Lightweight Graphical Models for Selectivity Estimation Without Independence Assumptions

Kostas Tzoumas
Aalborg University
Denmark
kostas@cs.aau.dk

Amol Deshpande
University of Maryland
College Park, MD, USA
amol@cs.umd.edu

Christian S. Jensen
Aarhus University
Denmark
csj@cs.au.dk

ABSTRACT

As a result of decades of research and industrial development, modern query optimizers are complex software artifacts. However, the quality of the query plan chosen by an optimizer is largely determined by the quality of the underlying statistical summaries. Small selectivity estimation errors, propagated exponentially, can lead to severely sub-optimal plans. Modern optimizers typically maintain one-dimensional statistical summaries and make the attribute value independence and join uniformity assumptions for efficiently estimating selectivities. Therefore, selectivity estimation errors in today's optimizers are frequently caused by missed correlations between attributes. We present a selectivity estimation approach that does not make the independence assumptions. By carefully using concepts from the field of graphical models, we are able to factor the joint probability distribution of all the attributes in the database into small, usually two-dimensional distributions. We describe several optimizations that can make selectivity estimation highly efficient, and we present a complete implementation inside PostgreSQL's query optimizer. Experimental results indicate an order of magnitude better selectivity estimates, while keeping optimization time in the range of tens of milliseconds.

1. INTRODUCTION

Real-world data sets often exhibit substantial skew and strong correlations between their attributes. For such data sets, the independence assumptions made by traditional DBMSs often result in substantial estimation errors during query optimization. Propagation of such errors can lead an optimizer to choose sub-optimal query plans [10]. Specifically, aiming at simplicity and low overhead in cost estimation, the original System-R paper [16] made three simplifying assumptions regarding the statistical summaries of data:

Uniform distribution assumption (Uniform): The values of an attribute $R.X$ are assumed to be uniformly distributed across its active domain $\mathrm{Dom}(R.X)$. This allows the approximation $\Pr(R.X = x) \approx 1/|\mathrm{Dom}(R.X)|$.

Attribute value independence assumption (AttrInd): Attributes are considered independent. For attributes $R.X$ and $R.Y$ of relation R, this allows the approximation $\Pr(R.X = x, R.Y = y) \approx \Pr(R.X = x)\Pr(R.Y = y)$.

Join predicate independence assumption (JoinInd): This is also called the join uniformity assumption in the literature. For join predicates $R.a = S.a$, and $S.b = T.b$, this assumption allows the approximation $\Pr(R.a = S.a, S.b = T.b) \approx \Pr(R.a = S.a)\Pr(S.b = T.b)$. Further, for a join predicate $R.a = S.a$ and an attribute $R.X$, the assumption allows the approximation: $\Pr(R.a = S.a, R.X = x) \approx \Pr(R.a = S.a)\Pr(R.X = x)$.

The third assumption is implied by the second. However, due to its importance, we discuss it separately. It is well known that these assumptions can lead to substantial estimation errors [8, 9], which, when propagated in an exponential fashion through join plans [10], can lead the query optimizer to recommend a sub-optimal plan [14]. Thus, a great body of research has been devoted to avoiding these assumptions, while keeping the overhead of selectivity estimation reasonable.

Uniform has been successfully addressed in a large body of research on *attribute-level* synopses, including one-dimensional histograms and wavelets [9, 13]. This research attempts to approximate the real probability distribution of an attribute, $P(X)$, in limited space using another distribution $\hat{P}(X)$, called a synopsis. Then, **Uniform** can be dropped by approximating $\Pr(X = x)$ as $\hat{P}(x)$. Modern DBMSs maintain such synopses, which allow for accurate estimation of simple predicates. Thus, the largest estimation errors in today's optimizers are due to the independence assumptions **AttrInd** and **JoinInd** [8].

To avoid **AttrInd**, *table-level* synopses such as multidimensional histograms [15] attempt to estimate the *joint* probability distribution of two or more variables. Attempting to capture the full joint distribution of the attributes of a table causes an exponential blowup of storage space and complexity. Deshpande et al. [6] and Getoor et al. [7] propose using the principle of *conditional independence* to factor the full joint distribution into a set of smaller distributions to address this problem.

Table-level synopses do not adequately address the **JoinInd** assumption. Assume a join predicate $R.a = S.b$ between two relations R and S, and an attribute $R.X$. In principle, one could construct a three-dimensional synopsis that approximates the joint distribution $P(R.X, R.a, S.b)$ and estimate the joint selectivity of the join and the selection $R.X = x$ correctly. This approach suffers from the problem that join attributes are typically keys, which yields distributions that are hard to approximate. One solution is to use binary random variables, called *join indicators* and construct a (decomposed) probability distribution on all the attributes and join indicators in the database [7]. This is the approach we follow in this paper. Our work falls in the category of *schema-level* synopses [1, 7, 17]. A statistical summary is constructed that approx-

Proceedings of the VLDB Endowment, Vol. 4, No. 11

imates the joint probability distribution of all attributes and join predicates in the database.

The theoretical underpinnings of our method stem from the field of probabilistic graphical models [4]. We use the notion of conditional independence to factor the full joint probability distribution of the database into a set of smaller marginal distributions. The exponential blow-up of space and time is thus avoided. Then, we use an adaptation of the junction tree algorithm [11] to efficiently compute joint probabilities of selection and join predicates. Graphical models have been used before to model the probability distribution induced by a database [7]. In contrast to previous work, we aim for a practical solution with low overhead. To this end, we make several contributions.

First, we carefully choose a *fixed* graphical model structure that exploits the inherent conditional independencies in a relational database, and preserves the most important dependencies, while limiting the dimensionalities of all stored probability distributions to two variables. Second, we develop a custom model-creation algorithm that significantly outperforms off-the-shelf algorithms (e.g., [5]) for large databases. Third, we present a novel dynamic programming algorithm for selectivity estimation that exploits the nature of selectivity estimation requests made by a query optimizer. Fourth, we present a full implementation of our algorithms in the PostgreSQL query optimizer. Finally, experimental results on our prototype suggest orders of magnitude better selectivity estimates, while keeping the estimation overhead within tens of milliseconds.

The rest of this paper is organized as follows. Section 2 reviews background material relating to graphical models. Section 3 presents the process of modeling a database with a graphical model, and discusses the assumptions we make along the way. It also discusses the specifics of our model construction algorithm. Section 4 discusses how the model is used during query optimization. Section 5 reports on our experimental study, and Section 6 presents related work. Finally, Section 7 concludes and offers research directions.

2. PRELIMINARIES

Assume the joint probability distribution of n random variables, $P_{\mathcal{D}}(X_1, \ldots, X_n)$. The cost of storing and extracting probabilities from the distribution grows exponentially with respect to n. The field of graphical models [4] offers ways to approximate the distribution $P_{\mathcal{D}}$ via another distribution $P_{\mathcal{M}}$ which can be expressed as a product of low-dimensional distributions. Thus, the exponential blow-up of storage and computation cost is avoided.

To factor a distribution, the notions of independence and conditional independence are used. When two variables X_i and X_j are independent (denoted $X_i \perp X_j$), their joint distribution can be factored as $P(X_i, X_j) = P(X_i)P(X_j)$. When X_i and X_j are conditionally independent given a third variable X_k (denoted $X_i \perp X_j | X_k$), the joint distribution of all three variables can be factored as $P(X_i, X_j, X_k) = P(X_i, X_k)P(X_j, X_k)/P(X_k)$. Intuitively, $X_i \perp X_j | X_k$ when knowledge about X_j does not convey further information about X_i once we know X_k.

Bayesian networks (BNs) offer a graphical representation of a set of (conditional) independencies. The *qualitative* component of a BN is a directed acyclic graph, as the one shown in Figure 1(b) (we explain the meaning of the variables later). The nodes of the graph are the random variables of the distribution $P_{\mathcal{D}}$ that we are trying to approximate. An edge $X_i \rightarrow X_j$ in the BN denotes that the value of X_i stochastically influences the value of X_j. Lack of an edge between two variables does not imply independence. For example, consider a chain of two edges $X_i \rightarrow X_k \rightarrow X_j$. Here, X_i influences X_j indirectly through X_k. Once X_k is known, X_i and X_j become independent. If on the other hand X_k is not known, an interaction between X_i and X_j exists. In fact, the chain of interactions represents the conditional independence $X_i \perp X_j | X_k$. Denote by $\text{Pa}(X)$ the parents of the variable X in the BN, and by $\text{NonDesc}(X)$ the non-descendants of variable X, i.e., the variables that cannot be reached from X following the directed edges of the BN. The full set of conditional independencies encoded by the BN is $X \perp \text{NonDesc}(X) | \text{Pa}(X)$ for all X.

The *inference problem* consists of computing marginal distributions of the form $P(X_i | X_j = x_j)$ from a BN. The junction tree algorithm [11] is an efficient solution to the inference problem that uses an intermediate representation called a junction tree (JT). The transformation of a BN to a JT is done in three steps. First, the *moral graph* is constructed; the moral graph is an undirected graph that contains the same nodes as the BN and contains an undirected edge for each directed edge in the BN. Additionally, two nodes that share a child in the BN are connected. Figure 1(c) shows the moral graph of the BN of Figure 1(b), if the dotted edges are excluded. To be able to construct a JT from the moral graph, the latter must be *chordal*: It cannot contain a cycle with more than three nodes without a chord, i.e., an edge that connects two nonconsecutive nodes in the cycle. If the moral graph is not chordal, an intermediate step called *triangulation* precedes the JT construction. It adds edges to the moral graph to make it chordal. The dotted edges in Figure 1(c) are the edges added by the triangulation process.

The nodes of the junction tree are the maximal cliques of the moral graph. An edge between two cliques is annotated with a *separator*, a node that contains the intersection of the cliques' variables. Figure 1(d) shows a valid junction tree for the moral graph of Figure 1(c). A junction tree represents a factorization of the joint distribution of all variables $P_{\mathcal{D}}$. The only distributions that need to be kept are the marginals of the cliques and separators. For example, for the central clique $\mathbf{C} = \{l\_sdate, l\_cdate, o\_odate\}$ of the junction tree in Figure 1(d), we need to store the joint distribution $P(l\_sdate, l\_cdate, o\_odate)$. The distribution $P_{\mathcal{M}}$ induced by the junction tree (that approximates $P_{\mathcal{D}}$) can be expressed as the product of the clique marginals divided by the separator marginals: $P_{\mathcal{M}} = \prod_{\mathbf{C}} P(\mathbf{C}) / \prod_{\mathbf{S}} P(\mathbf{S})$. It is easy to extract a marginal distribution from a JT if all variables belong to the same clique. To extract the marginal $P(l\_sdate, o\_odate)$ from the JT of Figure 1(d), we just need to sum out the variable l_cdate from the central clique:

$$P(l\_sdate, o\_odate) = \sum_{l\_cdate} P(l\_sdate, l\_cdate, o\_odate).$$

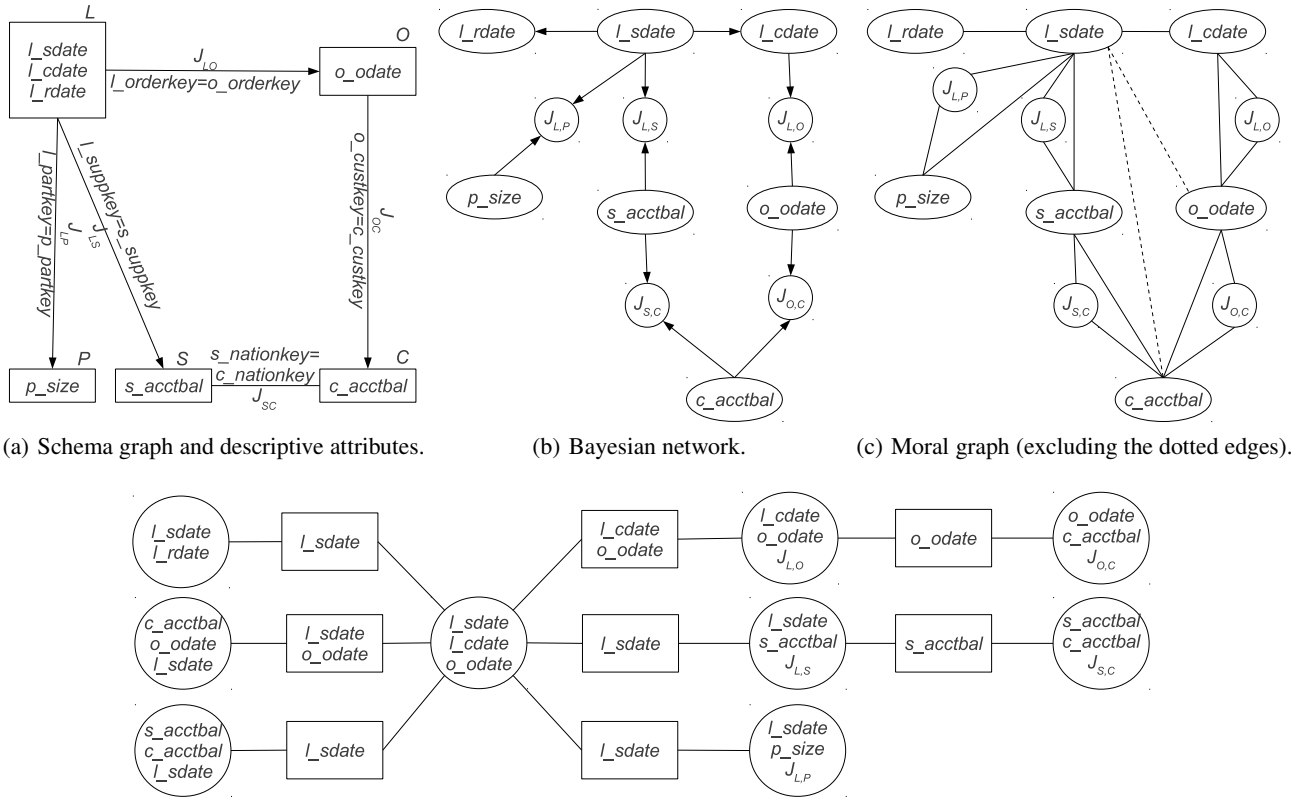
If the variables do not belong to the same clique, we need to multiply clique distributions. For example, to extract the marginal $P(l\_cdate, l\_rdate)$ we need to multiply the distributions of the central and the upper-left clique and divide by the separator before summing out the unnecessary variables:

$$P(l\_cdate, l\_rdate) = \sum_{l\_sdate, o\_odate} \frac{P(l\_sdate, l\_cdate, o\_odate) P(l\_sdate, l\_rdate)}{P(l\_sdate)}$$

Junction trees are effective for models with low *treewidth* τ, which is the cardinality of the largest clique minus 1. Thus, to achieve an efficient factorization it is essential to keep the clique cardinalities small.

3. MODEL SELECTION AND CONSTRUCTION

We aim to define a succinct statistical model of a database. The trade-off is between the expressiveness of the model in terms of



(a) Schema graph and descriptive attributes. (b) Bayesian network. (c) Moral graph (excluding the dotted edges).

(d) Junction tree.

Figure 1: A model constructed on a subset of the TPC-H schema graph along with seven descriptive attributes. The Bayesian network, the moral graph, and the junction tree representations of the model are shown. The dotted edges in (c) are the edges added to the moral graph during triangulation.

capturing complex statistical relationships versus the storage cost to maintain the model and the complexity of performing inference.

We follow two design guidelines. First, the model should capture the most important correlations that influence the sizes of intermediate relations during query optimization. Second, efficiency is important: we aim for estimation runtime performance similar to what is offered by commercial DBMSs.

3.1 The probability distribution of the database

The first step is to define the probability distribution that we aim to approximate. We assume that we are given a schema $\mathcal{S} = \{R_1, \ldots, R_n\}$ and a query workload $\mathcal{W} = \{q_1, \ldots, q_m\}$. First, we define the *schema graph* $\mathcal{G}_S(\mathcal{V}_S, \mathcal{E}_S)$ as a graph that contains one vertex for each relation R_i, and an edge for each possible join relationship between two relations. Note that there can be multiple edges between two relations, signifying multiple join predicates, as well as edges whose endpoints are the same vertex, indicating possible self-joins. Our assumption is that the schema graph is given as input to the model construction algorithm. In reality, it can be discovered using the workload $\mathcal{W}$ and possible information about foreign keys. Figure 1(a) shows an example schema graph for a subset of the TPC-H schema. Boxes are relations (that also contain attributes of interest), and the edges indicate possible join predicates between relations. In Figure 1(a), directed edges signify key-foreign key relationships. However, we note that our approach is general, and applies to arbitrary join predicates.

Descriptive attributes: For an attribute of interest in the database (e.g., attributes that appear in "where" clauses in the queries), we define a random variable called a descriptive attribute. We denote by $\mathcal{A}(\mathcal{S}, \mathcal{W})$ the set of all descriptive attributes. For example, the schema graph of Figure 1(a) defines the set $\mathcal{A}(\mathcal{S}, \mathcal{W})$ that contains the descriptive attributes *l_sdate*, *l_cdate*, *l_rdate*, *o_odate*, *p_size*, *s_acctbal*, *c_acctbal*.

Join indicators: For each edge in the schema graph, we define a *join indicator*, a binary random variable that captures the event that two random tuples from two relations join. For example, the join indicator J_{LO} in Figure 1(a) is a binary random variable capturing the event *l_orderkey=o_orderkey* for two L and O tuples. A join indicator takes the value **T** if two tuples join with the particular join predicate, and it otherwise takes the value **F**. Denote by $\mathcal{J}(\mathcal{S}, \mathcal{W})$ the set of all the join indicators defined by the schema graph $\mathcal{G}_S$. For example, the set $\mathcal{J}(\mathcal{S}, \mathcal{W})$ defined by the schema graph of Figure 1(a) contains the join indicators J_{LO},J_{OC},J_{LP},J_{LS}, and J_{SC}.

The universal relation: We define the universal relation $\mathcal{U}$ in two steps. Consider the Cartesian product $\mathcal{C}(\mathcal{S})$ of all the relations in the database. We initially add one column to the Cartesian product for each join indicator. The value of each join indicator in each row is **T** if the join predicate is true for the particular tuple of the Cartesian product, and it is false otherwise. Denote by $\mathcal{CJ}$ this enhanced relation. Then, we project on the set of join indicators and the set of descriptive attributes $\mathcal{A}(\mathcal{S}, \mathcal{W})$, i.e., the attributes of the database that will be included in the model: $\mathcal{U}(\mathcal{S}, \mathcal{W}) = \pi_{\mathcal{A}(\mathcal{S},\mathcal{W}) \cup \mathcal{J}(\mathcal{S},\mathcal{W})}(\mathcal{CJ}(\mathcal{S}))$. For example, the universal relation extracted from the schema graph of Figure 1(a) contains the join indicators J_{LO},J_{OC},J_{LP},J_{LS}, and J_{SC}, and the descriptive attributes *l_sdate*, *l_cdate*, *l_rdate*, *o_odate*, *p_size*, *s_acctbal*, and *c_acctbal*. See Appendix A for an example of a small database

of two relations and the derived universal relation, along with the produced graphical model and the probability distributions.

3.2 Model granularity

The universal relation $\mathcal{U}$ defines a set of random variables $\mathcal{V} = \mathcal{A} \cup \mathcal{J}$, as well as a probability distribution $P_{\mathcal{U}}(\mathcal{V})$. This distribution represents the "ground truth," the distribution that we will try to approximate using a graphical model. The graphical models literature offers algorithms to learn a graphical model structure from data [4, 5]. However, there are two problems in using an off-the-shelf algorithm to construct our graphical model structure. First, large cliques may be created. We aim to keep the clique size small, so that only one- or two-dimensional histograms will be used to store the cliques. Second, the way that the relation $\mathcal{U}$ was created induces some independencies between its attributes that are not exploited by generic algorithms. Finally, the running times of these algorithms are usually prohibitive for use in large databases.

Relational independencies: Recall that the universal relation $\mathcal{U}$ is a projection of an "enhanced" Cartesian product $\mathcal{CJ}$ of all the relations in the database. Using a frequentist interpretation of probability (which is appropriate for selectivity estimation [7]), a set of independencies hold in $P_{\mathcal{U}}$ for every possible database instance.

Theorem 1. *Consider the relations $R \neq S \neq T \neq U \in \mathcal{S}$. The following independencies hold in $P_{\mathcal{U}}$:*

1. *Descriptive attributes belonging to different relations are independent: $P_{\mathcal{U}}(R.X, S.Y) = P_{\mathcal{U}}(R.X)P_{\mathcal{U}}(S.Y)$.*
2. *Join indicators are independent if they do not share a relation: $P_{\mathcal{U}}(J_{RS}, J_{TU}) = P_{\mathcal{U}}(J_{RS})P_{\mathcal{U}}(J_{TU})$.*
3. *A descriptive attribute and a join indicator are independent if they do not share a relation: $P_{\mathcal{U}}(J_{RS}, T.X) = P_{\mathcal{U}}(J_{RS})P_{\mathcal{U}}(T.X)$.*

PROOF. See Appendix F.1

Fixed model structure: We exploit these independencies to develop a simple, fixed model structure that does, however, capture the most relevant statistical correlations for the purpose of selectivity estimation.

Assume that there is only one descriptive attribute per relation. We allow only one type of edge in our Bayesian network: an edge from a descriptive attribute $R.X$ to the corresponding join indicator J_{RS}: $R.X \rightarrow J_{RS}$. When multiple descriptive attributes are present, we restrict their dependencies within relations. Specifically, there are no edges $R.X \rightarrow S.Y$, and within a relation R we do not allow a "common effect" structure, where an attribute has two parents (e.g., $R.X \rightarrow R.Y \leftarrow R.Z$). Then, the "local" Bayesian network for a given relation R is a forest of directed rooted trees. Figure 1(b) shows a Bayesian network for the schema graph of Figure 1(a) that respects the above constraints.

The following theorem guarantees the validity of our fixed model structure (see Appendix F.2 for the proof).

Theorem 2. *The fixed model structure respects the relational independencies and produces a valid (acyclic) Bayesian network.*

It is clear that the introduced fixed model structure is a design decision, and that it may encode conditional independencies that do not necessarily hold in an arbitrary database distribution $P_{\mathcal{U}}$, i.e., $P_{\mathcal{M}}$ (the distribution induced by the graphical model) is an approximation of $P_{\mathcal{U}}$. We proceed to discuss the intuition behind our decisions and its limitations.

First, allowing edges between attributes of different relations (of the form $R.X \rightarrow S.Y$) would violate the relational independencies. The same holds for edges of the forms $R.X \rightarrow J_{ST}$ and $J_{RS} \rightarrow J_{TU}$. Therefore, disallowing these edges does not result in missed correlations.

Note that the lack of an edge between two attributes does not imply independence. Consider for example two relations R and S that join with each other and have one descriptive attribute each: $R.X$ and $S.Y$. Our Bayesian network would have an edge from $R.X$ to J_{RS} and from $S.Y$ to J_{RS}. Although the BN (correctly) models the two descriptive attributes as independent, conditioned on the event that $J_{RS} = \mathbf{T}$ (i.e., when the two relations are joined), the two variables are not independent of each other.

Second, if we were to allow a common effect structure within the local Bayesian network of a relation (of the form $X_1 \rightarrow X_2 \leftarrow X_3$) we would need to store and manipulate three- or higher- dimensional probability distributions. This would incur significant cost during selectivity estimation. The same is true if we were to allow a join indicator to have more than two parents. These restrictions do result in missed dependencies, however the resulting selectivity estimation errors are often not significant as demonstrated by our experimental results.

3.3 Model construction

The model construction algorithm takes as input a set of descriptive attributes $\mathcal{A}$ and a set of join indicators $\mathcal{J}$, and constructs a Bayesian network to be used for selectivity estimation (actually our algorithm directly constructs the moral graph as discussed below). Its final output is a junction tree that is stored in the DBMS catalog. Denote by $\mathcal{A}(R)$ the subset of descriptive attributes that belong to relation R.

As discussed in the previous section, we restrict the space of possible Bayesian networks by imposing two restrictions. First, a join indicator J_{ij} has at most two parents, one from relation R_i and one from relation R_j. Second, within a relation R_i, the "local" Bayesian network of the attributes $\mathcal{A}(R_i)$ is a directed rooted tree. Thus, the model construction algorithm needs to find the "best" parents of each join indicator, and a local Bayesian network for each relation. The concatenation of these is the Bayesian network of the database. Then off-the-shelf algorithms for moralization and triangulation [2] are applied to construct a chordal moral graph, from which a junction tree can be derived in a standard manner [4]. Figures 1(b),1(c), and 1(d) show respectively the Bayesian network, the moral graph, and the junction tree constructed for the schema graph of Figure 1(a).

Due to the fixed structure of our model, the only edges added by moralization are between two parents of a join indicator. Thus, we can operate on the moral graph directly. Algorithm 1 describes the construction of the moral graph of the database. Initially, the "local" moral graph, $\mathcal{MG}_R$, of each relation is constructed by first determining the dependence measure between each pair of its attributes and then constructing the maximum spanning tree using these measures as weights [3]. The dependence measure we use is mutual information:

$$I(X;Y) = \sum_x \sum_y P(x,y) \log\left(\frac{P(x,y)}{P(x)P(y)}\right).$$

Then the two best predictors (if they exist) for each join indicator are found in a similar fashion.

Consider a relation R with attributes $A_1, \ldots, A_n$. In order to test dependence between A_i and A_j, we need to issue the query

```
select A_i, A_j, count(*) from R group by A_i, A_j.
```

Algorithm 1 Construction of the moral graph

```
1: function CONSTRUCT-MG(A,J,R)
2:     MG = (A ∪ J, {})
3:     for R in R do
4:         MG_R = (A_R, {})
5:         for A_i in A(R) do
6:             for A_j in A(R) do
7:                 Add (A_i − A_j) to MG_R with weight I(A_i : A_j)
8:         MG_R = MAXIMUM-SPANNING-TREE(MG_R)
9:         Add edges of MG_R to MG
10:    for J_ij in J do
11:        A_i^best = arg max_{A_i ∈ A(R_i) ∧ I(A_i:J_ij)>0} I(A_i : J_ij)
12:        A_j^best = arg max_{A_j ∈ A(R_j) ∧ I(A_j:J_ij)>0} I(A_j : J_ij)
13:        if A_i^best ≠ null then
14:            Add (A_i^best − J_ij) to MG
15:        if A_j^best ≠ null then
16:            Add (A_j^best − J_ij) to MG
17:        if A_i^best ≠ null ∧ A_j^best ≠ null then
18:            Add (A_i^best − A_j^best) to MG
19:    return MG
```

The output of the above query is normalized to obtain the joint distribution $P(A_i, A_j)$. Since A_i and A_j will likely form a clique in the final junction tree, we save the distribution when we test dependence to avoid later recomputation. Once the local moral graph of the relation is found, we can delete the distributions of variable pairs that do not belong in it. Therefore, after the moral graph of the database has been found, very little access to the database is needed.

The same idea is used when testing dependence between a join indicator and an attribute. Consider the join indicator J_{ij}. We can form the joint distribution of J_{ij}, a descriptive attribute A_i from relation R_i, and a descriptive attribute A_j from relation R_j issuing first a query that will extract the distribution $P(A_i, A_j, J_{ij} = \mathbf{T})$.

```
select A_i, A_j,count(*) from R_i, R_j
where J_{i,j} group by A_i, A_j.
```

Since $A_i \perp A_j$, we can extract the distribution $P(A_i, A_j, J_{ij} = \mathbf{F})$ by simply subtracting the values of $P(A_i, A_j, J_{ij} = \mathbf{T})$ from the values of $P(A_i)P(A_j)$. To compute, e.g., $P(A_i)$, we need to issue the query

```
select A_i,count(*) from R_i group by A_i.
```

Algorithm 1 only checks pairs of attributes for dependence, and issues only joins between base relations. Our model construction algorithm issues the same queries to the database as CORDS [8], a highly efficient previous approach to discovering correlations. By using samples of tables, the overhead of the CORDS construction algorithm was shown to be independent of the size of the database, while still resulting to high quality estimates.

We have so far guaranteed that all the probability distributions that need to be maintained have cardinality two. Within a relation, all cliques are of the form $(R.A_i, R.A_j)$. A clique containing a join indicator is in the worst case of the form $(R_i.A, R_j.B, J_{ij})$, which can be stored as two 2-dimensional distributions, one for the value $J_{ij} = \mathbf{T}$, and one for the value $J_{ij} = \mathbf{F}$. This guarantee is a central point of our work, and is indeed the primary reason why we can achieve low selectivity estimation times.

There is one exception to this rule: Algorithm 1 may create a moral graph that is not chordal, and triangulation may introduce cliques of higher order. This is easy to prevent by further restricting the model, but we have chosen not to, since the unrestricted model does not incur significant overhead in our experiments. The dotted lines in Figure 1(c) show the edges in the moral graph added by the triangulation process. Note that in this example, these extra edges did not create larger histograms. Appendix D contains more details.

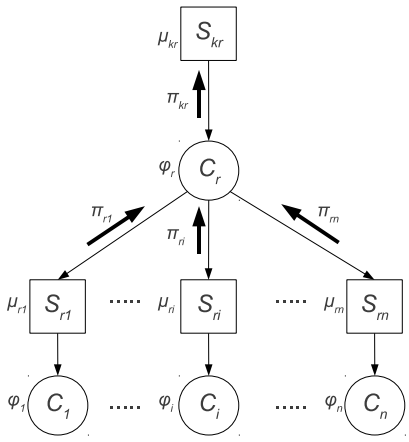


Figure 2: Selectivity estimation algorithm.

Model maintenance: When the underlying data changes, two kinds of updates need to be propagated to the model. First, updates that cause the probability distribution of an attribute to change. Second, updates that cause new correlations to be created or existing correlations to seize to be significant due to substantial changes in the data. For the first case, the distributions in which the attribute is contained are found and re-created. The second may need complete reconstruction of the model. However, strong correlations are usually a characteristic of the semantics of the database, and are not likely to change much over time. We plan to address incremental model updates in future work.

4. SELECTIVITY ESTIMATION

We move to the problem of using the statistical model for selectivity estimation. We assume that the statistical model is available in the form of a junction tree.

The task of selectivity estimation is to estimate the joint probability of a conjunction of selection and join predicates ψ. Assume that the predicates are over descriptive attributes $\mathcal{A}_q$ and join indicators $\mathcal{J}_q$. The task is then to compute $\Pr(\bigwedge_{A \in \mathcal{A}_q} \phi_A, \bigwedge_{J \in \mathcal{J}_q} J = \mathbf{T})$ where ϕ_A is a predicate over the attribute A.

We solve this problem using a junction tree representation of our statistical model in two steps. First, the so-called Steiner tree, the minimal sub-tree that contains $\mathcal{A}_q$ and $\mathcal{J}_q$, is extracted from the junction tree. Second, a propagation algorithm substitutes the values of the query variables, and eliminates the remaining variables in the tree. The final unnormalized number is returned as the result of the query.

For the sake of completeness, we give pseudo-code for the first step in Appendix C. For the second step, our propagation algorithm proceeds as follows. First, a clique of the reduced junction tree is selected as root. Messages pass from the leaves of the tree to the root, which will return the final result. Assume a clique C_r with parent clique C_k and child cliques $C_i, i = 1, \ldots, n$ as shown in Figure 2. Let $\mathcal{A}_{qr} \equiv \mathcal{C}_r \cap \mathcal{A}_q$, and $\mathcal{A}_{qri} \equiv \mathcal{S}_{ri} \cap \mathcal{A}_q$, i.e., the intersection of the query variables with a clique's variables and a separator's variables, respectively. Then, the following steps will be executed in C_r:

1. Messages $\pi_{ri}, i = 1, \ldots, n$ from the clique's are collected children by calling the algorithm recursively.
2. The relevant values from $\mathcal{A}_q$ are substituted to the clique potential ϕ_r and the potential of the upwards separator μ_{kr}:

$$\phi_r^*(C_r - \mathcal{A}_{qr} - \{J_r\}) = \phi_r[\bigwedge_{A \in \mathcal{A}_{qr}} A = a, J_r = \mathbf{T}]$$

$$\mu_{kr}^*(S_{kr} - \mathcal{A}_{qkr}) = \mu_{kr}[\bigwedge_{A \in \mathcal{A}_{qkr}} A = a]$$

   Note that we assumed that each clique contains at most one join indicator J_r and that separators contain only descriptive attributes. This always holds in our fixed model structure.
3. Let $S_{kr}^* = S_{kr} - \mathcal{A}_{qkr}$ be the set of attributes needed by the upwards clique. Then, the clique multiplies its new potential ϕ_r^* with the messages received from its children, summing out unnecessary variables.

$$\phi_r^* = \sum_{C_r \cup C_i \cup S_{ri} - \mathcal{A}_q - S_{kr}^*} \frac{\phi_r^*(C_r - \mathcal{A}_{qr})\pi_{ri}(C_i - \mathcal{A}_{qi})}{\mu_{ri}(S_{ri} - \mathcal{A}_{qri})}$$

4. The modified potential ϕ_r^* is the message π_{kr} passed to the upwards clique C_k.

The answer to the query is just the message π_{root} created by the root node. Pseudocode for the selectivity estimation algorithm is given in Algorithm 2.

We have further developed an improved dynamic programming algorithm that exploits the order of selectivity requests issued by a query optimizer, and computes estimates using less multiplications. Appendix E provides details and pseudo-code.

Algorithm 2 Basic propagation algorithm.

function COMPUTE-SELECTIVITY($\mathcal{T}$, $\mathcal{A}_q$, $\mathcal{J}_q$)
  return COMPUTE-PROB-REC($\mathcal{T}$.root, $\mathcal{A}_q$, $\mathcal{J}_q$)
function COMPUTE-PROB-REC(C_r, $\mathcal{A}_q$, $\mathcal{J}_q$)
  $\Pi = \emptyset$
  for $i = 0$ to n **do**
    π_{ri} =COMPUTE-PROB-REC(C_r.children[i], $\mathcal{A}_q$, $\mathcal{J}_q$)
    $\Pi = \Pi \cup \pi_{ri}$
  $\phi_r^* = \phi_r[J_R = \mathbf{T}, \bigwedge_{A \in \mathcal{A}_{qr}} A = a]$
  $\mu_{kr}^* = \mu_{kr}[\bigwedge_{A \in \mathcal{A}_{qkr}} A = a]$
  for $i = 1$ to n **do**
    $\phi_r^* = \frac{\phi_r^* \pi_{ri}}{\mu_{ri}^*}$
    $\phi_r^* = \sum_{C_r \cup C_i \cup \S_{ri} - \mathcal{A}_q - S_{kr}^*} \phi_r^*$
  return ϕ_r^*

5. EXPERIMENTAL STUDY

5.1 Implementation

We have implemented a graphical model foundation and the proposed selectivity estimation algorithms in PostgreSQL. To the best of our knowledge, this is the first implementation of graphical model based selectivity estimation in a real DBMS. Our implementation consists of two parts: the model construction prototype and the selectivity estimation prototype. Model construction (the implementation of Algorithm 1) is done outside the DBMS. It is written in Java, and accesses the database using SQL queries. The resulting junction tree structure is stored as four relational tables in the PostgreSQL catalog: (1) a relational table containing the descriptive attributes, (2) a table containing the join indicators, (3) a table containing the cliques and their inter-connections in the junction tree, and (4) a table containing the clique potentials.

The model construction algorithm is very efficient. It takes less than one minute to discover and construct the catalog tables for a scale-0.1 TPC-H data set, and approximately one hour for a full scale-1.0 TPC-H data set or the IMDB data set (see Appendix B for details on our data sets). These numbers are similar to those reported for tuple-graph synopses [17]. Note that we can reduce this time even further by using sampling [8]. In addition, the graphical model is space-efficient. It uses 13 pages in the PostgreSQL catalog, equivalent to roughly 100KB of storage space.

The selectivity estimation part is implemented in the PostgreSQL backend. When the optimizer is called, the clique catalog table is scanned, and the junction tree structure is created in the backend. The clique potentials are not read from disk at this point, since that would incur significant and unnecessary overhead. Then the Steiner tree for the query is created with an implementation of Algorithm 3 in Appendix C. Only then are the probability distributions in the much smaller, query-specific junction tree read from the catalog table. The startup overhead of loading the junction tree is very small: it takes between 1 and 3 milliseconds in all our experiments.

Selectivity estimation implements Algorithms 2 and 4 (in Appendix E) as operations on a junction tree structure. We implemented cliques and probability distributions as plan nodes in the PostgreSQL class system, including algorithms to multiply, divide, and marginalize multi-dimensional probability distributions. We used simple equi-width histograms for multi-dimensional probability distributions, where a multi-dimensional histogram is stored as an one-dimensional array. To ensure a fair comparison, we modified PostgreSQL to use equi-width histograms as well. We used arrays of size 10 for our distributions, and arrays of size 200 for the PostgreSQL histograms. A typical clique in our junction tree contains two descriptive attributes and one join indicator. Therefore, the size of the clique histogram is the same as a PostgreSQL histogram.

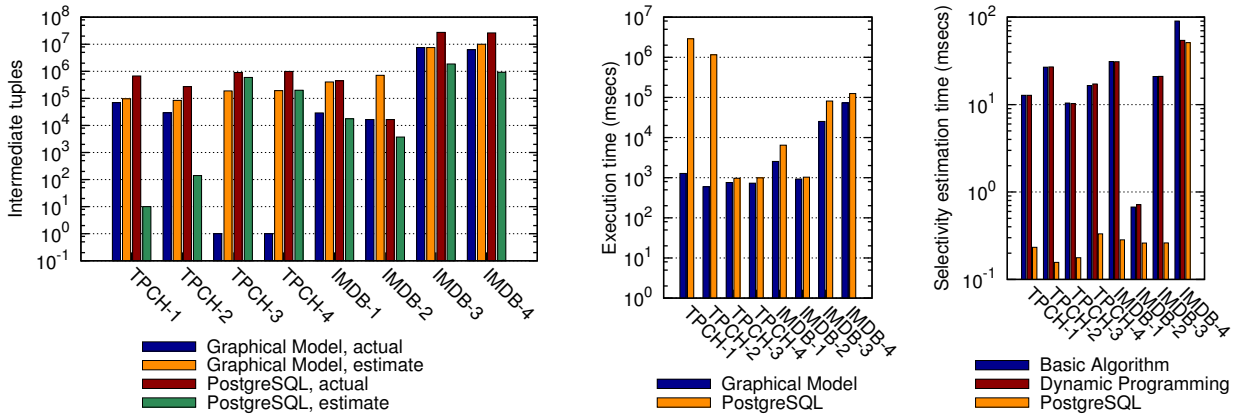
5.2 The impact of missed correlations

We proceed to offer insight into how missed correlation can affect the plan chosen by a query optimizer, and subsequently the time needed to execute a query. We use a skewed TPC-H data set with zipf factor $z = 3$. Correlations are inherent in the TPC-H schema. Consider the following query (denoted TPCH-1):

```
select c_name,c_address
from   lineitem,orders,customer
where  l_orderkey=o_orderkey and
       o_custkey=c_custkey and
       o_totalprice=x and
       l_extendedprice=y and
       c_acctbal=z
```

We seek to find customers that have placed orders with a particular combination of total price and prices of items, and with a further selection on the customer's account balance.

The attributes `l_extendedprice` and `o_totalprice` are correlated for tuples of `lineitem` and `orders` that join. Specifically, the total price of an order is a function of the price of its items, and their tax and discount. This correlation causes the selectivity $\Pr(J_{LO}, \phi_L, \phi_O)$ to be much higher than the product of the selectivities $\Pr(J_{LO})$, $\Pr(\phi_L)$, and $\Pr(\phi_O)$. PostgreSQL cannot capture such a correlation and therefore underestimates the selectivity $\Pr(J_{LO}, \phi_L, \phi_O)$ by a factor of 20 in our setting. This causes the optimizer to place the join $L \bowtie O$ before the join $O \bowtie C$ in the query plan. Further, it causes PostgreSQL to use a nested loop join



(a) Optimizer cost, as number of intermediate tuples generated. Estimated and actual costs are shown.

(b) Execution time.

(c) Time for selectivity estimation.

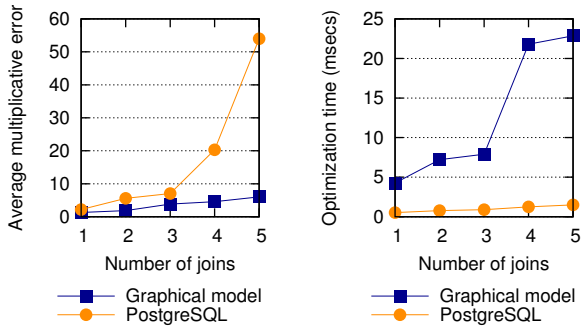
Figure 3: Examples of 2-join queries with correlations in the TPC-H and IMDB data sets. In all queries, PostgreSQL chose different plans using our versus the default estimates.

for the final join with C, using the join $L \bowtie O$ as the inner relation. The plan picked by the optimizer using the default PostgreSQL estimates is $A : (O \bowtie_{HJ} L) \bowtie_{NLJ} C$, where the right operand of the $\bowtie$ operator is pipelined. In contrast, using our improved estimates, the plan picked by the optimizer is $B : (C \bowtie_{HJ} O) \bowtie_{HJ} L$. Thus, both join order and the join algorithms picked were different. The execution time difference between these two plans is huge. While plan B (the one picked using our selectivity estimates) takes less than 2 seconds to execute in a cold state, plan A takes more than 40 minutes. After instructing PostgreSQL to not use a nested loop join, plan A is executed in 4 seconds. Therefore, the wrong join order can result in a $2\times$ execution time penalty, and the nested loop join increases the execution time by orders of magnitude. Both decisions were made due to the missed correlation between the attributes `l_extendedprice`, `o_totalprice`, and the join indicator J_{LO}. By capturing these correlations using a graphical model, the optimizer can pick a better query plan.

Figure 3 shows results for eight queries on the TPC-H and IMDB data sets. All axes are in logarithmic scale. Appendix B provides the SQL code of the queries, and details on the data sets. All queries are examples of missed correlations leading to a wrong join order, and indeed the PostgreSQL optimizer chose different plans using the default estimates versus using the estimates by our method. Our plans are better in all cases. Figure 3(a) shows the plan cost (measured as the number of intermediate tuples generated), both as estimated by the query optimizer, and its real value determined after query execution. Figure 3(b) shows the actual execution times for these queries.

PostgreSQL underestimates the cost of the query TPCH-1. As discussed, this causes the execution time to spiral (Figure 3(b)) due to a nested loop join. This is also the case for the query TPCH-2. For both these queries, our estimates are very close to the actual values. TPCH-3 and TPCH-4 are examples where our estimates resulted to overestimation. Here, a correlation causes a join to produce zero tuples. Using the default estimates, the optimizer misses the opportunity to place the join first. Although the difference is not as dramatic, the plan chosen using the default estimates was still worse by a factor of 1.5 for these two queries. In these queries, although the default estimates are more accurate for their resulting plan, an overestimation by our method can guide the optimizer to a completely different plan than the default estimates.

In the IMDB data set, correlations are present, but they are not as extreme as in the synthetic data set. Queries IMDB-1–IMDB-4 are all examples of underestimation by the PostgreSQL default selectivity estimates. The resulting differences in execution time can vary from very small (IMDB-2) to more than a factor of two (IMDB-1, IMDB-3, and IMDB-4).



(a) Average multiplicative error. (b) Average optimization time.

Figure 4: Results on a workload of 400 random queries over the TPC-H data set.

The time needed for selectivity estimation using our graphical model is significantly higher than that of PostgreSQL (Figure 3(c)). This is expected, given the complexity of performing propagation in a junction tree compared to simply checking a one-dimensional histogram. However, due to our optimizations, the selectivity estimation time is in the order of tens of milliseconds. The IMDB-4 query is a 3-join chain query. There, the dynamic programming algorithm (Appendix E) can reduce selectivity estimation time to about half, and it approaches the time needed by PostgreSQL. The rest of the queries are 2-join queries, and the time needed by the basic selectivity estimation algorithm and the dynamic programming algorithm are similar.

5.3 A random workload

Although what matters in practice is the execution time of the resulting plan (as shown in the previous section), using this as the sole metric for selectivity estimation can be misleading. Due to the complexity of query optimizers and the fine points of each individual optimizer, a better estimate does not always result in a better query plan. For example, a large over-estimation may be better than a slight under-estimation for certain queries, since it may result in a more "conservative" plan. In order to place equal emphasis on over- and under- estimation, we use the multiplica-

tive error metric, which has been shown to be the most appropriate for query optimization [6, 14]. Assume that a relation Q produced during query execution has cardinality $|r|$ and that the cardinality estimate is $|\hat{r}|$. Then, the multiplicative error is defined as $\max(|\hat{r}|, |r|)/\min(|\hat{r}|, |r|)$.

Given a query Q, we define the *average multiplicative error* as the geometric mean of the multiplicative errors for all estimates that are asked by the query optimizer during the optimization of the query: $\textit{avg-err}(Q) = \left(\prod_{i=1,\ldots,n} \frac{\max(|\hat{q}_i|, |q_i|)}{\min(|\hat{q}_i|, |q_i|)} \right)^{1/n}$, where q_i is a relation whose cardinality estimate was requested by the optimizer.

We generated a workload of 400 queries on a cyclic subset of the TPC-H data set (see Appendix B for details). Figure 4(a) shows the average multiplicative error of the default PostgreSQL and our selectivity estimates, averaged over queries with the same number of joins. Our estimates are better across all queries; and for 5-join queries, we can achieve a tenfold reduction of the multiplicative error. Figure 4(b) shows the optimization time. The basic selectivity estimation algorithm is used. The penalty for our better estimates is an increase in optimization time. However, optimization time is always in the order of tens of milliseconds, which is an acceptable overhead considering reduced estimation errors.

6. RELATED WORK

CORDS [8] provides a cost-effective schema-level synopsis that makes the uniform distribution assumption, but not the attribute and join independence assumptions. CORDS discovers correlated attributes, but does not create the corresponding probability distributions. Further, it is limited to pairs of attributes. Similarly to CORDS, we also consider only pairs of attributes in order to create the model efficiently. Contrary to CORDS, we avoid all three assumptions at once. Further, by organizing the dependencies in a junction tree, we can model indirect dependencies between attributes via chains or trees of dependencies. Our model construction algorithm has roughly the same complexity as CORDS.

Our work builds on Probabilistic Relational Models (PRMs) [7] that aim to model a database as a whole using a graphical model. Similarly to our work, PRMs can estimate the cardinality of a query without making any independence or uniformity assumptions. However, this approach cannot deal with cyclic schemas or non-key queries, both of which we can handle. Second, since Bayesian networks are used directly to model the database, large distributions may be kept, and most importantly the model construction algorithm is expensive. One can view our work as an efficient and practical adaptation of PRMs for use in real DBMSs.

Another approach is to maintain a "representative" sample of the database. Graph-based synopses [17] choose a subset of the complete tuple graph of a database and use that subset to perform selectivity estimation. The problem with that approach is that the independence assumptions made are encoded in the heuristic model construction algorithm. In contrast, our approach makes the (conditional) independence assumptions explicit by means of a graphical model. Although the approaches have a different outset (graphical models and XML graph summarization), it would be interesting to experimentally compare their efficiency in the future.

7. CONCLUSIONS AND FUTURE WORK

Missed correlations are frequently the reason behind estimation errors in modern optimizers. We present an approach to selectivity estimation that avoids the attribute value independence and the join uniformity assumptions. We carefully adapt techniques from graphical model theory. By restricting the space of possible models, we can capture the most important correlations while keeping the overhead of query optimization and model construction low. We present results based on an implementation inside a real DBMS. In several cases, we can achieve large execution time savings while maintaining optimization time in the range of tens of milliseconds.

We plan to explore several lines of research in the future. Within selectivity estimation, current multi-dimensional histograms do not fit well with our work. A novel structure is needed, that performs well in low dimensions with highly correlated attributes, and provides error guarantees after multiplications. Current work in this area has also highlighted such a need, in addition to minimization of the multiplicative error [14]. Second, we plan to address ways to incrementally update the statistical model when the underlying data changes. Another direction that we plan to pursue is advanced uses of correlations by a DBMS, such as correlation-aware physical design [12] and horizontally partitioned plans [18].

8. REFERENCES

[1] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *SIGMOD*, pp. 275–286, 1999.

[2] A. Berry, J. R. S. Blair, P. Heggernes, and B. W. Peyton. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica*, 39(4):287–298, 2004.

[3] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462467, 1968.

[4] R. G. Cowell, P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilisitc Networks and Expert Systems: Exact Computational Methods for Bayesian Networks*. Springer, 1999.

[5] A. Deshpande, M. N. Garofalakis, and M. I. Jordan. Efficient stepwise selection in decomposable models. In *UAI*, pp. 128–135, 2001.

[6] A. Deshpande, M. N. Garofalakis, and R. Rastogi. Independence is good: Dependency-based histogram synopses for high-dimensional data. In *SIGMOD*, pp. 199–210, 2001.

[7] L. Getoor, B. Taskar, and D. Koller. Selectivity estimation using probabilistic models. In *SIGMOD*, pp. 461–472, 2001.

[8] I. F. Ilyas, V. Markl, P. J. Haas, P. Brown, and A. Aboulnaga. CORDS: Automatic discovery of correlations and soft functional dependencies. In *SIGMOD*, pp. 647–658, 2004.

[9] Y. E. Ioannidis. The history of histograms (abridged). In *VLDB*, pp. 19–30, 2003.

[10] Y. E. Ioannidis and S. Christodoulakis. On the propagation of errors in the size of join results. In *SIGMOD*, pp. 268–277, 1991.

[11] F. V. Jensen and F. Jensen. Optimal junction trees. In *UAI*, pp. 360–366, 1994.

[12] H. Kimura, G. Huo, A. Rasin, S. Madden, and S. B. Zdonik. CORADD: Correlation aware database designer for materialized views and indexes. *PVLDB*, 3(1):1103–1113, 2010.

[13] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *SIGMOD*, pp. 448–459, 1998.

[14] G. Moerkotte, T. Neumann, and G. Steidl. Preventing bad plans by bounding the impact of cardinality estimation errors. *PVLDB*, 2(1):982–993, 2009.

[15] V. Poosala and Y. E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB*, pp. 486–495, 1997.

[16] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *SIGMOD*, pp. 23–34, 1979.

[17] J. Spiegel and N. Polyzotis. Graph-based synopses for relational selectivity estimation. In *SIGMOD*, pp. 205–216, 2006.

[18] K. Tzoumas, A. Deshpande, and C. S. Jensen. Sharing-aware horizontal partitioning for exploiting correlations during query processing. *PVLDB*, 3(1):542–553, 2010.

APPENDIX

A. EXAMPLE DATABASE

Figure 5 shows a database consisting of two relations: relation $R(X, Y, A)$ with five tuples, and relation $S(Z, W, B)$ with four tuples. The relations can join via the join predicate $R.A = S.B$. The descriptive attributes are $R.X$, $R.Y$, $S.Z$, and $S.W$.

The figure shows the universal relation, described in Section 3.1. The universal relation is the Cartesian product $R \times S$ projected on the descriptive attributes with the addition of the join indicator $J_{RS} \equiv (R.A = S.B)$. The universal relation captures the joint distribution $P(X, Y, Z, W, J_{RS})$, which is approximated using a graphical model.

The figure also shows one possible Bayesian network for the database, and the corresponding moral graph and junction tree. The junction tree is annotated by the probability distributions of its cliques, the only distributions that need to be kept and that factor the joint distribution as follows:

$$P(X, Y, Z, W, J_{RS}) = \frac{P(X, Y)P(Y, Z, J_{RS})P(Z, W)}{P(Y)P(Z)}.$$

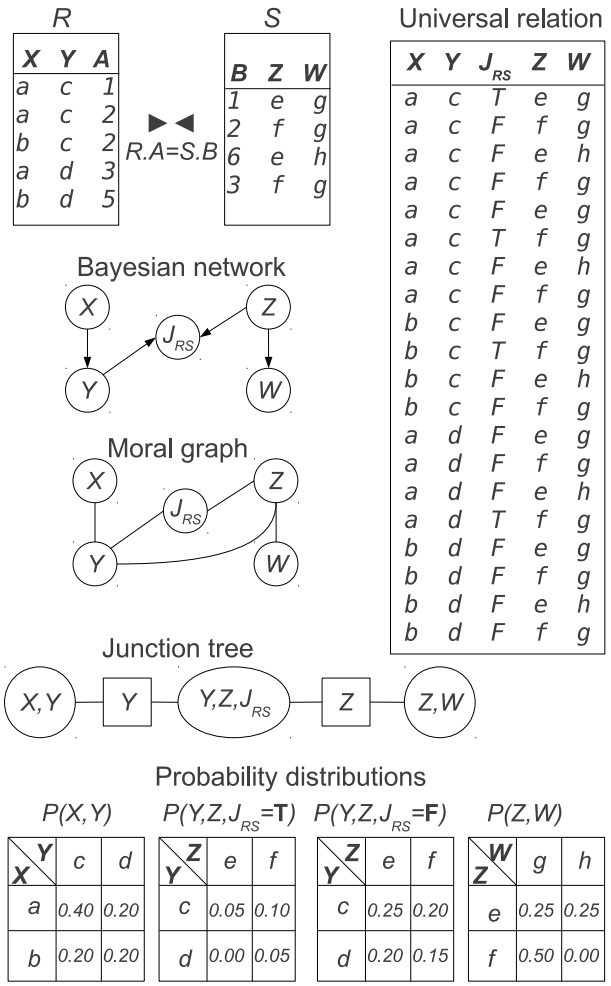


Figure 5: An example database of two relations. The universal relation, the Bayesian network, the moral graph, and the junction tree along with the probability distributions are shown.

B. DATA SETS AND QUERIES

Data sets. We use two data sets, the full IMDB data set and generated data for the TPC-H schema. The IMDB data set is approximately 800MB and consists of 21 tables, a total of 101 columns, and 54,400,459 rows. For the TPC-H data set, we tweaked the data generator to introduce additional pairwise correlations between attributes. We use a scale factor of $s = 0.1$, a zipf factor of $z = 3$, and a Pearson correlation parameter of $r = 0.9$ in our experiments.

Queries. In Section 5.2 we use a suite of queries that contain joins whose selectivity is correlated with specific values of descriptive attributes. The queries on the TPC-H schema are the following:

1. *TPCH-1*: The query finds information on customers with a given account balance that have placed orders with a given total price, as well as prices of the order lines. The correlation between `o_totalprice` and `l_extendedprice` causes the selectivity of the join $L \bowtie O$ to be very high.

```
select c_name,c_address
from   orders,lineitem,customer
where  o_orderkey=l_orderkey and
       o_totalprice=x and
       l_extendedprice=y and
       o_custkey=c_custkey and
       c_acctbal=z;
```

2. *TPCH-2*: The query finds information on customers with a given account balance, that have placed orders with a given ship date and order date. The strong correlation between `o_orderdate` and `l_shipdate` causes the selectivity of the join $L \bowtie O$ to be very high.

```
select c_name,c_address
from   lineitem,orders,customer
where  l_orderkey=o_orderkey and
       l_shipdate=x and
       o_orderdate=y and
       o_custkey=c_custkey and
       c_acctbal=z;
```

3. *TPCH-3*: The query finds information on suppliers with a given account balance that have participated in finished orders with a given ship date. The strong correlation between `o_orderstatus` and `l_shipdate` causes the selectivity of the join $L \bowtie O$ to be very low.

```
select s_name,s_address
from   orders,lineitem,supplier
where  o_orderkey=l_orderkey and
       o_orderstatus='F' and
       l_shipdate>x and
       l_suppkey=s_suppkey and
       s_acctbal=y;
```

4. *TPCH-4*: The query finds information on pricing and cost of orders that were shipped after a given date, and ordered before a given date, with a further selection on available quantity. The correlation between `o_orderdate` and `l_shipdate` causes the selectivity of the join $L \bowtie O$ to be very low.

```
select o_totalprice,ps_supplycost
from   lineitem,orders,partsupp
where  l_shipdate>x and
       o_orderdate<x and
       l_orderkey=o_orderkey and
       l_suppkey=ps_suppkey and
       ps_availqty=y;
```

The queries on the IMDB schema were the following:

1. *IMDB-1*: The query finds titles produced in 2009, for which the distributing company was US-based.

```
select T.title
from   title as T,
       movie_companies as MC,
       company_name as CN
where  MC.company_id=CN.id and
       MC.company_type_id=1 and
       CN.country_code='[us]' and
       T.id=MC.movie_id and
       T.production_year=2009;
```

2. *IMDB-2*: The query finds movie titles produced in 2009 for which the distributing company is known.

```
select T.title
from   title as T,
       movie_companies as MC,
       company_name as CN
where  MC.company_id=CN.id and
       MC.company_type_id=1 and
       T.kind_id=1 and
       T.id=MC.movie_id and
       T.production_year=2009;
```

3. *IMDB-3*: The query retrieves titles together with trivia on actors that have played in the movie.

```
select T.title,CI.info
from   cast_info as CI,
       person_info as PI,
       title as T
where  CI.person_id=PI.person_id and
       CI.role_id=1 and
       PI.info_type_id=17 and
       CI.movie_id=T.id;
```

4. *IMDB-4:* The query retrieves titles together with character names and trivia on actors that have played in the movie.

```
select T.title,CN.name,CI.info
from   cast_info as CI,
       person_info as PI,
       title as T,
       char_name as CN
where  CI.person_id=PI.person_id and
       CI.role_id=1 and
       PI.info_type_id=17 and
       CI.movie_id=T.id and
       CI.person_role_id=CN.id;
```

In Section 5.3 we use a random workload of selection and join queries over the restricted TPC-H schema graph shown in Figure 1(a). We use 21 descriptive attributes and introduced two predicates per attribute. Figure 1(d) also shows the junction tree created for a subset of those attributes. By taking all possible combinations, we generate 80 one-join queries. By randomly combining queries with one join, we generate 80 two-join queries, until 400 queries with one to five join predicates are created.

C. STEINER TREE COMPUTATION

Given a rooted junction tree $\mathcal{T}$ and a set of variables $\mathcal{V}$, our goal is to extract the minimal connected junction tree that contains these variables. For each clique of the tree C, we maintain a Boolean variable "C.include" that indicates whether the clique is included in the resulting tree, and we maintain a set "C.varsInSubtree" that contains the subset of $\mathcal{V}$ that is included in the subtree rooted at C. The algorithm, shown in Algorithm 3, sets the "include" value for each clique. If a clique is a leaf, it is included if it contains variables in the query. A non-leaf clique is included if the variables contained in the sub-tree originating from the clique contain query variables. The fine point of the algorithm is that we do not want to include a clique whose parent already covers the variables of interest. Lines 15 and 16 of the algorithm set the include bit of a clique's child to false if the parent clique already covers the needed variables.

Algorithm 3 Steiner tree computation

```
1:  procedure CONSTRUCT-STEINER-TREE(T,V)
2:      C_root = root of T
3:      CLIQUE-INCLUDE(C_root,V)
4:      return cliques with C.include = T
5:  procedure CLIQUE-INCLUDE(C,V)
6:      if C is leaf then
7:          C.varsInSubtree = V ∩ C.vars
8:          if C.varsInSubtree ≠ ∅ then
9:              C.include = T
10:     else
11:         localVarsInSubtree = V ∩ C.vars
12:         C.varsInSubtree = localVarsInSubtree
13:         for c in C.children do
14:             CLIQUE-INCLUDE(c,V)
15:             if localVarsInSubtree ⊃ c.varsInSubtree then
16:                 c.include = F
17:             C.varsInSubtree = C.varsInSubtree ∪ c.varsInSubtree
18:         if C.varsInSubtree ≠ ∅ then
19:             C.include = T
20:         if C.varsInSubtree = V then
21:             Stop the algorithm
```

D. TRIANGULATION AND ITS IMPLICATIONS

A junction tree can be constructed only if the moral graph $\mathcal{G}^m$ is *chordal*. A graph is chordal if it does not contain a cycle with more than three nodes without a "chord," an edge that connects two non-adjacent nodes in the cycle.

For the case that the moral graph is not chordal, an extra step before creating the junction tree is needed. Triangulation is a process that adds edges to the moral graph until a chordal graph has been constructed. It is usually realized as a process of node elimination [2]. A node is eliminated by connecting all its neighbors, and removing the node and its edges from the graph. A node is called simplicial if it can be eliminated without introducing extra edges (i.e., all its neighbors are connected).

Algorithm 1 may create a moral graph that is not chordal. This can happen if the query graph of the database contains cycles, but it cannot happen in a tree-shaped query graph. In our fixed model structure, join indicators are simplicial nodes because they have at most two parents that are connected during moralization (e.g., in the moral graph shown in Figure 1, we can see that the join indicator nodes $J_{LP}, J_{LS}, \ldots$ are all simplicial nodes). Thus, triangulation will not add edges involving join indicators. It will only add edges that contain descriptive attributes (e.g., edge between l_sdate and o_odate in Figure 1). This can only happen in the case of a cyclic query graph, where different descriptive attributes are parents to join indicators of the same relation. Disallowing the latter is an easy solution to guarantee that all histograms are 2-dimensional. Alternatively, one can choose to allow the possibility of higher-dimensionality histograms.

The latter is acceptable in most cases because, even with cliques containing 3 or more descriptive attributes, we often only need to store at most 2-dimensional histograms. For example, consider

the clique $\{s\_acctbal, c\_acctbal, l\_sdate\}$ created during triangulation in Figure 1. The three variables in this clique are independent of each other since they belong to different relations. Hence, this 3-dimensional probability distribution can be stored compactly as a collection of three 1-dimensional distributions. Although this seems counter-intuitive, it is actually expected because the junction tree encodes fewer conditional independencies than the original Bayesian network (because it contains more edges). On the other hand, the clique $\{l\_sdate, l\_cdate, o\_odate\}$ requires us to store a 2-dimensional histogram on $\{l\_sdate, l\_cdate\}$. In future work, we would like to investigate whether it is possible to construct a custom elimination sequence that always produces a chordal graph that needs only 2-dimensional distributions.

E. A DYNAMIC PROGRAMMING ALGORITHM

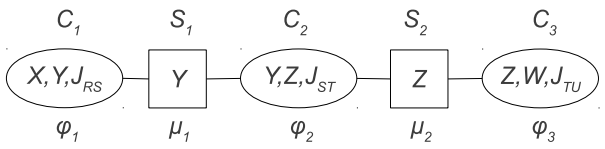


Figure 6: Junction tree for the query $R \bowtie S \bowtie T \bowtie U$.

The basic selectivity estimation algorithm presented in Section 4 can be improved if we consider the sequence of estimates requested by the query optimizer. Assume the chain join query $R \bowtie S \bowtie T \bowtie U$ and the junction tree for this query depicted in Figure 6. During plan enumeration, estimates are requested from the selectivity estimation component in a certain order. When a joint probability, e.g., $\Pr(J_{RS} = \mathbf{T}, J_{ST} = \mathbf{T})$, is to be estimated, the probabilities, e.g., $\Pr(J_{RS} = \mathbf{T})$ and $\Pr(J_{St} = \mathbf{T})$, have already been computed. Instead of executing the basic propagation algorithm for $\Pr(J_{RS} = \mathbf{T}, J_{ST} = \mathbf{T})$, we can re-use the previous findings in a dynamic programming manner to avoid redundant computation.

We maintain an array β. An entry $\beta[\mathcal{J}]$ caches the intermediate distribution that is used to compute $\Pr(\bigwedge_{J \in \mathcal{J}} J = \mathbf{T})$ and higher-order probabilities. When the query optimizer needs a cardinality estimate, it calls the function COMPUTE-SELECTIVITY, shown in Algorithm 4. It provides two arguments: a set of join indicators $\mathcal{J}$ that correspond to the joins in the intermediate relation, and possibly a set of predicates on descriptive attributes $\mathcal{A}$. There are two cases:

1. The probability contains only one join indicator J_i that belongs to the clique C_i of the junction tree. Then, a new β entry is created:
$$\beta[J_i](\bigcup_j S_{ij}) = \sum_{C_i - \bigcup_j S_{ij}} \phi_i[J_i = \mathbf{T}, \bigwedge_{A \in \mathcal{A}_q \cap C_i} A = a].$$
The returned result is simply $\sum \beta[J_i]$.
2. The set $\mathcal{J}$ contains more than one join indicators. Then, there should exist sets $\mathcal{J}_1$ and $\mathcal{J}_2$ such that $\mathcal{J}_1 \cup \mathcal{J}_2 = \mathcal{J}$, $\mathcal{J}_1 \cap \mathcal{J}_2 = \emptyset$, and the entries $\beta[\mathcal{J}_1]$ and $\beta[\mathcal{J}_2]$ are not null. These two sets correspond to two sub-trees of the junction tree, separated by the separator S_{12}. Then, the entry $\beta[\mathcal{J}]$ is created: $\beta[\mathcal{J}] = \sum_{S_{12}} \beta[\mathcal{J}_1]\beta[\mathcal{J}_2]/\mu_{12}$. The returned result is simply $\sum \beta[\mathcal{J}]$.

According to this (simplified) algorithm, only one clique multiplication per estimate is needed. Unfortunately, that is not true in the general case. The two entries $\beta[\mathcal{J}_1]$ and $\beta[\mathcal{J}_2]$ may not correspond to neighboring cliques. In that case, they must be multiplied with all the cliques that lie between them. However, the number of clique multiplications is always lower than in the basic propagation algorithm. A second caveat exists for junction trees with high fanout. There, the number of variables of an entry in a β table may grow to be so large that it dominates the optimization time. Thus, the dynamic programming algorithm is suitable for junction trees with low fanout.

Algorithm 4 Dynamic programming algorithm.

function COMPUTE-SELECTIVITY($\mathcal{J} \subset \mathcal{J}_q, \mathcal{A} \subset \mathcal{A}_q$)
  if $|\mathcal{J}| = 1$ **then**
    $\beta[J_i] = \sum_{C_i - \bigcup_j S_{ij}} \phi_i[J_i = \mathbf{T}, \bigwedge_{A \in \mathcal{A}_q \cap C_i} A = a]$
    return $\sum \beta[J_i]$
  else
    Let $\mathcal{J}_1, \mathcal{J}_2$ such that $\mathcal{J}_1 \cup \mathcal{J}_2 = \mathcal{J} \wedge \mathcal{J}_1 \cap \mathcal{J}_2 = \emptyset \wedge \beta[\mathcal{J}_1] \neq$ null $\wedge\ \beta[\mathcal{J}_2] \neq$ null
    Let S_{12} be the separator that connects the sub-trees of the junction tree that contain $\mathcal{J}_1$ and $\mathcal{J}_2$.
    $\beta[\mathcal{J}] = \sum_{S_{12}} \frac{\beta[\mathcal{J}_1]\beta[\mathcal{J}_2]}{\mu_{12}}$
    return $\sum \beta[\mathcal{J}]$

F. PROOFS

F.1 Proof of Theorem 1

Theorem 3. *Assume the relations $R \neq S \neq T \neq U \in \mathcal{S}$. The following independencies hold in $P_{\mathcal{U}}$:*

1. *Descriptive attributes belonging to different relations are independent: $P_{\mathcal{U}}(R.X, S.Y) = P_{\mathcal{U}}(R.X)P_{\mathcal{U}}(S.Y)$.*
2. *Join indicators are independent if they do not share a relation: $P_{\mathcal{U}}(J_{RS}, J_{TU}) = P_{\mathcal{U}}(J_{RS})P_{\mathcal{U}}(J_{TU})$.*
3. *A descriptive attribute and a join indicator are independent if they do not share a relation: $P_{\mathcal{U}}(J_{RS}, T.X) = P_{\mathcal{U}}(J_{RS})P_{\mathcal{U}}(T.X)$.*

PROOF. The proof uses simply the frequentist definition of probability:

$$\Pr(R.X = x) \equiv \frac{|\sigma_{R.X=x}(R)|}{|R|}$$
$$\Pr(J_{RS} = \mathbf{T}) \equiv \frac{|R \bowtie S|}{|R \times S|}$$
$$\Pr(J_{RS} = \mathbf{F}) \equiv \frac{|R \not\bowtie S|}{|R \times S|};$$

and it uses the fact that selection commutes over the Cartesian product:

$$\sigma_{R.X=x}(R \times S) = \sigma_{R.X=x}(R) \times S.$$

1. Assume arbitrary values $x \in \text{Dom}(R.X)$ and $y \in \text{Dom}(S.Y)$. Then,
$$\begin{aligned}\Pr(R.X = x, S.Y = y) &\equiv \frac{|\sigma_{R.X=x \wedge S.Y=y}(R \times S)|}{|R \times S|} \\ &= \frac{|\sigma_{R.X=x}(R)||\sigma_{S.Y=y}(S)|}{|R||S|} \\ &= \frac{|\sigma_{R.X=x}(R)|}{|R|}\frac{|\sigma_{S.Y=y}(S)|}{|S|} \\ &\equiv \Pr(R.X = x)\Pr(S.Y = y).\end{aligned}$$

2. Assume $J_{RS} \equiv (R.a = S.a)$ and $J_{TU} \equiv (T.c = U.c)$. We need to prove four cases ($\{\mathbf{T},\mathbf{T}\}$, $\{\mathbf{T},\mathbf{F}\}$, $\{\mathbf{F},\mathbf{T}\}$, $\{\mathbf{F},\mathbf{F}\}$) for the possible values of the two join indicators. For example, for $J_{RS} = \mathbf{T}$ and $J_{TU} = \mathbf{F}$ we have:

$$\begin{aligned}
&\Pr(J_{RS} = \mathbf{T}, J_{TU} = \mathbf{F}) \\
&\quad\equiv \frac{|\sigma_{R.a=S.a \wedge T.c \neq U.c}(R \times S \times T \times U)|}{|R \times S \times T \times U|} \\
&\quad= \frac{|\sigma_{R.a=S.a}(R \times S)||\sigma_{T.c \neq U.c}(T \times U)|}{|R \times S||T \times U|} \\
&\quad= \frac{|R \bowtie S||T \not\bowtie U|}{|R \times S||T \times U|} \\
&\quad\equiv \Pr(J_{RS} = \mathbf{T}) \Pr(J_{TU} = \mathbf{F}).
\end{aligned}$$

The remaining cases are proven similarly.

3. Proven similarly. □

F.2 Proof of Theorem 2

Theorem 4. *The fixed model structure respects the relational independencies and produces a valid (acyclic) Bayesian network.*

PROOF. It is easy to prove that any Bayesian network that conforms to our fixed model structure is acyclic. First, since the "local" Bayesian network in each relation is a directed tree, it cannot contain a cycle. Second, since a join indicator has no children, it cannot be a part of a directed cycle.

For the first part of the theorem, we need to prove that any BN that conforms to our fixed model structure encodes the relational independencies. Recall that the conditional independencies encoded by a Bayesian network are

$$X \perp \text{NonDesc}(X) | \text{Pa}(X) \forall X.$$

We need to prove three cases:

1. Two descriptive attributes from different relations are independent. Consider the attribute X from relation R, and the attribute Y from relation S. The local Bayesian network of the relation R is a directed tree. Consider the path from the root of that tree, X_1, to X: $X_1 \rightarrow X_2 \rightarrow \cdots \rightarrow X_n \rightarrow X$. Due to the fixed model structure, Y is a non-descendant of all attributes $X_1, \ldots, X_n, X$. Since X_1 has no parent, we obtain from the Bayesian network that

$$X_1 \perp Y | \emptyset \Rightarrow P(Y|X_1) = P(Y).$$

Using the above result, and that X_2 is independent of Y given X_1 we obtain that

$$\begin{aligned}
&X_2 \perp Y | X_1 \Rightarrow \\
&P(X_2, Y|X_1) = P(X_2|X_1)P(Y|X_1) \\
&P(X_2, Y|X_1) = P(X_2|X_1)P(Y).
\end{aligned}$$

For the joint distribution of X_2 and Y we have:

$$\begin{aligned}
P(X_2, Y) &= \sum_{X_1} P(X_1, X_2, Y) \\
&= \sum_{X_1} P(X_2, Y|X_1)P(X_1)) \\
&= \sum_{X_1} P(X_2|X_1)P(Y)P(X_1)) \\
&= \sum_{X_1} P(X_2, X_1)P(Y) \\
&= P(X_2)P(Y).
\end{aligned}$$

Therefore, $X_2 \perp Y$. In the same way, by following the chain from X_2, we can deduce $X \perp Y$.

2. A descriptive attribute is independent from a join indicator that does not involve the attribute's relation. Consider the attribute X of relation R and the join indicator J_{ST}. J_{ST} has at most two parents, one descriptive attribute from relation S, and one descriptive attribute from relation T. Therefore, J_{ST} is a non-descendant of X. We can prove that $X \perp J_{ST}$ in the same way as in item 1 by substituting Y with J_{RS}.
3. Two join indicators that do not involve a common relation are independent. Consider the join indicators J_{RS} and J_{TU}. Due to the fixed model structure, J_{TU} is a non-descendant of J_{RS}. If J_{RS} does not have any parents,

$$J_{RS} \perp J_{TU} | \emptyset \Rightarrow J_{RS} \perp J_{TU}.$$

If J_{RS} has one parent, assume from relation R, X:

$$J_{RS} \perp J_{TU} | X.$$

From item 2 we have $X \perp J_{TU}$.

$$\begin{aligned}
P(J_{RS}, J_{TU}) &= \sum_X P(J_{RS}, J_{TU}, X) \\
&= \sum_X P(J_{RS}, J_{TU}|X)P(X) \\
&= \sum_X P(J_{RS}|X)P(J_{TU}|X)P(X) \\
&= \sum_X P(J_{RS}|X)P(J_{TU})P(X) \\
&= \sum_X P(J_{RS}, X)P(J_{TU}) \\
&= P(J_{RS})P(J_{TU})
\end{aligned}$$

If J_{RS} has two parents,$R.X$ and $S.Y$:

$$J_{RS} \perp J_{TU} | X, Y.$$

From item 2 we have $X \perp J_{TU}$ and $Y \perp J_{TU}$.

$$\begin{aligned}
P(J_{RS}, J_{TU}) &= \sum_{X,Y} P(J_{RS}, J_{TU}, X, Y) \\
&= \sum_{X,Y} P(J_{RS}, J_{TU}|X, Y)P(X, Y) \\
&= \sum_{X,Y} P(J_{RS}, J_{TU}|X, Y)P(X)P(Y) \\
&= \sum_{X,Y} P(J_{RS}|X, Y)P(J_{TU}|X, Y)P(X)P(Y) \\
&= \sum_{X,Y} P(J_{RS}|X, Y)P(J_{TU})P(X)P(Y) \\
&= \sum_{X,Y} P(J_{RS}, X, Y)P(J_{TU}) \\
&= P(J_{RS})P(J_{TU})
\end{aligned}$$

□

Acknowledgements

This research was funded in part by grant 645-06-0517 from the Danish Agency for Science, Technology and Innovation. C. S. Jensen is an Adjunct Professor at University of Agder, Norway.