ do                                ▷ case (ii)
            $\Gamma_{c',c} \leftarrow \{c(d)$ for all $d \in D(S[v, c'])$ s.t.
                    $\sum_{d \in D(S[v,c'])} c(d) \le c - c'\}$
            $\phi_2 \leftarrow P(S[v, c'])+$
                    $\sum_{d \in D(S[v,c']), c(d) \in \Gamma_{c',c}} P(S[d, c(d)])$
            $H_2[v, c] \leftarrow \max_{c', \Gamma_{c',c}} \{\phi_2\}$
            $W_2[v, v, c] \leftarrow \arg\max_{c'} \{\phi_2\}$
            for each $d \in D(S[v, c'])$ do
                $W_2[v, d, c] \leftarrow \arg\max_{c(d) \in \Gamma_{c',c}} \{\phi_2\}$
        for $c \leftarrow 0$ to $K$ do
            if $H_2[v, c] > H_1[v, c]$ then
                $I[v, c] \leftarrow 1$
                $H[v, c] \leftarrow H_2[v, c]$
            else $H[v, c] \leftarrow H_1[v, c]$
return $I, H$
```

Algorithm 4 Reconstruct solution found by BUDP algorithm

```
procedure RECONSTRUCT($v, c$)
    if $I[v, c] = 1$ then
        $c' \leftarrow W_2[v, v, c]$
        $\mathcal{S} \leftarrow \mathcal{S} \cup S[v, c']$
        for each $d \in D(S[v, c'])$ do
            $c(d) \leftarrow W_2[v, d, c]$
            return RECONSTRUCT($u, c(d)$)
    else
        for each $child \in Children(v)$ do
            $c(child) \leftarrow W_1[child, c]$
            return RECONSTRUCT($child, c(child)$)
```

the single optimal shortcut potentials for each node $r_S \in V$ and each cost value $c \in [K]$. We proceed with a bottom-up dynamic-programming algorithm, named BUDP and shown as Algorithm 3. In the BUDP algorithm, each node v is visited when all its children $Children(v)$ have already been visited. When visiting v, if it is not a leaf, we compute the value of the benefit of the optimal packing in the subtree rooted at v, while considering two cases: (i) the solution includes a shortcut potential subtree rooted at v, and (ii) the solution does not include a shortcut potential subtree rooted at v. In case (i), the total benefit $H_1[v, c]$ at v for a given cost c is simply obtained by combining the optimal solutions at each node $child \in Children(v)$. In case (ii), the total benefit $H_2[v, c]$ at v for a given cost c is the benefit of the best combination of the optimal shortcut potential S rooted at v and the optimal solutions at the nodes s belonging to the set $D(S)$ of descendants of S defined as $D(S) = \{s : d \notin V(S[v, c])$ and $p_u \in V(S[c, v])\}$. For any internal node v, we also use an indicator matrix I to store which of the two cases provides the largest benefit, i.e., $I[v, c] = 1$ if $H_2[v, c] > H_1[v, c]$, and $I[v, c] = 0$ otherwise. Additionally, in order to reconstruct the optimal solution, for the optimal packing of cost c in the subtree rooted at a node v, it is necessary to store the cost allocated to each shortcut potential that is part of the optimal packing in that subtree. For this purpose, we use two matrices W_1 and W_2. The matrix W_1 is 2-dimensional, while W_2 is 3-dimensional because a node can be a child of only one parent but a descendant of multiple shortcut potentials. Having computed the benefit associated with the optimal packing of k shortcut potentials, it is necessary to reconstruct the solution. It is convenient to address this task recursively, as shown in Algorithm 4, where the nodes of tree T are traversed top-down. The matrices I, W_1, and W_2 returned by BUDP are used in Algorithm 4. Every time a node v is visited, we use the indicator matrix I to distinguish the two cases discussed above. If case (i) is optimal for the subtree rooted at v, we continue the recursion at nodes $u \in Children(v)$ with weights indicated by matrix W_1. Otherwise, if case (ii) is optimal, we continue the recursion at nodes $d \in D(S)$ with the weights given by matrix W_2. For the initial invocation of RECONSTRUCT we pass as arguments the root of T and the budget K. The method returns the optimal packing $\mathcal{S}$ of node-disjoint shortcut potentials, i.e., the solution to the MOSP problem.

Time complexity. The running time of the BUDP algorithm is $O(nK^3)$. Each of the internal nodes of the junction tree is visited once, and the total benefits corresponding to cases (i) and (ii) are evaluated. The cubic term stems from the evaluation of the total benefit under case (ii). Accounting for the preprocessing, where the LRDP algorithm is called $O(n)$ times, the total complexity of solving the MOSP problem is $O(n^2K^2 + nK^3)$.

4.4 A strongly-polynomial algorithm

In addition to the pseudo-polynomial algorithms introduced in the previous sections, we now present a strongly-polynomial variant for both problems SOSP and MOSP. This strongly-polynomial algorithm is not optimal, clearly, but it performs very well in practice as we will see in our experimental evaluation. The main idea is to reduce the amount of admissible values for the budget. Instead of considering all budget values $c \in \{0, \dots, K\}$, we can work with an exponentially smaller set. In particular, for some real value $\epsilon \ge 1$, the set $\{0, \dots, K\}$ is partitioned into a smaller number of bins of increasing size $\lfloor \epsilon^i \rfloor$, where i is the bin number. We thus form a reduced set $\{0, \lfloor \epsilon \rfloor, \lfloor \epsilon^2 \rfloor, \dots, K\}$, which is used by agorithms LRDP and BUDP in lieu of $\{0, \dots, K\}$. The parameter ϵ embodies the trade-off between solution quality and running time. Increasing the value of ϵ makes the dynamic-programming framework more efficient, but it decreases the quality of the solution. As a rule of thumb, a value of ϵ close to 1.2 offers a solution that is often found to be close to the optimal while ensuring a substantial reduction of running time. A *fully-polynomial time approximation scheme* (FPTAS) can also be designed by rounding and scaling the benefit values. However, such an approach does not scale well in practice, and therefore in our experimental evaluation we focus solely on the strongly-polynomial algorithm discussed above, which, while being a heuristic, is empirically found to be remarkably effective, particularly when the budget available for materialization is large.

4.5 Putting everything together

The techniques discussed in this section are incorporated into a comprehensive method, called PEANUT. Its structure is simple: PEANUT is composed of an offline and an online component.

In the offline component, algorithm LRDP is executed once for each internal node as a root, so as to obtain the optimal shortcut potentials corresponding to each combination of root and cost. The resulting computation is then used by algorithm BUDP to compute an optimal packing of multiple node-disjoint shortcut potentials, which is used by the online component of the system to answer inference queries. PEANUT can use the strongly-polynomial algorithm to reduce the time required to identify the materialization.

In the online component, inference queries are processed. First, given a query q, the associated Steiner tree T_q is extracted. PEANUT checks whether there are materialized shortcut potentials that are useful for the query q and, if there are, uses them to reduce the size of T_q and consequently the message-passing cost to answer q.

4.6 Relaxing the node-disjointness constraint

The algorithms presented so far find the optimal set of shortcut potentials under the constraints that the shortcut potential subtrees are node-disjoint. While such a constraint is crucial to make the problem tractable, it limits the budget that can be used for materialization. In particular, in our experiments we observe that PEANUT tends to under-utilize the available budget, resulting in a smaller benefit than what could be possible if the whole budget is used. To overcome this limitation, we propose PEANUT+, a method that combines the ideas discussed above with a simple *greedy packing heuristic.* In PEANUT+, we only run Algorithms 1 and 2 at each possible root and then, from the retrieved shortcut potentials, we greedily select the ones with the largest ratio r of benefit $B(\cdot)$ to cost $\mu(\cdot)$ until the budget is filled.

The online component of PEANUT+ is more complex than the one of PEANUT because overlapping shortcut potentials cannot be simultaneously used. Hence, for each query q, we assemble a conflict graph G_c in which the shortcut potentials that are useful for q are weighted by the corresponding value of r and are joined by an edge if they are overlapping. From G_c, we extract a maximum weighted independent set by using the greedy GWMIN algorithm [29]. In practice, this extra step incurs an overhead, which can be expected to be negligible with respect to the execution time of the queries.

5 EXPERIMENTAL EVALUATION

We present an experimental evaluation of our methods, using benchmark Bayesian network datasets, and comparing PEANUT and PEANUT+ with the strong (but not workload-aware) baseline of Kanagal and Deshpande [21], and with the recent techniques for optimal workload-aware materialization for inference based on variable-elimination [4]. We also study the performance of our methods in the presence of temporally-shifted workloads.

5.1 Experimental setting

We describe in more detail the benchmark Bayesian network datasets we use, the baseline method we compare, and the other settings and design choices for our experimental evaluation.

Datasets. We use 8 real-world Bayesian network datasets, whose statistics are shown in Table 1. The number of parameters in the table refers to the number of probablility values required to define all probabilistic dependencies in the Bayesian network. All datasets are available online.[3] Child [32] is a model for congenital heart-disease diagnosis in new born "blue babies." Hepar II [26] is a model for liver-disorder diagnosis. Andes [11] is used in an intelligent tutoring system for teaching Newtonian physics to students. Hailfinder [2] combines meteorological data and human expertise to forecast severe weather in Northeastern Colorado. TPC-H is a Bayesian network learned from TPC-H benchmark data [33]. Munin [3] is a subnetwork of a Bayesian network model proposed to be exploited in electromyography. PathFinder [17] assists with the diagnosis of lymph-node diseases. Barley [23] supports decision making in growing malting barley. Table 2 shows salient characteristics of the junction trees associated with the datasets. Due to space constraints, we show experimental results only for a subset of datasets. Additional results can be found in the extended version of the paper [10].

Table 1: Summary statistics of Bayesian networks.

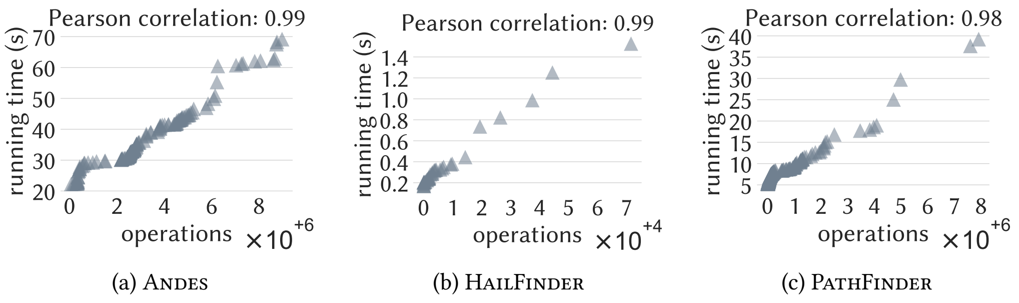
dataset	# nodes	# edges	# parameters	max in-degree
Child	20	25	230	2
Hepar II	70	123	1.4 K	6
Andes	223	338	1.1 K	6
Hailfinder	56	66	2.6 K	4
TPC-H	38	39	355.5 K	2
Munin	186	273	15.6 K	3
PathFinder	109	195	72.1 K	5
Barley	48	84	114 K	4

Table 2: Summary statistics of junction trees.

dataset	# cliques	diameter	treewidth
Child	17	10	3
Hepar II	58	14	6
Andes	175	25	17
Hailfinder	43	14	4
TPC-H	33	16	2
Munin	158	23	11
PathFinder	91	17	6
Barley	36	14	7

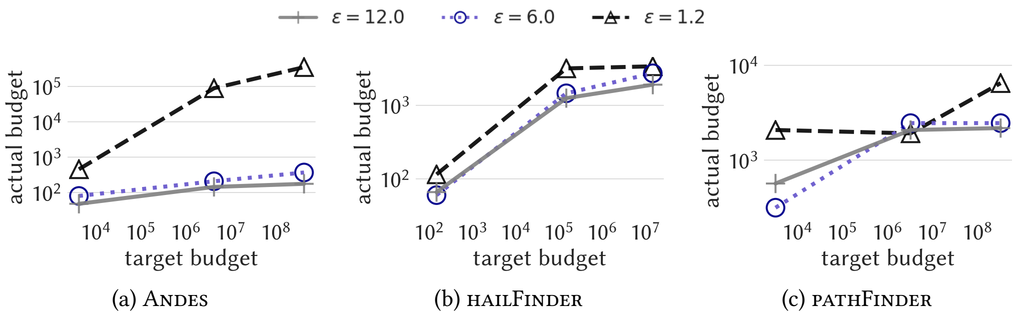
Baselines. We compare our approach against INDSEP [21], which relies on a *tree-partitioning technique* [24] to build a hierarchical index over the junction tree. The index construction hinges on the disk block size, which provides an upper bound on the memory required by each index node. An index node I corresponds to a connected subtree of the junction tree and is associated with a shortcut potential given by the joint-probability distribution of the variables in the separators adjacent to I. Queries are processed by a recursive algorithm, and shortcut potentials prune the recursion tree for some queries. A multi-level approximation scheme for shortcut potentials is also implemented to deal with the scenario in which the size of a shortcut potential exceeds the block size. We furthermore compare PEANUT and PEANUT+ with the optimal materialization for variable elimination that we have developed in our previous work [4]. We refer to this method as VE-n, where n refers to the number of materialized factors. The time and space requirements for the construction and calibration

[3]See https://github.com/martinociaperoni/PEANUT for TPC-H and www.bnlearn.com/bnrepository/ for the rest



(a) Andes (b) HailFinder (c) PathFinder

Figure 3: Running time against cost values for a set of queries processed with the standard junction-tree algorithm.



(a) Andes (b) hailFinder (c) pathFinder

Figure 4: Materialized budget against target budget in PEANUT with different levels of approximation (log-log scale).

of the junction tree (JT) and for the materialization step with the different methods under comparison are shown in Table 4. Note that for the TPC-H, Munin and Barley datasets, calibration of the tree did not terminate after a one-day-long execution (asterisk and NA entries in Table 4). The calibration of the junction tree in principle represents a preliminary step for INDSEP, PEANUT and PEANUT+ as they take as input the calibrated junction tree. However, the calibration process is only necessary to obtain correct answers to inference queries, and it is not needed for comparing exact query processing costs using different materialization strategies which is the goal of our experimental evaluation. Therefore, such datasets are also considered in the experiments. In this case, INDSEP, PEANUT and PEANUT+ take as input the uncalibrated junction tree and, as a consequence, query answers contain erroneous probability values. Nonetheless, the computational burden associated with query processing is not affected by the calibration process.

Query workloads. Since we do not have access to real query workloads for the datasets we use in these experiments, we resort to random sampling to construct synthetic query workloads.

In the *skewed workload*, variables are sampled from a distribution skewed towards the leaves of the tree, i.e., variables have probabilities of being queried for proportional to their distance from the pivot of the junction tree. We generate $N_q = 3\,000$ queries in this way, of which 2 000 are used to estimate benefits, and the remaining 1 000 to assess the materialization.

In the *uniform workload*, variables are sampled uniformly at random. This ensures fairness in the comparison between materialization of junction trees and VE-n. In this case, we use the same $N_q = 250$ queries, all of which contain 1 to 5 query variables, for both optimization and evaluation. As shown in the extended version of the paper [10], the impact of parameter N_q on materialization performance is minor.

Cost values. The standard junction-tree method is prohibitively time-consuming for several datasets; note "NA" entries in Table 4.

Table 3: Offline running times (seconds) for PEANUT and PEANUT+ (in parenthesis) with different approximation levels and INDSEP.

dataset	$\epsilon = 1.2$	$\epsilon = 6$	$\epsilon = 12$	INDSEP
Child	0.028 (0.020)	0.024 (0.022)	0.016 (0.015)	0.004
Hepar II	2.88 (2.81)	1.79 (1.75)	1.30 (1.27)	0.086
Andes	4.7 K (4.6 K)	252 (238)	196 (162)	2.19
Hailfinder	6.71 (5.89)	0.46 (0.38)	0.32 (0.29)	0.039
TPC-H	16.93 (12.23)	0.44 (0.37)	0.24 (0.21)	0.025
Munin	8.8 K (8.4 K)	140 (138)	99.41 (99.03)	2.53
PathFinder	80.42 (73.22)	2.76 (2.71)	2.18 (2.15)	0.25
Barley	86.30 (78.44)	1.67 (1.63)	1.027 (1.018)	0.030

To circumvent this limitation, we measure the performance of a materialization using the total number of operations required to process a query workload. In more detail, given a query q, a cost value $c(v)$ is assigned to all nodes of the Steiner tree T_q associated with q, which could be either cliques or shortcut potentials replacing part of the original Steiner tree T_q. The cost $c(v)$ is the number of operations required to compute the message sent to π_v, or to compute the final answer to the query if v coincides with the pivot of T_q. The cost of q is then the sum of the cost values over the nodes in T_q. We confirm empirically that the assigned cost values align almost perfectly with the corresponding execution times, as shown in Figure 3, where we show the running time against the cost for a set of queries, for three datasets. In Figure 3, we additionally display the Pearson correlation coefficient. The extremely strong correlation clearly provides empirical support for using our cost values to assess the performance of a materialization.

Experiment parameters. Next we discuss the choice of parameters for space budget K, and approximation ϵ.

Table 4: Materialization phase statistics for the experiments comparing PEANUT, PEANUT+, INDSEP and VE-n.

	Disk Space (MB)					Time (seconds)				
dataset	VE-5	JT	INDSEP	PEANUT	PEANUT+	VE-5	JT	INDSEP	PEANUT	PEANUT+
Child	0.025	0.005	0.002	0.002	0.044	0.020	0.035	0.016	34.12	23.013
Hepar II	0.025	0.013	0.011	0.001	1.55	0.040	0.37	0.12	493.86	313.21
Andes	0.62	70	78	55.16	17.7 K	0.84	3.7 K	2.55	31 K	20 K
Hailfinder	0.025	0.071	0.11	0.13	8.28	0.041	0.93	0.056	199.32	78.96
TPC-H*	0.041	23.77	43.19	1.23	483.78	2.76	NA	0.027	320.71	57.83
Munin*	265	2.8 K	361.95	358.16	1 K	253.67	NA	2.57	25.6 K	16.9 K
Pathfinder	0.06	1.37	1.40	0.14	220.79	0.31	302	2.80	1 K	420.32
Barley*	24	186.99	3 K	2.2 K	13.2 K	11.57	NA	0.034	635.051	263.57

Budget. First we note that the two materialization methods we evaluate, PEANUT and INDSEP, do not always make use of the full available budget, as is typically the case with such *discrete-knapsack* constraints, both because the node-disjointess constraint may limit the space that can be materialized and because of the strongly-polynomial approximation, if used. We thus make a distinction between the *target budget K*, i.e., the value of the budget constraint, and the *actual budget*, i.e., the materialized space used by a method. Indicatively, differences between target and actual budget are demonstrated in Figure 4 for PEANUT, for different values of ϵ and a subset of datasets. It is evident that large values of ϵ only take up a small part of the target budget, and, in general, the smaller the value of ϵ, the closer the materialization space is to the target one. Moreover, the target budget K is expressed differently for PEANUT and INDSEP. In the case of PEANUT, the target budget is specified directly as part of the input. The larger the target budget, the larger the materialized budget is expected to be. Instead, in INDSEP, the materialized budget is determined by the block size and cannot be directly controlled. Unlike PEANUT and INDSEP, PEANUT+ can control the actual budget, which is greedily filled. This allows to compare PEANUT+ and INDSEP at (approximately) parity budget. In skewed workload experiments, for INDSEP we consider a large set of possible block-sizes $\{10, 20, 50, 100, 150, 500, 1000, 5 \times 10^3, 5 \times 10^4, 5 \times 10^5, 5 \times 10^6\}$, and we choose the three values leading to the minimum, maximum, and median materialization space. For PEANUT we consider three different target budgets, namely, $\{b_T/10, 10\, b_T, 10000\, b_T\}$, where b_T is the total potential size of the separator in T. In uniform workload experiments, for INDSEP we consider block-size 10^3, and similarly for PEANUT and PEANUT+ we consider target budget $1000\, b_T$.

Approximation. For PEANUT and PEANUT+, the user-specified variable ϵ that controls the trade-off between solution quality and running time is an important parameter. Setting $\epsilon = 1$ gives the optimal solution, but it will be time-consuming. We show results for $\epsilon \in \{1.2, 6, 12\}$. Table 3 shows, for the skewed workload, PEANUT and PEANUT+ wall-clock time for finding the optimal materialization and INDSEP wall-clock time to build the INDSEP data structure. Here, the budget is fixed to $\frac{1}{10} b_T$ for PEANUT, and similarly, to the smallest budget used for INDSEP. Note that PEANUT and PEANUT+ are usually much slower than INDSEP in choosing the shortcut potentials to materialize. However, the offline overhead of our approach is counterbalanced by its superior performance during the online query-processing phase, which is the focus of our work.

Implementation. Experiments are executed on a machine with 2×10 core Xeon E5 2680 v2 2.80 GHz processor and 256 GB memory. All methods have been implemented in Python. Our implementation is available online.[4]

5.2 Query-processing results

We first report results on the cost of processing inference queries.

Skewed workload. In Figure 5, we show the performance gains due to materialization of shortcut potentials achieved by INDSEP and PEANUT+ with different approximation levels when processing the skewed workload of queries. The performance gains are given by the net cost savings obtained by resorting to the shortcut potentials, compared to the case when no shortcut potential is used. The results confirm that PEANUT+ typically outperforms INDSEP, even for large values of the approximation parameter ϵ. As expected, for PEANUT+, the lowest value of $\epsilon = 1.2$ typically yields the greatest cost savings, while $\epsilon = 12$ tends to result in the worst performance. In general, although it is not guaranteed, for both PEANUT and PEANUT+, increasing ϵ tend to decrease savings. Next, we explore the relation between the net cost savings and the diameter of the Steiner tree extracted for query answering. Figure 6 displays the average net cost savings in the skewed workload as a function of the Steiner-tree diameter for INDSEP, PEANUT+ and PEANUT with different values of the approximation parameter ϵ. Here, for each query we take the maximum savings over the considered budgets for INDSEP, PEANUT and PEANUT+. While the figure does not offer a fair comparison between candidate strategies since savings are obtained with largely varying budgets, it reveals that savings grow quickly with the Steiner-tree diameter simply because larger Steiner trees lead to a higher shortcut potential hit rate. The increasing trend is common to all datasets, although the growth rate depends heavily on the characteristics of the materialized shortcut potentials.

Uniform workload. In Figure 7 we compare, in the uniform workload, the total cost for inference based on the junction tree (JT) without additional materialization, INDSEP, VE-n with $n = 5$ materialized factors, PEANUT and PEANUT+ for fixed $\epsilon = 1.2$. In addition, we analyse how the cost varies with the query size $|q|$. As expected, larger queries are more computationally demanding because they yield larger Steiner trees and larger messages. Clearly, for $|q| = 1$, VE-n performs remarkably worse than the other methods. PEANUT+ always leads to the best query processing except for the Munin dataset, in which junction-tree-based approaches exhibit very poor performance. Similarly, PEANUT outperforms INDSEP in all datasets and VE-5 in five datasets.

Finally, it is important to observe that the scale of savings across different datasets is highly variable. In particular, the effectiveness of materialization depends largely on the tree structure.

[4]https://github.com/martinociaperoni/PEANUT

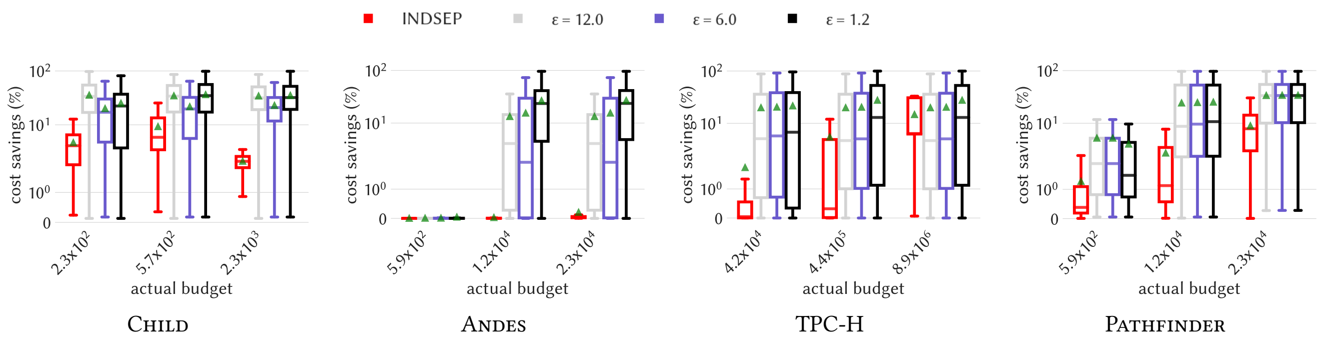


Figure 5: Distribution of cost savings percentage against materialized budget for INDSEP and PEANUT+ with different approximation levels. The average of the distribution is displayed in green. The y-axis is on a logarithmic scale.

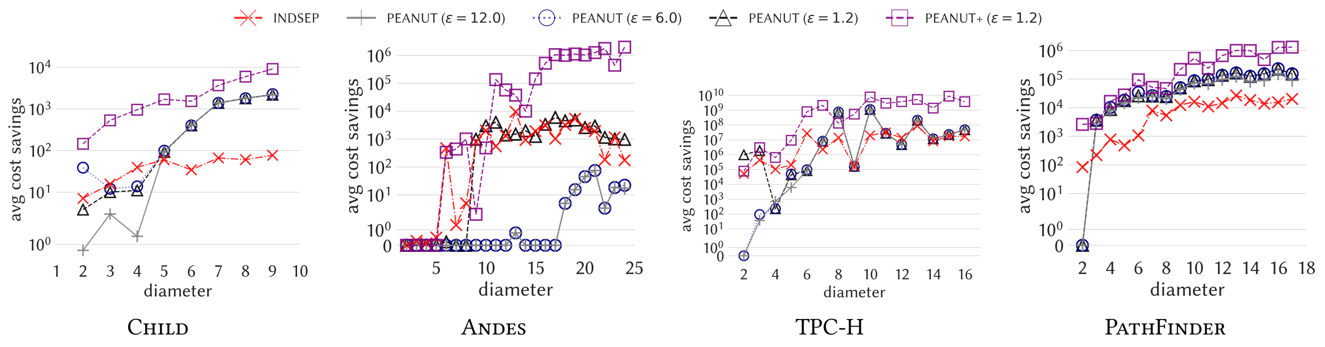


Figure 6: Average cost savings against Steiner-tree diameter for INDSEP, PEANUT+ and PEANUT with different approximation levels. The y-axis is on a logarithmic scale.

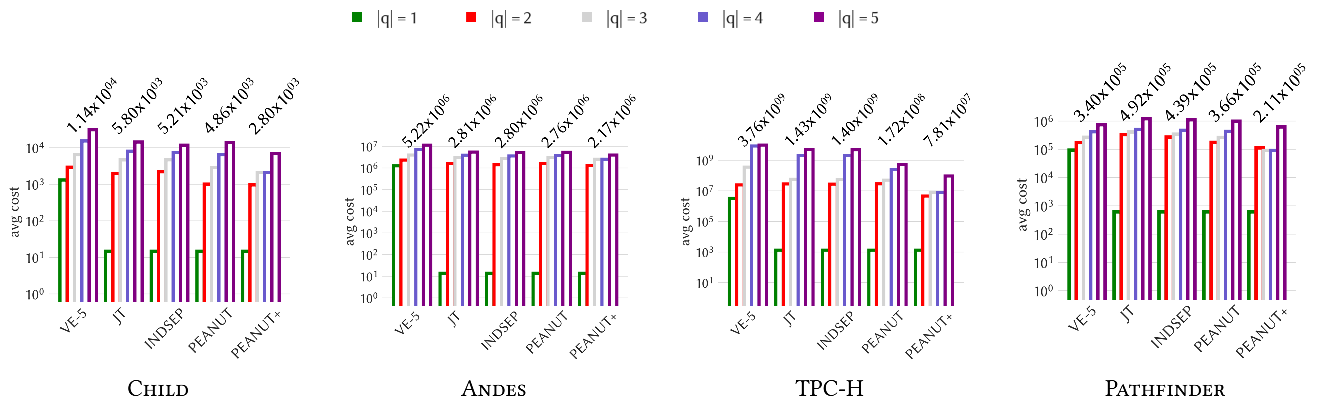


Figure 7: The y-axis shows on logarithmic scale the value of the average query processing cost by query size $|q|$ for different algorithms. Labels report the value of the average query processing cost aggregated over all values of $|q|$ for each method.

As it can be confirmed with a simple regression analysis of the average cost on characteristics of the datasets, for a fixed number of cliques, the benefits of our methods are larger in sparser Bayesian networks leading to junction trees with limited treewidth. Similarly, the diameter of the junction tree appears to have a relevant positive effect on the performance of PEANUT.

5.3 Robustness analysis

We carry out additional experiments to explore the robustness of our system PEANUT with respect to drifts in the query workload distribution. In particular, we explore the setting in which materialization is optimized with respect to a query workload Q and then the results are evaluated on a different query workload Q'. Both Q and Q' are of size 500. When Q is the skewed (uniform) query workload considered in our experiments, Q' consists of queries from the same skewed (uniform) workload in proportion λ and of queries from the uniform (skewed) workload in proportion $1-\lambda$, with varying $\lambda \in [0,1]$. The average cost of processing Q' with and without materialization for different values of λ in the case that Q is the skewed and the uniform workload are shown in Figures 8 and 9, respectively. Here, PEANUT and PEANUT+ parameters are fixed to $K = 10b_T$ and $\epsilon = 1.2$. The

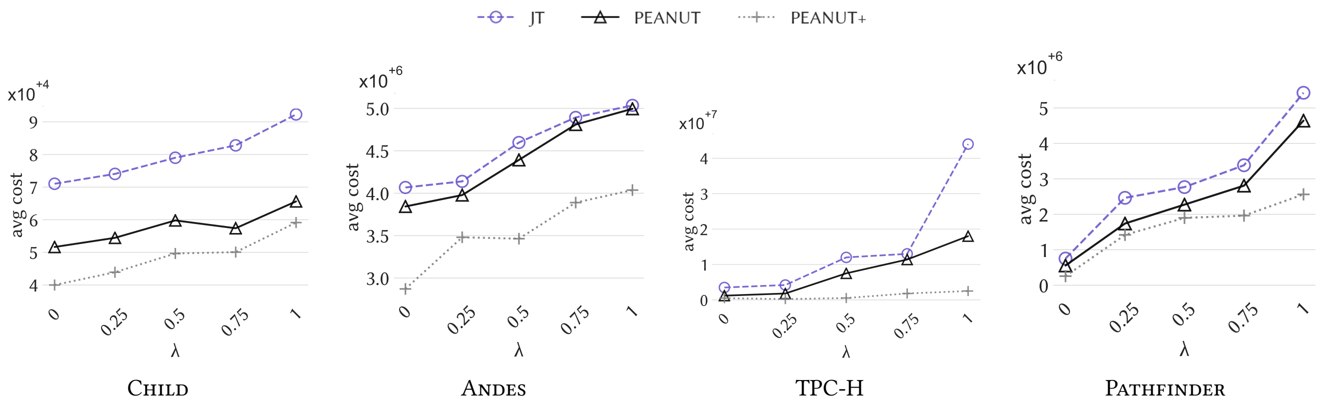


Figure 8: Average cost of processing Q' against proportion λ of queries from Q for the standard junction-tree algorithm (JT), PEANUT and PEANUT+. Here, Q is the skewed workload.

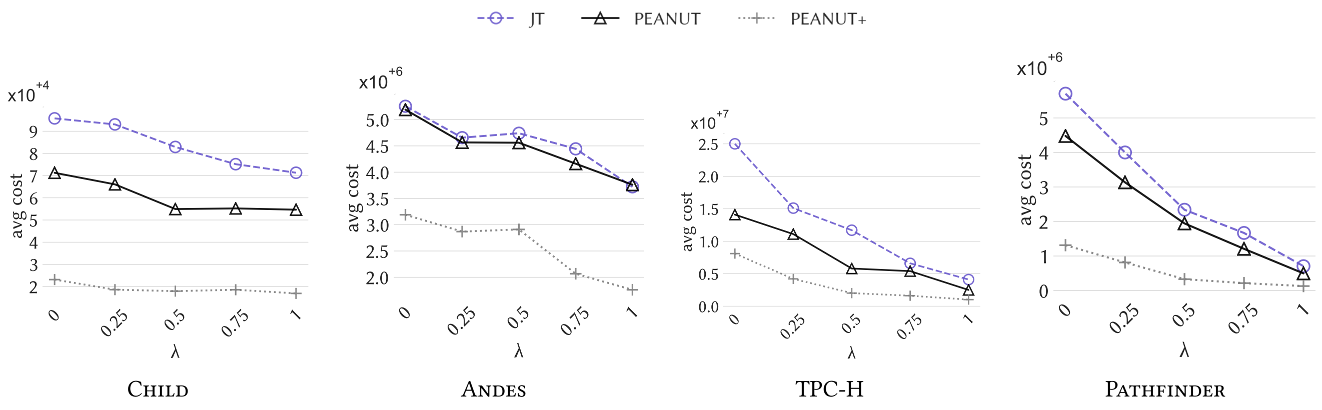


Figure 9: Average cost of processing Q' against proportion λ of queries from Q for the standard junction-tree algorithm (JT), PEANUT and PEANUT+. Here, Q is the uniform workload.

cost savings in Q' are generally not drastically affected by the value of λ. Thus, the results of this experiment highlight that PEANUT and PEANUT+ are fairly robust with respect to drifts in the query workload distribution. Note that the average cost tends to increase with the proportion of skewed queries. This is not surprising since skewed queries are more likely to correspond to Steiner trees of large diameter. This trend is observed in all datasets except for HeparII, in which, although skewed queries are as usual associated with larger Steiner tree diameters, they also favour query variables with smaller cardinality. Despite the robustness of the proposed approach, it is appropriate to recompute the materialization either periodically over time or whenever there is sufficient evidence that the query distribution has changed significantly.

6 CONCLUSIONS

We presented a novel technique to accelerate inference queries over Bayesian networks using the junction-tree algorithm. Our approach builds on the idea introduced by Kanagal and Deshpande [21] to materialize shortcut potentials over the junction tree. However, unlike their work, we have framed the problem of choosing shortcut potentials to precompute and materialize as a workload-aware optimization problem. In particular, we have formulated the problems of selecting a single optimal shortcut potential and an optimal set of shortcut potentials in view of evidence provided by historical query logs. We have proven hardness results for these problems, and we have developed a method, called PEANUT, consisting of a two-level dynamic-programming framework that tackles the problems in pseudo-polynomial time. We have additionally introduced a strongly-polynomial algorithm that trades solution quality with speed, as well as PEANUT+, a simple modification of PEANUT, which ensures a better utilization of the available space budget. Extensive experimental evaluation confirms the effectiveness of our materialization methods in reducing the computational burden associated with a given query workload and the superiority with respect to previous work. Interesting avenues to explore for future work include extending our techniques to allow for node-overlapping shortcut-potential subtrees in a more principled way, or studying a similar approach for optimal materialization of intermediate computations to speed up inference based on a different, possibly approximate, algorithm, such as *loopy belief propagation*.

ACKNOWLEDGMENTS

This research is supported by the Academy of Finland projects AIDA (317085) and MLDB (325117), the ERC Advanced Grant REBOUND (834862), 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] Mahmoud Abo Khamis, Hung Q Ngo, and Atri Rudra. 2016. FAQ: questions asked frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 13–28.

[2] Bruce Abramson, John Brown, Ward Edwards, Allan Murphy, and Robert L Winkler. 1996. Hailfinder: A Bayesian system for forecasting severe weather. *International Journal of Forecasting* 12, 1 (1996), 57–71.

[3] Steen Andreassen, Finn V Jensen, Stig Kjær Andersen, B Falck, U Kjærulff, M Woldbye, AR Sørensen, A Rosenfalck, and F Jensen. 1989. MUNIN: an expert EMG Assistant. In *Computer-aided electromyography and expert systems*. Elsevier, 255–277.

[4] Cigdem Aslay, Martino Ciaperoni, Aristides Gionis, and Michael Mathioudakis. 2021. Workload-aware materialization for efficient variable elimination on Bayesian networks. In *IEEE International Conference on Data Engineering (ICDE)*. IEEE, 1152–1163.

[5] Christopher M Bishop. 2006. *Pattern recognition and machine learning*. Springer.

[6] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseok Shim. 1995. Optimizing queries with materialized views. In *Proceedings of the Eleventh International Conference on Data Engineering*. IEEE, 190–200.

[7] Mark Chavira and Adnan Darwiche. 2005. Compiling Bayesian networks with local structure. In *IJCAI*, Vol. 5. 1306–1312.

[8] Mark Chavira and Adnan Darwiche. 2007. Compiling Bayesian Networks Using Variable Elimination.. In *IJCAI*. 2443–2449.

[9] Geon Cho and Dong X Shaw. 1997. A depth-first dynamic programming algorithm for the tree knapsack problem. *INFORMS Journal on Computing* 9, 4 (1997), 431–438.

[10] Martino Ciaperoni, Cigdem Aslay, Aristides Gionis, and Michael Mathioudakis. 2021. Workload-Aware Materialization of Junction Trees. 2110.03475.pdf. https://arxiv.org/pdf/

[11] Cristina Conati, Abigail S Gertner, Kurt VanLehn, and Marek J Druzdzel. 1997. On-line student modeling for coached problem solving using Bayesian networks. In *User Modeling*. Springer, 231–242.

[12] Robert G Cowell, Philip Dawid, Steffen L Lauritzen, and David J Spiegelhalter. 2006. *Probabilistic networks and expert systems: Exact computational methods for Bayesian networks*. Springer Science & Business Media.

[13] Adnan Darwiche. 2003. A differential approach to inference in Bayesian networks. *Journal of the ACM (JACM)* 50, 3 (2003), 280–305.

[14] Rina Dechter. 1999. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence* 113, 1-2 (1999), 41–85.

[15] Amol Deshpande, Lise Getoor, and Prithviraj Sen. 2009. Graphical models for uncertain data. *Managing and Mining Uncertain Data* (2009), 77–105.

[16] Lise Getoor, Benjamin Taskar, and Daphne Koller. 2001. Selectivity estimation using probabilistic models. In *ACM SIGMOD Record*, Vol. 30. 461–472.

[17] David Earl Heckerman, Eric J Horvitz, and Bharat N Nathwani. 1990. *Toward normative expert systems: The Pathfinder project*. Knowledge Systems Laboratory, Stanford University Stanford.

[18] Max Henrion. 1988. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In *Machine Intelligence and Pattern Recognition*. Vol. 5. Elsevier, 149–163.

[19] Finn V Jensen et al. 1996. *An introduction to Bayesian networks*. Vol. 210. UCL press London.

[20] David S Johnson and KA Niemi. 1983. On knapsacks, partitions, and a new dynamic-programming technique for trees. *Mathematics of Operations Research* 8, 1 (1983), 1–14.

[21] Bhargav Kanagal and Amol Deshpande. 2009. Indexing correlated probabilistic databases. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 455–468.

[22] Mahmoud Abo Khamis, Hung Q Ngo, and Atri Rudra. 2017. Juggling functions inside a database. *ACM SIGMOD Record* 46, 1 (2017), 6–13.

[23] Kristian Kristensen and Ilse Rasmussen. 2002. The use of a Bayesian network in the design of a decision support system for growing malting barley without use of pesticides. *Computers and Electronics in Agriculture* 33 (03 2002), 197–217.

[24] Sukhamay Kundu and Jayadev Misra. 1977. A linear tree partitioning algorithm. *SIAM J. Comput.* 6, 1 (1977), 151–154.

[25] Steffen L Lauritzen and David J Spiegelhalter. 1988. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society: Series B (Methodological)* 50, 2 (1988), 157–194.

[26] Agnieszka Onisko. 2003. Probabilistic causal models in medicine: Application to diagnosis of liver disorders. In *Ph. D. dissertation, Inst. Biocybern. Biomed. Eng., Polish Academy Sci., Warsaw, Poland*.

[27] Judea Pearl. 2014. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier.

[28] Hoifung Poon and Pedro Domingos. 2011. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. IEEE, 689–690.

[29] Shuichi Sakai, Mitsunori Togasaki, and Koichi Yamazaki. 2003. A note on greedy algorithms for the maximum weighted independent set problem. *Discrete applied mathematics* 126, 2-3 (2003), 313–322.

[30] Natthawut Samphaiboon and Y Yamada. 2000. Heuristic and exact algorithms for the precedence-constrained knapsack problem. *Journal of optimization theory and applications* 105, 3 (2000), 659–676.

[31] Maximilian Schleich, Dan Olteanu, Mahmoud Abo Khamis, Hung Q Ngo, and XuanLong Nguyen. 2019. A layered aggregate engine for analytics workloads. In *Proceedings of the 2019 International Conference on Management of Data*. 1642–1659.

[32] David J Spiegelhalter, A Philip Dawid, Steffen L Lauritzen, and Robert G Cowell. 1993. Bayesian analysis in expert systems. *Statistical science* (1993), 219–247.

[33] Kostas Tzoumas, Amol Deshpande, and Christian S Jensen. 2013. Efficiently adapting graphical models for selectivity estimation. *The VLDB Journal* 22, 1 (2013), 3–27.

[34] Nevin L Zhang and David Poole. 1994. A simple approach to Bayesian network computations. In *Proc. of the Tenth Canadian Conference on Artificial Intelligence*.

[35] Nevin Lianwen Zhang and David Poole. 1996. Exploiting causal independence in Bayesian network inference. *Journal of Artificial Intelligence Research* 5 (1996), 301–328.