# Selectivity Estimation for Hybrid Queries over Text-Rich Data Graphs

Andreas Wagner
AIFB, KIT
Karlsruhe, Germany
a.wagner@kit.edu

Veli Bicer
IBM Research, Smarter Cities
Technology Centre
Dublin, Ireland
velibice@ie.ibm.com

Thanh D. Tran
AIFB, KIT
Karlsruhe, Germany
ducthanh.tran@kit.edu

## ABSTRACT

Many databases today are *text-rich*, comprising not only *structured*, but also *textual* data. Querying such databases involves predicates matching structured data combined with string predicates featuring textual constraints. Based on selectivity estimates for these predicates, query processing as well as other tasks that can be solved through such queries can be optimized. Existing work on selectivity estimation focuses either on *string* or on *structured query predicates* alone. Further, probabilistic models proposed to incorporate dependencies between predicates are focused on the relational setting. In this work, we propose a *template-based probabilistic model*, which enables selectivity estimation for general *graph-structured data*. Our probabilistic model allows dependencies between structured data and its text-rich parts to be captured. With this general probabilistic solution, $BN^+$, selectivity estimations can be obtained for queries over text-rich graph-structured data, which may contain structured and string predicates (*hybrid queries*). In our experiments on real-world data, we show that capturing dependencies between structured and textual data in this way greatly improves the accuracy of selectivity estimates without compromising the efficiency.

## 1. INTRODUCTION

Databases today are increasingly *text-rich* comprising *structured* and *textual* data. Examples include databases storing documents enriched with structured data, e.g., in the form of RDFa or Microformats[1], or structured data (like RDF) containing textual attributes. In recent years, the amount of RDF data on the Web drastically increased, e.g., published as Linked Data[2]. RDF data comprises entity descriptions, where each is a set of *triples* $\{\langle s, p, o\rangle\}$. Every triple describes a particular entity $s$ (*subject*) through a *predicate object* pair $p, o$. RDF entity descriptions oftentimes contain text-rich predicates such as `comment` or `description`. RDF, like many other kinds of data, can generally be conceived as a data graph, as shown in Fig. 1. In the case of RDF, such a graph is formed by triples.

Standard languages for querying graph-structured data include SQL and SPARQL. At the core, these languages support *conjunctive queries*. With respect to data graphs, conjunctive queries can be seen as being composed of *query predicates* $\langle s, p, o\rangle$. Here, each $s$, $p$ and $o$ might be a *variable* or correspond to an entity, a predicate or an object in the data (called *constant*).

Processing conjunctive queries typically involves a query optimizer, which relies on *selectivity estimates* for query predicates to construct an optimal query plan. An optimal query plan aims at minimizing the amount of intermediate results. In fact, selectivity estimates, as studied in this paper, are not only crucial for standard query processing, but all problems that be solved via conjunctive queries. For instance, data extraction [24] and data integration programs [3] have been formulated as queries, which involve selection and (similarity) join predicates.

Targeting a low computational overhead, estimation is based on *data synopses*, which approximately capture underlying data value distributions through a statistical summary. Several assumptions are commonly employed to keep such a synopsis small respectively simple.

(1) The *uniform distribution assumption* implies that all values for a predicate are equally likely. For instance, for predicate `name` (Fig. 1) featuring four distinct values, denoted as $|\Omega(X_{name})| = 4$ (with $\Omega$ as *sample space*), the probability for an entity $x$ having `name` "William Wyler" is $P(X_{name} =$ "William Wyler"$) \approx 1/|\Omega(X_{name})| = 1/4$. In other words, the probability for a query predicate $\langle x, name,$"William Wyler"$\rangle$ is $1/4$. Clearly, this assumption may lead to misestimates, when "William Wyler" is a common name shared by several entities.

(2) Given a second query predicate $\langle x, comment,$"Audrey Hepburn was..."$\rangle$, with $P(X_{comment} =$ "Audrey Hepburn was..."$) = 1$, the *predicate value independence assumption* dictates that our two predicate values are independent. That is, the probability of observing both events is $P(X_{name} =$ "William Wyler", $X_{comment} =$ "Audrey Hepburn was..."$) \approx P(X_{name} =$ "William Wyler"$) \cdot P(X_{comment} =$ "Audrey Hepburn was..."$) = 1/4 \cdot 1$. However, as we can observe in the data, there is actually no entity that is associated with that `name` and `comment`. Thus, the joint probability should actually be 0. Such a misestimate is due to correlations among data values. Given the value for `name`, a particular value for `comment` is more or less likely to occur (instead of being equally likely).

[1] `http://webdatacommons.org/`
[2] `http://www.w3.org/DesignIssues/LinkedData.html`

(3) Finally, there is the *join predicate independence*, which is a special case of the previous assumption. It states that the existence of a predicate is independent of the value respectively existence of another predicate. Reconsidering our example, the existence of `comment` and any value for `name` would be assumed independent. Again, such simplification would lead to errors, as `comment` only occurs with `name` "Audrey Hepburn".

A large body of work has been devoted to avoid one or more independence assumptions. Approaches wish to consider data correlations, thereby allowing for more accurate estimates. Assumption (1) is addressed by counting values and embedding the resulting frequency statistics into synopses such as histograms [19] and wavelets [17]. Dealing with assumptions (2) respectively (3) requires a joint distribution of two or more random variables, which may be approximated via join synopses [2], tuple-graph synopses [21] or Probabilistic Relational Models [10, 23] (PRMs). A joint distribution comprising all possible dependencies between random variables is, however, high-dimensional. Thus, a data synopsis may suffer from an exponential blowup of storage space and computational cost. On the other hand, oftentimes two events are actually independent, if conditioned on a particular third event. PRM approaches [10, 23], like any instantiation of graphical models, exploit this *conditional independence* between random variables to factor a full joint distribution into multiple low-dimensional distributions. This factorization allows high-dimensional distributions to be captured more compactly.

**Graph-Structured Data and Queries.** Existing PRM-based solutions [10, 23] are proposed for relational data. In particular, they assume a partitioning scheme that determines the tables in which data is stored. Further, such approaches take queries as inputs, which explicitly specify the tables from which data shall be retrieved. For instance, consider the query predicate $\langle x, name, \text{"William Wyler"}\rangle$, which selects all bindings from the entire data graph matching that `name`. In contrast to that, previous works assume a selection predicate to have a `FROM` clause that specifies the table from which data is selected, e.g., `Person`. Thus, the probability $P(X_{name} = \text{"William Wyler"})$ is estimated for bindings in the `Person` table only. Applying such solutions to a graph-structured setting is not directly possible, because queries used here do not contain table information. Further, data graphs can be partitioned in various ways. Different partitioning schemes, however, yield different tables, which in turn greatly affect the performance of existing solutions.

**Queries over Text-Rich Data.** Another problem with PRM-based solutions [10, 23] is that random variables are assumed to have small sample spaces. In existing works, random variables capture structured query predicates with constants that are bounded to a fixed number of values. In addition to structured predicates, we aim to support string predicates for specifying keyword constraints over textual values. In particular, string predicates comprise keywords, which match any value that contains such keywords. That is, results for these string predicates do not have to exactly match a specified constant, but only have to *contain* a given keyword. For instance, bindings for $\langle x, name, \text{"William Wyler"}\rangle$ would also satisfy the predicate $\langle x, name, \text{"William"}\rangle$, as they both contain "William". To query via string predicates on predicate `name`, a sample space $\Omega(X_{name})$ must comprise all words as well as phrases (sequences of words) contained in text values for `name`. Clearly, $\Omega(X_{name})$ may potentially be very large. This problem is exacerbated, when dependencies between values in these sample spaces have to be considered. For dealing with string predicates, specific *string synopses* summarizing the value spaces of textual attributes have been proposed. For instance, synopses based on pruned suffix trees, Markov tables, clusters or $n$-grams have received much attention [4, 13, 24]. However, previous works estimate the selectivity of *single* string predicates. In our setting, we aim to support queries that comprise a combination of structured and string query predicates: *hybrid queries*. In particular, there is no work, which considers dependencies between these different types of predicates.

**Contributions.** Towards a holistic solution for selectivity estimation of queries over text-rich data graphs, we provide the following contributions: (1) for our work we rely on an instantiation of a general template-based representation of Bayesian networks (BN). Our probabilistic model is able to capture value distributions and dependencies between them for general data graphs. As opposed to assuming specific tables [10, 23] in which data stored, our model is learned – independent from a partitioning scheme – directly from information captured in the data graph. (2) For supporting hybrid queries over text-rich data graphs, we show how existing string synopses can be integrated into our template-based model, called BN$^+$. (3) We implemented this approach to perform experiments on real-world data. Using a baseline relying on independence between structured query predicates and string predicates, we show that our solution greatly improves the accuracy of selectivity estimates. Also in terms of efficiency our solution is promising, as BN$^+$ performs comparable to our baseline systems. In fact, our results suggest that BN inferencing requires only negligible computational overhead.

**Outline.** The remainder is structured as follows: first, in sect. 2 we present preliminaries. Sect. 3 discusses BN$^+$, our template-based model with integrated string synopsis. In sect. 4 we present evaluation results, before we outline related work in sect. 5. Finally, we conclude with sect. 6.

## 2. PRELIMINARIES

In this work, we use graphs as data and query model.

**Data.** Given a set of attribute labels $\ell_a$ and relation labels $\ell_r$, we model our data as a directed labeled graph $\mathcal{G} = (V, E, \ell_a, \ell_r)$, where $V$ is the disjoint union $V = V_E \uplus V_A \uplus V_C$ of *entity* nodes $V_E$, *attribute value* nodes $V_A$, and *class* nodes $V_C$. The edges (called triples) $E = E_R \uplus E_A \uplus \{type\}$ represent a disjoint union of *relation* and *attribute* edges. Relation edges $E_R$ connect entity nodes, i.e., $\langle s, r, o\rangle \in E_R$ iff $s, o \in V_E$ and $r \in \ell_r$. Attribute edges $E_A$ connect an entity with an attribute value, $\langle s, a, o\rangle \in E_A$ iff $s \in V_E, o \in V_A$ and $a \in \ell_a$. Attribute values may stand for long textual descriptions. Each such value is seen as a bag of $n$-grams. Here, we define an $n$-gram to be a sequence of $n$ words. Thus, in order to form an $n$-gram representation for a given attribute value, all possible words and phrases (up to length $n$) are extracted from that value. Further, a special edge $\langle s, type, o\rangle \in E$, $s \in V_E$ and $o \in V_C$, captures an entity $s$ belonging to a class $o$. Different kinds of data, including relational, XML and RDF data have been modeled as graphs. For instance, the intuitive mapping of relational data to this graph model is: a database tuple captures an entity, its attributes, and connections to related entities in the form of foreign keys. Further, our model closely resembles the graph-structured RDF data model, omitting special features such as RDF blank nodes. Example data is given in Fig. 1.

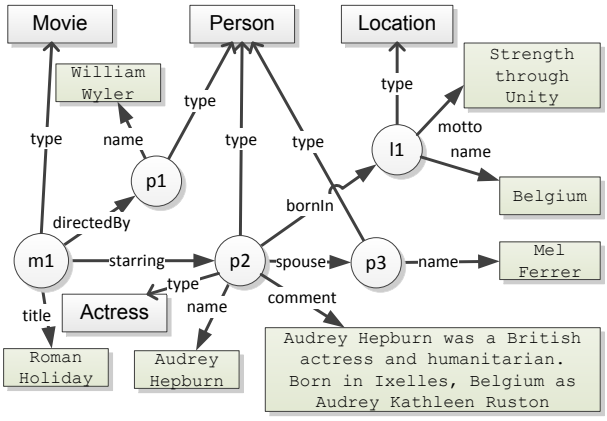



**Figure 1: A data graph about Audrey Hepburn and her movie "Roman Holiday".**

**Query.** Conjunctive queries represent a large part of SQL as well as the *Basic Graph Pattern* (BGP) feature of SPARQL[3]. A conjunctive query is a conjunction of query predicates. We use a particular type of conjunctive queries that corresponds to graph patterns of the following form: a query $Q$ over a data graph $G$ is a directed labeled graph $G_Q = (V_Q, E_Q)$, where $V_Q$ is a disjoint union $V = V_{Q_V} \uplus V_{Q_C} \uplus V_{Q_K}$ of *variable* nodes ($V_{Q_V}$), *constant* nodes ($V_{Q_C}$), and *keyword* nodes ($V_{Q_K}$). Each $o \in V_{Q_K}$ is a word or a sequence of words. Predicates are explicitly given in a query, i.e., $p$ is a constant for all query predicates $\langle s, p, o\rangle \in E_Q$. Corresponding to edge types, $\ell_a$, $\ell_r$ and *type* in $G$, we distinguish three kinds of query predicates: (1) class predicates $\langle s, type, o\rangle, s \in V_{Q_V}, o \in V_{Q_C}$, (2) relation predicates $\langle s, r, o\rangle, s \in V_{Q_V}, o \in V_{Q_C} \uplus V_{Q_V}, r \in \ell_r$ and (3) string predicates $\langle s, a, o\rangle, s \in V_{Q_V}, o \in V_{Q_K} \uplus V_{Q_V}, a \in \ell_a$. Query semantics are as defined for SPARQL BGPs. Namely, results are subgraphs of a data graph matching the graph pattern captured by a query. The only extension to these semantics are due the special keyword nodes: a value node in the data graph matches a keyword node if its bag of $n$-grams *contains* the keyword. Note, a query may also contain attribute predicates with non-string values such as date or time. We do not distinguish them from string predicates, because they can be regarded as a special case: attribute value nodes, against which these query predicates are evaluated, can be seen as bags with exactly one element. This one element would stand for the entire value. Such a value node "contains" the constant in the query predicate, if its value element exactly matches it. That is, evaluating such query predicate boils down to exact matching. An example query graph is illustrated in Fig. 2.

**Problem Definition.** Given a query $Q$, the selectivity, denoted by $sel(Q)$, is defined as the cardinality of $Q$'s result set. In this work, we address the problem of estimating $sel(Q)$, which can be decomposed into two tasks. First, there is $\mathcal{R}$, $\mathcal{R} : \mathcal{Q} \to \mathbb{N}$, giving an upper bound cardinality of a result set for query $Q \in \mathcal{Q}$. Second, a *probabilistic component* $\mathcal{P}$ defines a probability function mapping every $Q \in \mathcal{Q}$ to a probability. More precisely, $\mathcal{P}$ assigns a probability to a binary random variable, say $\mathbb{1}_Q$, modeling whether or not $Q$'s result set is non-empty. That is, $\mathbb{1}_Q$ captures whether

[3] `http://www.w3.org/TR/rdf-sparql-query/`

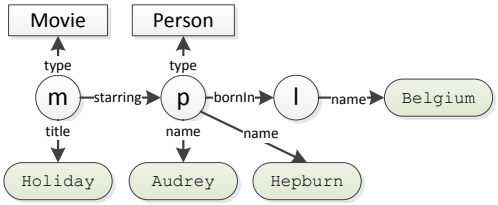



**Figure 2: A query graph asking for movies having a `title` "Holiday" and `starring` a person `named` "Audrey Hepburn", who is `born in` "Belgium".**

$Q$ holds. For simplicity, we write $\mathcal{P}(\mathbb{1}_Q = \mathbf{T})$ as $\mathcal{P}(Q)$. Overall, the selectivity is: $sel(Q) = \mathcal{R}(Q) \cdot \mathcal{P}(Q)$.

In previous works [10], $\mathcal{R}(Q)$ is estimated as size of the cross-product of the tables, which a query is evaluated over. In our setting, table names are not explicitly given in a query. To obtain $\mathcal{R}(Q)$, we consider an upper bound of results for every distinct query variable. That is, for each $v \in V_{Q_V}$, we upper-bound the number of its bindings, $R(v)$, as number of all entities belonging to class $c$: $|\{s|\langle s, type, c\rangle \in E\}|$. However, computing $R(v)$ like this requires $Q$ to contain a class predicate $\langle v, type, c\rangle$. If $v$ has no class assigned, we use the number of all entities, $|V_E|$, as an estimate for $R(v)$. Then, $\mathcal{R}(Q)$ is given by $\mathcal{R}(Q) = \prod_{v \in V_{Q_V}} R(v)$.

$\mathcal{P}(Q)$ captures the joint probability over a set of random variables – one for each query predicate $\in Q$. Intuitively, each such random variable models whether the associated query predicate holds or not. In Section 3, we present a template-based probabilistic model to obtain an accurate estimate for $\mathcal{P}(Q)$, while omitting any independence assumption. Before going into details, we first introduce *Bayesian networks* (BNs) and their *template-based representation* as the foundation of our approach. Note, while we rely on BNs for estimating $\mathcal{P}(Q)$, other kinds of graphical models are also applicable and may be used in a similar fashion.

**Probabilistic Framework.** A Bayesian network (BN) represents a directed graphical model that allows for a compact representation of joint distributions through two components: a network *structure* and model *parameters*. The former is given by a directed acyclic graph, where nodes stand for random variables and edges represent dependencies among them. Given parent nodes $Pa(X_i) = \{X_j, \ldots, X_k\}$, a random variable $X_i$ is dependent on $PA(X_i)$, but conditionally independent of all non-descendant nodes (random variables), i.e., all nodes which are not reachable from $X_i$ when removing $Pa(X_i)$. BN parameters comprise *conditional probability distributions* (CPDs) for random variables in the network. That is, each node $X_i$ is associated with a CPD capturing the conditional probability $P(X_i|Pa(X_i))$. A BN allows for computing the joint distribution $P(X_1, \ldots, X_n)$ via the chain rule: $P(X_1, \ldots, X_n) = \prod_i P(X_i|Pa(X_i))$. An example BN (more precisely, a template-based BN as discussed below) for our data graph is illustrated in Fig. 3-a. From its structure one can observe that, e.g., $\mathcal{X}_{directedBy}$ is dependent on $\mathcal{X}_{title}$ respectively $\mathcal{X}_{name}$, but independent of all other random variables given its two parents. Two example CPDs are shown in 3-c. Each row captures a probability, given one particular assignment to its parent random variables ($PA$). For instance, the CPD for $\mathcal{X}_{title}$ holds probabilities for $n$-grams of `title` values, conditioned on whether or not the particular entity having that value is a `Movie`.

We use template-based BNs [14, Ch. 6] (*template models*) as means to compactly represent correlations in graph-structured data. A template model is a framework featuring two parts: *template variables* (or *templates* in short) and *template factors*. Each template can be instantiated to obtain multiple random variables in a *ground* BN, which share the same sample space and the same semantics. Specifically, a template is defined as a function $\mathcal{X}(\alpha_1, \ldots, \alpha_k)$, whose sample space is $\Omega(\mathcal{X})$, and each argument $\alpha_i$ is a placeholder to be instantiated to obtain random variables. Fig. 3-a shows a template model, containing templates such as $\mathcal{X}_{movie}$, $\mathcal{X}_{starring}$, or $\mathcal{X}_{name}$, which are derived from classes, relations and attributes in our running example. A ground BN can be obtained using data in the graph for template instantiations. That is, placeholders $\alpha_i$ are instantiated by entities in the data, forming an *entity skeleton* of a template. Given a template $\mathcal{X}(\alpha_1, \ldots, \alpha_n)$, an entity skeleton of $\mathcal{X}$ is defined as $\mathcal{E}(\alpha_1, \ldots, \alpha_n) \subseteq \mathcal{E}(\alpha_1) \times \ldots \times \mathcal{E}(\alpha_n)$, where each $\mathcal{E}(\alpha_i) \subseteq V_E$ specifies all possible entity assignments to $\alpha_i$. Using entity skeletons, we can define a ground BN by instantiating a template as a set of random variables: $\mathbf{X} = \{X(e) | e \in \mathcal{E}\}$, where $\Omega(X(e)) = \Omega(\mathcal{X})$. For example, for the template $\mathcal{X}_{person}(\alpha_2)$ and $\mathcal{E}_{person}(\alpha_2) = \{p_1, p_2, p_3\}$, the set of random variables obtained for the ground BN is $\mathbf{X}_{person} = \{X_{person}(p_1), X_{person}(p_2), X_{person}(p_3)\}$.

Thus, different assignments to a template argument result in different random variables in the ground BN, which share the same probabilistic semantics. That is, they share the structure dependencies and parameters (CPDs) as defined for the template. The latter information is captured as *template factors*, which define probability distributions shared by all instantiated random variables of a given template.

Such a template-based representation is flexible as various ground BNs can be obtained for different entity skeletons based on the same templates. In our approach, we will exploit this flexibility to define a suitable ground BN for a given query, while relying on a fixed template model. Consider a "query" ground BN in Fig. 3-b. Random variables are instantiated for each query predicate, while having a query variable as placeholder for entity bindings. In the following, we will go into more details.

# 3. BN$^+$: STRING SYNOPSES + BN

In this section, we discuss why previous BN approaches [10, 23] for selectivity estimation over relational data do not directly fit a graph-structured setting. Instead, we propose a different template model for our probabilistic component $\mathcal{P}$. Further, we show how string synopses can be integrated into $\mathcal{P}$ to obtain a solution (BN$^+$) for supporting both, query predicates over structured as well as textual data.

## 3.1 BN$^+$ Template Model

Given a data graph $\mathcal{G} = (V, E, \ell_a, \ell_r)$, we define a template for each (1) *attribute* $a \in \ell_a$, $\mathcal{X}_a(\alpha_1)$, (2) *relation* $r \in \ell_r$, $\mathcal{X}_r(\alpha_1, \alpha_2)$, and (3) *class* $c \in V_C$, $\mathcal{X}_c(\alpha_1)$. For our running example templates are depicted in Fig. 3-a. Each template for a relation $r$ respectively a class $c$ is binary, i.e., $\Omega(\mathcal{X}_r) = \Omega(\mathcal{X}_c) = \{\mathbf{T}, \mathbf{F}\}$. On the other hand, a sample space for $\mathcal{X}_a$ comprises a bag of $n$-grams representation derived from attribute values of $a$. For instance, the sample space for $\mathcal{X}_{name}$ is given by $\Omega(\mathcal{X}_{name}) =$ {"Audrey", "Hepburn", "Audrey Hepburn", "Mel", ...}. To obtain a ground BN, these templates are instantiated using the following entity skeletons:

- The entity skeleton for a class template $\mathcal{X}_c(\alpha_1)$ is given by all entities belonging to that class: $\mathcal{E}_c(\alpha_1) = \{s | \langle s, type, c\rangle \in E\}$.
- For an attribute template an entity skeleton consists of all entities having that attribute: $\mathcal{E}_a(\alpha_1) = \{s | \langle s, a, o\rangle \in E_A\}$.
- Finally, the entity skeleton $\mathcal{E}_r(\alpha_1, \alpha_2)$ contains the pairs of source and target entities having relation $r$: $\mathcal{E}_r(\alpha_1, \alpha_2) = \{\langle s, t\rangle | \langle s, r, t\rangle \in E_R\}$. Let source and target entities be denoted as $\mathcal{E}_r^s(\alpha_1) = \{s | \langle s, r, t\rangle \in E_R\}$ and $\mathcal{E}_r^t(\alpha_2) = \{t | \langle s, r, t\rangle \in E_R\}$, s.t. $\mathcal{E}_r(\alpha_1, \alpha_2) \subseteq \mathcal{E}_r^s(\alpha_1) \times \mathcal{E}_r^t(\alpha_2)$, respectively.

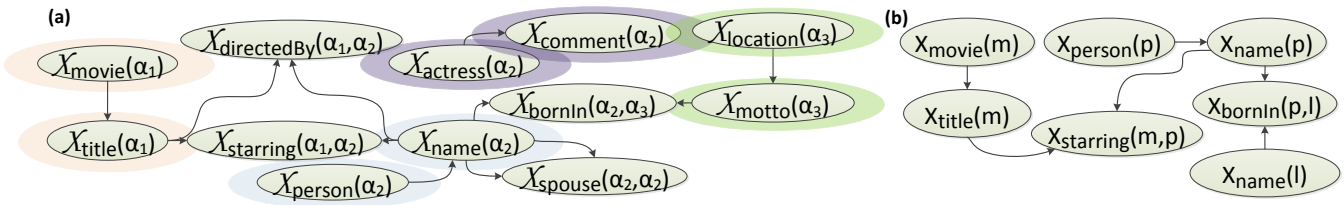
Such a template-based approach has the merit of being compact. The number of templates is far less than the number of random variables in a ground BN. Structure and parameters (CPDs) are learned for templates only. At runtime, templates are instantiated with entities to construct a ground BN. For inferencing, a CPD learned for a template is shared among all random variables in the ground BN that instantiate that template.

**Discussion.** In our work, we use a general template-based model as probabilistic framework. Previous instantiations of template-based models focus on relational data. Most notably, Probabilistic Relational Models (PRM) [8] and Probabilistic Entity Relation Models (PER) [9, Ch. 7] have been proposed. In fact, PRMs have also been applied for selectivity estimation [10, 23]. However, PRM-based solutions are not well-suited for a graph-structured setting, due to differences in the data as well as query model.

In a relational context, data is stored in tables corresponding to relations captured by a conceptual model. Further, relation names are explicitly given in a query – stated in a `FROM` clause. Correspondingly, previous works [10, 23] employ a PRM to model selection predicates through random variables of the form $X_{R.A}$, where $R$ is a relational table and $A$ is an attribute. For instance, $X_{Person.name}$ = "Audrey" is a random variable capturing a selection on table `Person` where `name` equals "Audrey". Analogously, join predicates are modeled as binary random variables that involve two explicitly specified tables.

As opposed to that, graph-structured data, such as RDF, can be partitioned in different ways. For instance, there may be a table for every entity class, e.g., a `Person` table capturing different person attributes [25]. On the other hand, a table might be constructed for every attribute respectively relation leading to, e.g., a table `name`. The latter partitioning is also known as *vertical partitioning* [1]. Thus, at query level, there is no explicit information about the tables from which data shall be selected. Further, schema information may be queried via class predicates, which are not supported in the relational setting. Due to these differences, the following problems occur when storing graph data in tables and applying a PRM-based solution:

- *Sensitivity to Data Partitioning:* A PRM assumes tables to be given. Thus, random variables are defined and their parameters/dependencies are learned, all of which w.r.t. these tables. Different partitioning schemes for data graphs, however, yield different tables. Therefore, models learned from such tables might largely vary – in terms of dependency structure as well as parameters. In particular, [23] focuses on learning correlations between attributes, which are comprised within one table, while assumptions are made to simplify cross-table dependencies. While resulting in a very "lightweight" PRM, this approach assumes that data is partitioned in tables comprising related attributes. In the case of vertical partitioning, how-




(c)

| $P(X_{title} \mid X_{movie})$ | $X_{title}(\alpha_1)$ | $X_{movie}(\alpha_1)$ |
|---|---|---|
| 1/3 | „Roman“ | T |
| 0 | „Roman“ | F |
| 1/3 | „Holiday“ | T |
| 0 | „Holiday“ | F |
| 1/3 | „Roman Holiday“ | T |
| 0 | „Roman Holiday“ | F |

| $P(X_{directedBy} \mid X_{title}, X_{name})$ | $X_{directedBy}(\alpha_1, \alpha_2)$ | $X_{title}(\alpha_1)$ | $X_{name}(\alpha_2)$ |
|---|---|---|---|
| 0 | T | „Roman“ | „Audrey“ |
| 0 | F | „Roman“ | „Audrey“ |
| 0 | T | „Roman“ | „Hepburn“ |
| 0 | F | „Roman“ | „Hepburn“ |
| 1/9 | T | „Roman“ | „William“ |
| 0 | F | „Roman“ | „William“ |
| 1/9 | T | „Roman“ | „Wyler“ |
| 0 | F | „Roman“ | „Wyler“ |
| ... | ... | ... | ... |

**Figure 3: (a) Template-based BN for the running example. (b) Query ground BN for query in Fig. 2. Note, templates $\mathcal{X}_{directedBy}$, $\mathcal{X}_{comment}$, $\mathcal{X}_{motto}$, and $\mathcal{X}_{location}$ are marginalized out. (c) CPD for template $\mathcal{X}_{title}$ respectively $\mathcal{X}_{directedBy}$.**

ever, where every attribute constitutes a table, there are no local dependencies to be learned and cross-table dependencies are more important. Generally speaking, the performance of PRM solutions is sensitive to the partitioning strategy. Our template-based solution does not make any assumptions about data partitioning. Instead, a template model is learned from entity skeletons and values from a data graph, independent from the way data is stored in tables.

- *Cross-Table Selection:* Besides vertical partitioning, another common strategy for graph data partitioning is to construct a table for every class [25]. However, oftentimes common attributes, such as `name`, are used to describe different types of entities, e.g., `Person` and `Location`. Given such a class-based partitioning, the attribute predicate $\langle p, name, \text{“Audrey”} \rangle$ would select data from different tables. However, these tables may not be explicitly specified in a query. At the same time, this explicit specification is required by PRM-based approaches. A possible solution is to maintain information to find out in which tables `name` occurs, and to construct corresponding random variables to refer to these different tables. Finally, one would need to aggregate the probabilities obtained for these variables. In contrast, with our template-based solution, only one template variable, $X_{name}$, is needed to support this predicate. A PRM-based approach, on the other hand, requires consulting one variable respectively CPD for every table.
- *Multi-Table Joins:* A similar problem occurs when dealing with joins. In a PRM context, a join predicate involves data from two explicitly specified tables. Such joins correspond to relation predicates in our setting. That is, a relation may be seen as referring to two foreign keys, which connect a source with a target entity. However, depending on the data partitioning, processing such a join (relation predicate), might involve one or more unspecified tables. For instance, given a relation predicate $\langle p, bornIn, l \rangle$, `bornIn` could join either `Person` or `Actress` entities with `Location` instances. Thus, data for the entities to be joined, might be located in different tables. As before, in a PRM one may handle this issue via using several random variables and aggregating their probabilities. Using our approach, however, merely one single CPD respectively random variable representing the given relation predicate is required.

## 3.2 String Synopses

Inferencing costs are driven by two factors: (1) dependency structure of a BN, and (2) sample space sizes. Existing works on PRMs have focused on the former, targeting a lightweight, tree-shaped BN structure [23]. The latter aspect, however, is crucial as CPD sizes are a mere reflection of sample space sizes. Essentially, for supporting string predicates with all possible keywords, $\Omega(\mathcal{X}_a)$ must capture all words and phrases, which occur in $a$'s values. Oftentimes, attribute values comprise long texts, resulting in a sample space to quickly blow up. In order to compactly represent $\Omega$, being a large set of strings, we propose the use of string synopses such as Markov tables [4], histograms [13] or $n$-gram synopses [24]. We generalize from existing works to define the following class of applicable synopses:

Definition 1 (String Synopsis). *A string synopsis for a template $\mathcal{X}_a$ is a tuple $\mathcal{S}(\nu, count)$. The synopsis function $\nu$ maps elements in the bag of $n$-grams for attribute $a$ (denoted by $\mathcal{B}_a$) to elements in a compact synopsis sample space $\Omega(\mathcal{X}_a)$. A function count: $\Omega(\mathcal{X}_a) \rightarrow \mathbb{N}$ returns the number of elements in the “original” space, $\mathcal{B}_a$, represented by a given synopsis element $\in \Omega(\mathcal{X}_a)$.*

Our definition of a synopsis is generic, however, a well-suited synopsis function $\nu$ should adhere to the following goals. First, $\nu$ should lead to a *small* sample space, $\Omega(\mathcal{X}_a)$, as a compact representation facilitates learning and keeps the CPD size manageable. Second, $\nu$ should be most accurate, i.e., each synopsis element $\in \Omega(\mathcal{X}_a)$ should represent only few $n$-grams from the original space, $\mathcal{B}_a$. Finally, a synopsis function should capture all “important” $n$-grams – while discarding not important ones. From a conceptual point of view, discarded $n$-grams are mapped to a bottom element $\perp$, capturing the probability mass for all missed $n$-grams.

While we do not restrict our approach to a particular type of string synopsis, recent work has shown that w.r.t. above goals, synopses based on *n-grams* are well-suited for the task of selectivity estimation for the *contains* operator on dictionaries [24]. This operator has the same semantics as our string predicate, matching text values that contain a given keyword. Thus, we follow this line of work and integrate $n$-gram synopses into BN$^+$ for our evaluation systems.

An $n$-gram synopsis function first projects a given textual attribute value to its $n$-gram representation $\subseteq \mathcal{B}_a$. For example, attribute `comment` has one attribute value, which would be mapped to $\mathcal{B}_{comment}$ = {"Audrey", "Hepburn", "Audrey Hepburn", ...}. Then, the space $\mathcal{B}_a$ is reduced by using a decision criterion to dictate which $n$-grams $\in \mathcal{B}_a$ to include in a synopsis sample space $\Omega(\mathcal{X}_a)$. That is, a synopsis space represents a subset of "important" $n$-grams. Note, $n$-gram synopses are most accurate, as each synopsis element represents exactly one $n$-gram $\in \mathcal{B}_a$ – in contrast to, e.g., histograms. Recent work has outlined several such decision criteria [24]. One simplistic strategy is to choose $\nu$ as a function that randomly *samples* $n$-grams from $\mathcal{B}_a$. Another approach is to construct a *top-k* $n$-gram synopsis. For this, $n$-grams are extracted together with their number of occurrences (counts). Then, the $k$ most frequent $n$-grams are included in the synopsis space. Considering attribute `comment`, the count for $n$-gram "Audrey" would be 2, while "Hepburn" only occurs once. Thus, a top-$k$ $n$-gram synopsis would rank "Audrey" as more important than "Hepburn". Further, a *stratified bloom filter* synopsis has been proposed [24], which uses bloom filters as a heuristic map that projects $n$-grams to their counts.

## 3.3 BN$^+$ Construction

Our template-based BN$^+$ should compactly represent the joint distribution over templates capturing structured data elements as well as $n$-grams of textual attribute values. However, large sample spaces and complex dependencies among templates may lead to prohibitive storage and inferencing costs. In fact, during our experiments we observed sample space sizes up to 2 million $n$-grams for some attributes. Such sample spaces translate to large CPDs, which in turn make fast inferencing at runtime impossible. Furthermore, dependencies between templates aggravate this problem: the size of a CPD multiplies, with each parent a particular template is dependent on. We aim for a *compact template model* via exploiting two strategies. Firstly, we utilize string synopses in order to "compress" an attribute template sample space into a manageable size. Secondly, instead of constructing a complex network structure featuring all possible dependencies, we solely focus on the most important ones. That is, we aim for an approximation of the joint distribution that shall limit the dimensions of the CPDs, while preserving key dependencies.

**Structure Learning.** An efficient and well-known technique in BN literature [5, 18] is based on using a *product approximation* of rich structures via trees. These tree structures guarantee that each template has at most one parent. Recently, such approximation has been adopted to PRMs for a relational setting. The resulting "lightweight" structure has shown to improve efficiency, while still producing high-quality estimates [23]. We apply product approximation to a graph-structured setting by imposing a fixed structure of independences among template variables:

DEFINITION 2 (FIXED STRUCTURE). *Given a data graph, the following conditional independences hold: (a) two templates $\mathcal{X}_1$ and $\mathcal{X}_2$ are conditionally independent given their parents, if they do not share a common entity in their skeletons $\mathcal{E}_1$ and $\mathcal{E}_2$. Note, in case either of these templates, say $\mathcal{X}_i$, captures a relation, we use $\mathcal{E}_i = \mathcal{E}_i^s \cup \mathcal{E}_i^t$ as the skeleton, i.e., the union of its source and target entities. (b) Each class template $\mathcal{X}_c$ has no parent. (c) Each relation template $\mathcal{X}_r$ is independent of any class template $\mathcal{X}_c$, given its parents.*

**Algorithm 1:** Construct BN$^+$ template model.

**Input**: Templates $\mathcal{X} = \{X_a\}_{\forall a \in \ell_a} \cup \{X_r\}_{\forall r \in \ell_r} \cup \{X_c\}_{\forall c \in V_C}$, entity skeletons $\boldsymbol{\mathcal{E}} = \{\mathcal{E}_a\}_{\forall a \in \ell_a} \cup \{\mathcal{E}_r\}_{\forall r \in \ell_r} \cup \{\mathcal{E}_c\}_{\forall c \in V_C}$ and synopsis size $k$.
**Output**: Template model $\mathcal{T}$

```
1  begin
2      T_local ← ∅
3      Ω(X_r) = Ω(X_c) = {T, F} for all c and r
4      Ω(X_a) = InitializeSynopsis(a, k) for all a
5      foreach X_a ∈ X do
6          foreach non-independent X_c to X_a w.r.t. Def. 2 do
7              Add (X_c --mi(X_c,X_a)--> X_a) to T_local
8          foreach non-independent X_a' to X_a w.r.t. Def. 2 do
9              Add (X_a' --mi(X_a',X_a)--> X_a) to T_local
10     T*_local = MAX-SPANNING-FOREST(T_local)
11     T ← T*_local // initialize T
12     foreach X_r ∈ X do
13         X_{a_s}^best = arg max_{X_a ∧ a ∈ source(r)} mi(X_a, X_r)
14         X_{a_t}^best = arg max_{X_a ∧ a ∈ target(r)} mi(X_a, X_r)
15         if X_{a_s}^best ≠ null then
16             Add (X_{a_s}^best → X_r) to T
17         if X_{a_t}^best ≠ null then
18             Add (X_{a_t}^best → X_r) to T
19     return T
```

We argue that the independences induced via our fixed structure are meaningful due to two reasons. (1) We impose that strong correlations among templates only occur, if they share some common entities – they need to "talk about the same things" (Def. 2-a). (2) We argue that there is a causal dependence (independence) between a class and an attribute (relation) template (Def. 2-b, -c). In other words, assigning an entity to a given class causally affects the probability of its attribute values, which in turn, influences the probability of observing a particular relation.

Exploiting the fixed structure, we can decompose structure learning: we construct a disconnected graph, coined *local part* ($\mathcal{T}_{local}$), of the template model by learning dependencies between class/attribute respectively attribute/attribute template pairs. Then, we simplify $\mathcal{T}_{local}$ via an approximation $\mathcal{T}^*_{local}$. Finally, we add relation templates to the structure $\mathcal{T}^*_{local}$ and obtain a final template model $\mathcal{T}$.

For learning the local part, we add weighted edges between each class/attribute (attribute/attribute) template pair, which is not independent w.r.t. the fixed structure. That is, each pair has an "overlap" in their skeletons – they share one or more entities. In our example, $\mathcal{T}_{local}$ comprises,

e.g., a tree $\mathcal{X}_{movie} \rightarrow \mathcal{X}_{title}$, as both skeletons are equal to $\{m_1\}$ (Fig. 3-a). In order to calculate the dependency weight between two templates we use the *mutual information* ($mi$) quantity, which essentially represents the amount of information shared between the given templates:

$$mi(\mathcal{X}_1, \mathcal{X}_2) = \sum_{x_1 \in \Omega_1} \sum_{x_2 \in \Omega_2} P(x_1, x_2) \log \left( \frac{P(x_1, x_2)}{P(x_1)P(x_2)} \right)$$

with $\Omega_i = \Omega(\mathcal{X}_i)$ being the sample space of template $\mathcal{X}_i$. The maximum likelihood estimation of $P(\mathcal{X}_i = x_i)$ is:

$$P(\mathcal{X}_i = x_i) = \frac{\breve{M}_i[x_i]}{N}$$

with $N$ as normalization factor. $\breve{M}$ is a *sufficient statistic*[4] counting entities in the skeleton of $\mathcal{X}_i$ having $x_i$ as value:

$$\breve{M}_i[x_i] = \sum_{e \in \mathcal{E}_i} \mathbb{1}\{X_i(e) = x_i\}$$

Note, $\mathbb{1}$ is an indicator function, i.e., it returns 1 if its expression is true, 0 otherwise. Similarly, the joint distribution of $\mathcal{X}_1$ and $\mathcal{X}_2$ is:

$$P(\mathcal{X}_1 = x_1, \mathcal{X}_2 = x_2) = \frac{\breve{M}_{1,2}[x_1, x_2]}{N}$$

with $N$ as normalization factor and $\breve{M}_{1,2}[x_1, x_2]$ as count of entities having both values:

$$\breve{M}_{1,2}[x_1, x_2] = \sum_{e \in \mathcal{E}_1 \cap \mathcal{E}_2} \mathbb{1}\{X_1(e) = x_1, X_2(e) = x_2\}$$

Once the weighted edges are added to $\mathcal{T}_{local}$, the model comprises all possible dependencies between class/attribute templates according to our fixed structure. Then, we capture only the most important correlations in $\mathcal{T}_{local}$ by reducing it to its *maximum-spanning forest*[5]. This yields a simpler structure, $\mathcal{T}^*_{local}$. For our example, $\mathcal{T}^*_{local}$ is depicted in Fig. 3-a: four spanning trees are highlighted. Note, due to our fixed structure restriction, the maximum-spanning forest algorithm may find no solution. In such cases, we iteratively remove the weakest attribute/attribute weighted edges pair until a spanning tree can be obtained. For instance, while the pink and blue tree have been comprised in one component in $\mathcal{T}_{local}$, we needed to remove the weighted edges between $\mathcal{X}_{name}$ and $\mathcal{X}_{comment}$, leading to two trees in $\mathcal{T}^*_{local}$ (Fig. 3-a). Overall, the construction of $\mathcal{T}^*_{local}$ results in a dynamic partitioning of the dependencies based on information contained in entity skeletons.

We now incorporate relations. Mutual information is used to quantify dependencies between relation and attribute templates comprised in trees in $\mathcal{T}^*_{local}$. For every relation template, its mutual information w.r.t. all possible (non-independent, Def. 2) source respectively target attribute templates is computed. The two attribute templates that exhibit the highest mutual information are used as parents of that relation template. In Fig. 3-a $\mathcal{X}_{directedBy}$ connects two trees (red and blue) via $\mathcal{X}_{title}$ and $\mathcal{X}_{name}$ as parents.

Algorithm 1 captures the entire procedure for structure learning. The algorithm takes the set of all templates, the entity skeletons and the string synopsis size parameter $k$ as inputs. Then, it initializes the sample spaces of all templates (lines 3-4). Note, initializing the sample space for an attribute template requires string synopsis construction as discussed before. Local template model learning is performed in lines 5-9 by adding weighted edges. The approximation of $\mathcal{T}_{local}$ is done in line 10 by solving the maximum-spanning forest problem and adding its result, $\mathcal{T}^*_{local}$, to an intermediate model. The connections between trees in $\mathcal{T}^*_{local}$ are constructed in lines 12-18, resulting in a final model.

THEOREM 1. *The template model constructed according to Algorithm 1 is acyclic.*

*Proof Sketch:* A local model is reduced to a forest of trees, $\mathcal{T}^*_{local}$, via a maximal spanning tree algorithm. Thus, every tree in $\mathcal{T}^*_{local}$ represents a valid acyclic fragment of our network. Then, we connect these tree structures by incrementally adding (lines 12-18) edges representing relation templates. However, each relation template must not have children. Thus, no cycles can be introduced at this step ■

**Parameter Learning.** Having build a network structure, we now learn model parameters, i.e., conditional probability distributions. As done in recent works [10, 23], learning CPDs can be achieved based on our sufficient statistic, $\breve{M}$. More precisely, according to Bayes rule it holds that: $P(\mathcal{X}_i|\mathcal{X}_j) = \frac{P(\mathcal{X}_i, \mathcal{X}_j)}{P(\mathcal{X}_j)}$. Thus, we may compute $P(\mathcal{X}_i, \mathcal{X}_j)$ respectively $P(\mathcal{X}_j)$, as we did for obtaining the mutual information. Note, in case of a relation template, say $\mathcal{X}_i$, we need to estimate a distribution conditioned on two other templates: $P(\mathcal{X}_i|\mathcal{X}_j, \mathcal{X}_k)$. This can be achieved by extending our $\breve{M}$ function to capture three templates: $\breve{M}_{1,2,3}[x_1, x_2, x_3]$.

For an efficient parameter learning, we employ two sorts of optimizations. First, we use caching strategies for keeping frequently needed $\breve{M}$ statistics in memory. In fact, caching may be applied to store results already produced during mutual information computation. For example, sufficient statistics for the template $\mathcal{X}_{title}$ are needed at least twice, as it is a parent of $\mathcal{X}_{directedBy}$ as well as $\mathcal{X}_{starring}$. Second, we can formulate $\breve{M}$ expressions as queries to be issued at a database. For instance, $\breve{M}_{p_1,p_2}[x_1, x_2]$ can be calculated based on the cardinality of a result set for query $\{\langle s, p_1, x_1\rangle, \langle s, p_2, x_2\rangle\}$, with $\mathcal{X}_{p_i}$ being the template for $p_i$.

**Maintenance.** With evolving data, triples might be added or removed from a graph. On the one hand, such changes may result in minor modifications of entity skeletons or sample spaces. As a consequence, some model parameters may no longer be accurate enough for effective selectivity estimation. Such affected CPDs should be recomputed given an updated data graph. For minor changes such a reestimation, however, does not influence other parameters and/or the structure and thus, can be performed incrementally. In fact, while frequent changes to model parameters might have to be calculated, a network structure is more "stable". On the other hand, given drastic changes to a data graph, its structure as well as parameters have to be recomputed. Our experiments show that even in this case, learning is feasible within a short amount of time. We observed that computation of the entire BN$^+$ model, including string synopses, took at most 3 hours.

## 3.4 Selectivity Estimation

In order to employ BN$^+$ for selectivity estimation, its templates are instantiated by given query predicates to form a ground BN. To be precise, we do not model a standard ground BN, since this would solely capture entities as random variable assignments. Instead, in a "query" ground BN, random variables have *sets* of entities as assignments, which

[4]Sufficient statistic is a term from the statistical learning literature, which refers to frequencies or counts.

[5]A maximum-spanning forest is defined in our work as a collection of spanning trees, one for each component in $\mathcal{T}_{local}$.

are result bindings to query variables. For each relation predicate $\langle s, r, o\rangle$ and class predicate $\langle s, type, c\rangle$ in $Q$, we instantiate a random variable $X_r(s, o) = \mathbf{T}$ and $X_c(s) = \mathbf{T}$, respectively. For every string predicate $\langle s, a, w\rangle$, its keyword $w$ is mapped to a corresponding element in our synopsis, $\nu(w)$, such that a resulting instantiated variable is $X_a(s) = \nu(w)$. As for the running example, we instantiate one random variable for each query predicate as shown in Fig. 3-b. Notice that we need two instantiations of $\mathcal{X}_{name}$. Templates which are not relevant to the query, e.g., $\mathcal{X}_{directedBy}$, are marginalized out. For selectivity estimation, the joint probability of these random variables is computed as an inference problem:

$$P(Q) \approx \gamma \cdot P\left(\bigwedge X_a(s) = \nu(w) \bigwedge X_c(s) = \mathbf{T} \bigwedge X_r(s, o) = \mathbf{T}\right)$$

where $\gamma = {}^1/_{\prod_w count(\nu(w))}$ is a correction factor. $\gamma$ is necessary as $\nu(w)$ may not only capture the probability mass for $w$, but could also include other words (phrases). Consider a histogram synopsis. Here, "Wiliam" and "Wyler" could be represented by a single bucket $[Wi-Wy]$. Then, a query predicate $\langle p, name,$ "Wiliam" $\rangle$ would be translated to $X_{name}(p) = [Wi-Wy]$. However, the bucket $[Wi-Wy]$ not only comprises "Wiliam", but also "Wyler". Thus, its probability must by "corrected". Note, such a correction implies a uniform distribution among all words (phrases), which are captured by a single synopsis element.

For the above inferencing problem, each instantiated random variable reuses the CPD from its template. In the simplest case, inferencing for $P(Q)$ could be performed via "brute-force" marginalization. However, as marginalization is expensive, we employ belief propagation allowing an approximation, which operates on a *junction tree* representation of the ground BN [23]. We adopt the inferencing to deal with the following problems that arise in our setting:

**Multiple Value Assignments.** Commonly, a string synopsis restricts the length of its phrases, due to a limited amount of space. If a query predicate contains a phrase as keyword, which is longer than this threshold, a simple strategy is to break that phrase into multiple smaller phrases. For instance, if a synopsis only allows 1-grams, a keyword phrase with $k$ words must be split into $k$ 1-grams. In such cases, instantiated random variables (referring to the same query variable) have multiple values. For instance, $X_{name}(p)$ has "Audrey" and "Hepburn" as values (Fig. 2 and 3-b). This problem can be addressed through an *aggregation function*. We use a *stochastic mode* aggregation, which uses all values as evidence, but weights each one with its frequency within the query [22]. In our example, $P(X_{bornIn} = \mathbf{T}|X_{name}(p))$ is given by $\frac{1}{2} \cdot P(X_{bornIn} = \mathbf{T}|X_{name}(p) =$ "Audrey") $+ \frac{1}{2} \cdot P(X_{bornIn} = \mathbf{T}|X_{name}(p) =$ "Hepburn"). Note, the probability for each $X_{name}(p)$ assignment is weighted with $\frac{1}{2}$, as both values occur once in the query.

**Missing Synopsis Values.** There are synopses, such as the top-$k$ $n$-gram, for which some query keywords do not have a corresponding synopsis element. That is, the synopsis discarded that particular word (phrase) during construction for space reasons. The probability for these "missing" keywords cannot be estimated within BN$^+$, as they are not included in any sample space. To deal with this case, a string predicate featuring a missing keywords is assumed to be independent from the remainder of the query. Then, its probability can be estimated based on a string synopsis heuristic [24]. We employ the *leftbackoff* strategy, which finds the longest known $n$-gram that is a prefix (postfix) of the missing keyword and estimates its probability based on statistics for that prefix (postfix) [24].

# 4. EVALUATION

In the following, we present results of experiments we performed to analyze the *accuracy* (effectiveness) and the *time performance* (efficiency) of BN$^+$. As baseline, we used an approach that assumes independence among string predicates as well as between them and structured query predicates. Overall, our results suggest this baseline yields very low accuracy, when dependencies between query predicates exist. For IMDB we observed such strong correlations in the data. Here, given we employ the most accurate string synopsis (stratified bloom filters), BN$^+$ improved the baseline's accuracy by 93% in terms of multiplicative error. In other words, BN$^+$ achieved a decrease of error by a factor of 13.6. With respect to efficiency, we found that the BN inferencing overhead was actually negligible. The main factor driving computation time was the string synopsis we employed. When both approaches, BN$^+$ respectively the baseline, used the same type of string synopsis, their performance was comparable.

## 4.1 Setup

**Data.** We used two real-world datasets: DBLP comprising computer science bibliographies and IMDB holding information from the movie domain. Tab. 1 provides basic statistics for both datasets. DBLP as well as IMDB hold text-rich attributes like `name`, `label` or `info`. We employed $n$-gram string synopses as presented in [24]. However, we only used 1-grams in our experiments, as a larger values for $n$ resulted in synopses that exceed our memory space limit. Overall, we extracted $25,540,172$ and $7,841,347$ 1-grams from DBLP and IMDB, respectively. We chose these two datasets, as in one of them (IMDB) textual attribute values strongly correlate among each other respectively with structured data. In particular, we noticed strong dependencies during structure learning between values of attributes such as `label` and `info`. Hence, IMDB is appropriate to test our hypothesis that assuming independence hurts the quality of selectivity estimates, given datasets that exhibit correlations. We also used DBLP, which on the other hand, shows almost no such correlations. Using DBLP data, we expect accuracy differences to be less significant. Comparing the accuracy performances across such two datasets shall illustrate the relative benefits of our solution.

**Listing 1: Example queries for IMDB and DBLP. Variables are red, keywords blue and classes respectively predicates black.**

```
// imdb query
<x,type,Title>
<x,title,"star">
<x,title,"trek">
<x,cast_info,c>
<c,type,Cast_info>
<c,role,r>
<r,name,rn>
<r,type,Char_name>
<c,person,p>
<p,type,Person>
<p,name,"brent">
<p,name,"spiner">
```

```
// dblp query
<x,label,"clustering">
<x,label,"mining">
<x,year,"2005">
<x,type,Article>
<x,author,y>
<y,type,Person>
<y,name,"nikos">
```

**Queries.** We employed queries that have been used for keyword search evaluation. These queries capture information needs expressed as keywords. Based on query keywords and their structured results, we constructed corresponding

| | IMDB | DBLP |
|---|---|---|
| # Triples | 7,310,190 | 11,014,618 |
| # Resources | 1,673,097 | 2,395,467 |
| # 1-grams | 7,841,347 | 25,540,172 |
| # Attributes | 11 | 21 |
| # Relations | 8 | 18 |
| # Classes | 6 | 18 |

**Table 1: Dataset statistics.**

| Predicates: | #Relation | | | #String | | |
|---|---|---|---|---|---|---|
| | 0 | 1 | [2, 4] | [1, 2] | 3 | [4, 7] |
| # Queries | 33 | 44 | 23 | 28 | 35 | 26 |
| Predicates: | #Class | | | #Total | | |
| | 1 | 2 | [3, 4] | [2, 3] | [4, 6] | [7, 11] |
| # Queries | 49 | 30 | 21 | 28 | 31 | 41 |

**Table 2: Query statistics providing the number of relation, string and class predicates contained in our query load.**

graph patterns, comprising string, class, and relation predicates. In particular, we generated 54 DBLP queries based on [16]. Additionally, 46 queries were constructed for IMDB, based on a recent keyword search benchmark [6]. We omitted 4 queries from [6], as they could not be translated to our query model. Our workload includes queries containing $[2, 11]$ predicates in total: $[0, 4]$ relation, $[1, 7]$ string, and $[1, 4]$ class predicates (cf. Tab. 2). Note, since we extracted 1-grams only, every string predicate with a phrase of length $n$ is decomposed into $n$ string predicates, each capturing exactly one word. In our subsequent analysis, we rely on the number of predicates as an indicator for query complexity. We expect queries with a larger number of predicates to be more "difficult" in terms of both accuracy and efficiency. That is, accurate estimates may be harder to obtain and require additional computation. However, most crucial are actually dependencies between query predicates: we observed that there are more correlated predicates in IMDB, e.g., `info` (class `Movie`) and `title` (class `Movie`). Queries in DBLP, on the other hand, often include, e.g., `name` (class `Author`) and `label` (class `Title`) predicates, for which we could not measure any significant correlations. Tab. 2 gives an overview of our query load, while example queries are given in Listing 1. All queries can be found in our appendix.

**Systems.** As string synopses we employed strategies proposed in [24]. That is, we obtained a random sample of 1-grams, top-$k$ 1-grams and stratified bloom filters (sbf) on 1-grams. For selectivity estimating of the entire query, string predicates were integrated via (1) *independence* (`ind`) or (2) *conditional independence* (`bn`) assumption. In the former case, selectivity of string and relation/class query predicates was estimated using string synopses and histograms, respectively. More precisely, structured query parts were estimated similar to [11]. In the latter case, selectivity estimation was performed using BN$^+$. Combining string synopses with the (conditional) independence assumption resulted in seven different systems: $\texttt{ind}_{\text{sample}}$, $\texttt{ind}_{\text{top-}k}$ and $\texttt{ind}_{\text{sbf}}$ rely on the independence assumption, and $\texttt{bn}_{\text{sample}}$, $\texttt{bn}_{\text{top-}k}$ as well as $\texttt{bn}_{\text{sbf}}$ represent BN$^+$ approaches.

**Synopsis Size.** We experimented with synopses of various sizes. The key factor driving the overall synopsis size was the employed string synopsis. The string synopsis determined the size of the (conditional) probability distribution for $\texttt{ind}_*$ ($\texttt{bn}_*$), which was the most costly type of statistic – other statistics, e.g., the BN network structure, were negligible in terms of space. We varied the number of 1-grams comprised by the top-$k$ and sample synopsis, i.e., #1-grams per attribute $\in \{0.5K, 1K, 5K, 10K\}$. Regarding our sbf approach, we captured up to $\{2.5K, 5K, 25K, 50K\}$ of the most frequent 1-grams for each attribute and varied the bloom filter sizes, resulting in similar memory requirements. All systems loaded their synopsis into main memory. Overall, different string synopses (sizes) yielded different systems with $\{2, 4, 20, 40\}$ MByte of memory consumption, while no additional hard disk space was required. We observed that while selectivity estimations become more accurate with greater size, no further improvements could be achieved, using synopses $\geq 20$ MByte. In order to allow for the best accuracy and to illustrate this convergence, we report results from synopses with up to 40 MByte.

**Implementation and Offline Learning.** For $\texttt{bn}_*$ we used the BN construction procedure as discussed in sect. 3. That is, we learned a model structure, capturing the most important correlations only. Then, we calculated model parameters (CPDs) based on sufficient statistics. String synopsis construction could be done efficiently: each synopsis, including sbf-based synopses, could be computed in less than one hour. Structure and parameter learning for $\texttt{bn}_*$ combined took in the worst case up to three hours. Inferencing needed by our systems was done using a Junction tree algorithm [23].

As $\texttt{bn}_*$ and $\texttt{ind}_*$ systems rely on the same probability distributions for string predicates, parameters were shared. That is, $\texttt{ind}_*$ approaches did not need a BN model structure, but merely kept its marginalized parameters. Further, histograms for $\texttt{ind}_*$ comprising relation respectively class statistics were constructed similar to [11].

Model structure (histograms) as well as parameters for $\texttt{bn}_*$ ($\texttt{ind}_*$) were stored in a key-value store outside the database system – both were loaded into memory at start-up. Depending on the synopsis size loading took up to 3s.

We implemented all systems and algorithms using Java 6. Experiments were run on a Linux server with two Intel Xeon 5140 CPUs (each with 2 cores at 2.33GHz), 48GB RAM (with 16GB assigned to the JVM), and a RAID10 with IBM SAS 148GB 10k rpm disks. Before each query execution, all operating system caches were cleared. The presented values are averages collected over five runs.

## 4.2 Selectivity Estimation Effectiveness

As measurement for selectivity estimation accuracy, we employed the *multiplicative error* metric (*me*) used in previous work [7]. The multiplicative error is defined as

$$me(Q) = \frac{\max(sel(Q), \widetilde{sel}(Q))}{\min(sel(Q), \widetilde{sel}(Q))}$$

with $sel(Q)$ and $\widetilde{sel}(Q)$ as exact and estimated selectivity for $Q$, respectively. Intuitively, *me* represents the factor at which $\widetilde{sel}(Q)$ under-/overestimates $sel(Q)$.

**Overall Results.** Fig. 4-a, -b (-e, -f) depict the multiplicative error for DBLP (IMDB). Best accuracy results were achieved by $\texttt{ind}_*$ and $\texttt{bn}_*$ having a size $\geq 20$ MByte, as such synopses had sufficient space to capture most query keywords. Further, the results confirmed our conjecture that the degree of data correlations has a significant impact on

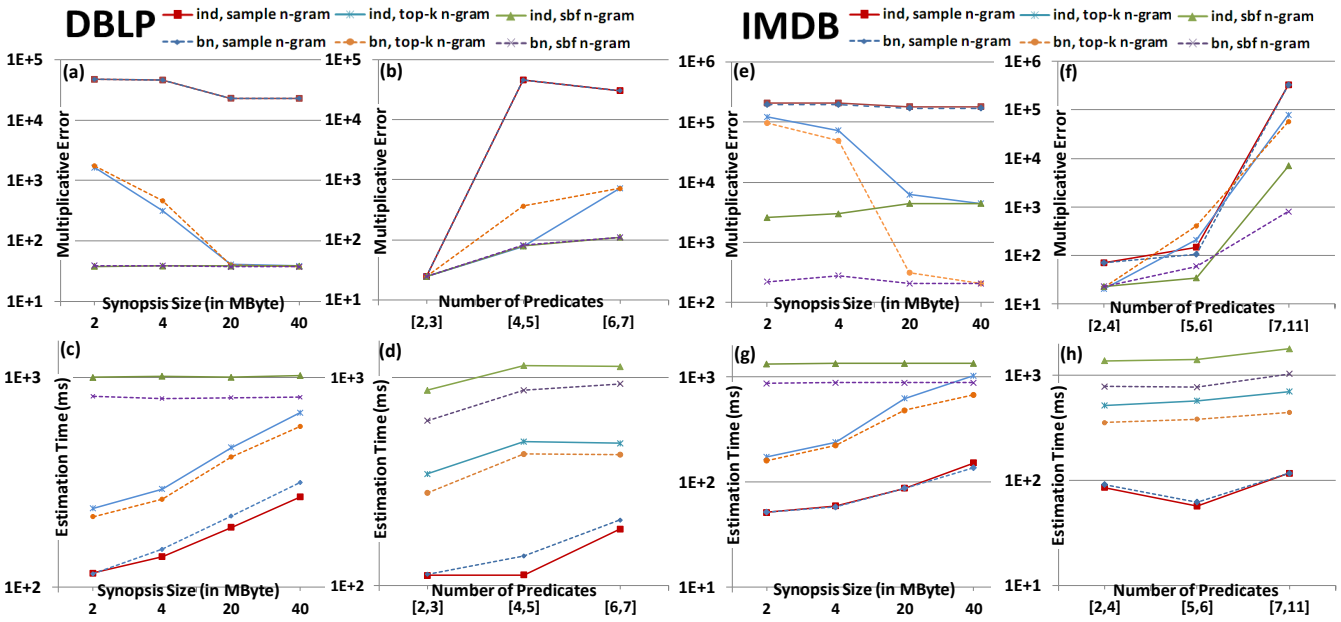



**Figure 4: Evaluation results for DBLP and IMDB. All y-axes are in logarithmic scale.**

the overall accuracy performance differences between $\texttt{ind}_*$ and $\texttt{bn}_*$ approaches. That is, a high degree of correlation in the IMDB dataset translated to large accuracy differences, while the improvement $\texttt{bn}_*$ could achieve over the baseline was small for DBLP. Last, comparing $\texttt{ind}_*$ ($\texttt{bn}_*$) systems in terms of their string synopsis, we found that sampling-based approaches were outperformed by systems using top-$k$ 1-gram synopses. Such systems, in turn, performed worse than sbf-based approaches. In fact, when using samples, the $\texttt{bn}_{\text{sample}}$ system achieved results similar to the one from $\texttt{ind}_{\text{sample}}$. This behavior is due to the fact that many keywords in query predicates were "missed" in the sample synopses. In these cases, both approaches rely on similar heuristics (*leftbackoff* strategy) to calculate the probability for such keywords, which translated to large misestimates.

**Synopsis Size.** Fig. 4-a and -e depict estimation errors w.r.t. different synopsis sizes for DBLP and IMDB, respectively. Given a small synopsis ($\leq$ 4 MByte), we observed that top-$k$ and especially sample-based systems performed poorly, while results for sbf-based approaches were fairly stable. With increasing synopsis size ($\in [4, 20]$ MByte), the performance of top-$k$ 1-gram approaches converged to the most accurate selectivity estimations achieved by sbf-based systems. Differences in estimation quality can be explained by missed query keywords. More precisely, when missing a keyword, approaches have to rely on inaccurate heuristics for probability computation. The relatively good and stable performance of sbf-based systems suggest that using stratified bloom filters is an effective strategy providing enough space for most relevant 1-grams.

**Data Correlations.** Results obtained for IMDB and DBLP largely varied. For the IMDB dataset, $\texttt{bn}_{\text{sbf}}$ could reduce errors of the $\texttt{ind}_{\text{sbf}}$ approach by 93 %, while improvements were much smaller given DBLP. For instance, for DBLP queries with string predicates `name` and `label`, there are no significant correlations in our BN. Thus, the probabilities obtained by $\texttt{bn}_*$ were almost identical to the ones from $\texttt{ind}_*$. However, while $\texttt{ind}_*$ led to fairly good estimates for the overall query load on DBLP, we could achieve more accurate selectivity computations via $\texttt{bn}_*$ for specific "correlated" queries. For instance, for DBLP query Q1 we could approximate an 10% better selectivity estimation.

**Query Size.** Fig. 4-b and -f show the multiplicative error for a varying number of query predicates. We noticed the error to increase in the number of predicates. This effect is expected, as more query predicates (hence more "difficult" queries) lead to an increasingly error-prone probability estimation. An interesting observation is that $\texttt{ind}_*$ outperformed $\texttt{bn}_*$ for some queries – see IMDB queries with 5 predicates and DBLP queries with 4 predicates (Fig. 4-b and -f). For instance, given IMDB query Q28, $\texttt{ind}_{\text{top-}k}$ achieved 13% better results than $\texttt{bn}_{\text{top-}k}$. In such cases, string query predicates were translated to multiple values (1-grams) that are assigned to one single random variable. For processing these multiple assignments, $\texttt{bn}_*$ employed value aggregation. However, the stochastic mode aggregation led to over-/underestimations for these queries due to inaccurate evidence weights. On the other hand, $\texttt{ind}_*$ systems could approximate the probability simply via independence assumption. Overall, we observed that while stochastic mode aggregation resulted in worse estimates for some queries, it led to better results on average.

## 4.3 Selectivity Estimation Efficiency

During the second part of our experiments, we studied efficiency aspects of selectivity estimation for varying synopsis sizes (Fig. 4-c and -g) and query complexities (Fig. 4-d and -h). For all systems, our reported times represent solely the inference task, while times for model construction and loading were omitted.

**Overall Results.** An important observation is that BN inferencing did not have a decisive impact on the overall performance. Instead, the employed string synopsis was a key factor driving the efficiency: systems with sample-based synopses, $\texttt{bn}_{\text{sample}}$ and $\texttt{ind}_{\text{sample}}$, were faster than approaches relying on top-$k$ 1-gram synopses, which in turn outperformed sbf-based systems $\texttt{bn}_{\text{sbf}}$ and $\texttt{ind}_{\text{sbf}}$. In fact, when

employing the same string synopsis, $\texttt{bn}_*$ approaches led to computation times comparable to those from $\texttt{ind}_*$. This can be explained with the lightweight model structure used by $\texttt{bn}_*$, which only captures the most important correlations. Further, our structure contained many tree-shaped parts, which could be processed efficiently through Junction tree inferencing.

Interestingly, we noticed $\texttt{ind}_*$ systems to be even slower than $\texttt{bn}_*$ in some cases. We explain this with (1) a computational overhead of histogram-based estimation of structured query constraints for $\texttt{ind}_*$, and (2) with runtime advantages of $\texttt{bn}_*$ due to stochastic aggregation. That is, fewer probability computations were performed by $\texttt{bn}_*$, because through value aggregation, the system could process several string predicates via one single inference task. On the other hand, $\texttt{ind}_*$ approaches needed to compute the probability for each string predicate individually. For instance, $\texttt{bn}_*$ needed 30% less computation time compared to $\texttt{ind}_*$ for Q33 in the IMDB query load. This is because Q33 contains 7 `info` string predicates that were aggregated by $\texttt{bn}_*$, leading to one single random variable assignment.

**String Synopses.** Compared to other synopses, the time savings achieved with sample-based systems were possible due to missing 1-grams. However, such savings came at the expense of accuracy. If a particular query keyword is not included in a synopsis, heuristics are employed. In this case, the probability computation is done without the use of (conditional) probability distributions. Thus, no time-consuming marginalization was needed. Further, the missing 1-gram could not be added to the model "as evidence" for further inferencing. Sbf-based systems performed worst. We explain this behavior with the computational overhead introduced by bloom filters. Further, as sbf synopses comprised a larger number of 1-grams, marginalization was more expensive. Note, however, with an increasing number of 1-grams to be managed, the performance of sample-based respectively top-$k$ systems converged to the one exhibited by sbf-based approaches.

**Synopsis Size.** Fig. 4-c and -g show selectivity estimation time vs. synopsis size. As expected, larger string synopses translated to bigger (conditional) probability distributions and hence, resulted in longer inference times. Sbf-based approaches are an exception, as they provided a stable performance for different synopsis sizes. This constant estimation time was due to the fact that computational costs for sbf systems are largely determined by their bloom filters. In fact, we observed that costs only marginally depended on the overall number 1-grams.

**Query Size.** Fig. 4-d and -h show that selectivity estimation times increase with query size. This is because each additional query predicate translated to more inferencing iterations and probability lookups that were needed by $\texttt{bn}_*$ and $\texttt{ind}_*$ systems, respectively.

# 5. RELATED WORK

For better accuracy, selectivity estimation approaches aim to avoid uniform distribution, predicate value independence and join predicate independence assumptions.

One line of research employs *table-level* data synopses, i.e., data reduction techniques capturing joint distributions of attribute values within a table. Previous approaches utilize, e.g., histograms [7, 19] or wavelets [17]. Such table-level approaches are suitable for addressing the uniform distribution and predicate value independence assumption. However, join independence assumptions can not be omitted, as these synopses are restricted to a single table and do not incorporate foreign-key relations.

Another line of research is concerned with *schema-level* synopses. Here, a synopsis does not only capture a single table, but also related tables connected via foreign keys. Approaches based on graphical models [10, 23], graph synopses [21], and join samples [2] have been proposed. Such solutions can avoid all three assumptions and thereby allow for accurate selectivity estimates. Our approach falls into this category. In fact, closest to our work are solutions based on PRMs [10, 23]. However, PRM-based approaches focus on relational data. We discussed in detail why PRMs are not directly applicable to graph-structured data. A key problem is that such approaches assume queries with selection and join predicates, which are evaluated against explicitly specified tables. Queries in our setting, however, may not specify tables from which data shall be selected. In general, the effectiveness of PRM systems is greatly determined by the chosen data partitioning scheme. Addressing these shortcomings, we rely on a different template-based representation of BNs, which is more suited for modeling probabilistic dependencies in graph-structured data. Further, no previous approach supports predicates having large domains of textual values. In fact, some authors pointed out that the the number of nominal values can be limited via clustering or, if possible, using feature hierarchies [10]. However, there is no work studying how clustering techniques may be integrated into a selectivity estimation framework, or how it may affect estimation effectiveness respectively efficiency. In this paper, we build upon string synopses, and show how they can be used in a template-based BN.

Another direction of related work is concerned with estimating the selectivity of string predicates [4, 12, 13, 15, 24]. Some approaches aim at substring respectively fuzzy string matching [4, 12, 15], while other systems target "extraction" operators, e.g., regular expression or dictionary-based operators [20, 24]. However, these works do not consider dependencies among multiple string predicates and/or query predicates evaluated against structured data. In this paper, we show that string synopses can be integrated into a template-based BN to deal with a combination of string and structured query predicates (hybrid queries). Overall, our approach represents a general schema-level synopsis capable of handling hybrid queries.

# 6. CONCLUSION

We tackled the problem of selectivity estimation for conjunctive queries, which may comprise structured query predicates as well as string predicates. In the graph-structured setting, where queries in the form of graph patterns are evaluated against data graphs, we showed that existing probabilistic approaches introduced for relational databases do not fit well. We propose a template-based model ($\text{BN}^+$), which is better suited for a graph setting. $\text{BN}^+$ allows dependencies between query predicates to be considered independent of the data partitioning design. Our model has the merit of being compact, capturing dependencies only at the level of templates, which are then instantiated to compute selectivity estimates for specific queries. In order to incorporate string predicates, we further propose the integration of string synopses into this model to compactly summarize textual values. We conducted experiments on real-world datasets, showing that, given there are dependencies between predicates and values, the accuracy of selectivity estimation can be greatly improved, when compared to a base-

line relying on the independence assumption. Our higher accuracy does not come at the expense of performance, as the inferencing needed to consider the dependencies required only negligible overhead. Room for further improvements lies in the string synopses, which have the biggest impact on the computational efficiency. Thus, we will study optimized string synopses for this problem as future work.

## 7. REFERENCES


[1] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. Scalable semantic web data management using vertical partitioning. In *VLDB*, pages 411–422, 2007.

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

[3] S. Chaudhuri, B.-C. Chen, V. Ganti, and R. Kaushik. Example-driven design of efficient record matching queries. In *VLDB*, pages 327–338, 2007.

[4] S. Chaudhuri, V. Ganti, and L. Gravano. Selectivity estimation for string predicates: Overcoming the underestimation problem. In *ICDE*, pages 227–238, 2004.

[5] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

[6] J. Coffman and A. C. Weaver. A framework for evaluating database keyword search strategies. In *CIKM*, pages 729–738, 2010.

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

[8] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *IJCAI*, pages 1300–1309, 1999.

[9] L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning*. MIT Press, 2007.

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

[11] H. Huang and C. Liu. Estimating selectivity for joined rdf triple patterns. In *CIKM*, pages 1435–1444, 2011.

[12] H. V. Jagadish, O. Kapitskaia, R. T. Ng, and D. Srivastava. One-dimensional and multi-dimensional substring selectivity estimation. *VLDB Journal*, 9(3):214–230, 2000.

[13] L. Jin and C. Li. Selectivity estimation for fuzzy string predicates in large data sets. In *VLDB*, pages 397–408, 2005.

[14] D. Koller and N. Friedman. *Probabilistic graphical models*. MIT press, 2009.

[15] H. Lee, R. T. Ng, and K. Shim. Extending q-grams to estimate selectivity of string matching with low edit distance. In *VLDB*, pages 195–206, 2007.

[16] Y. Luo, W. Wang, X. Lin, X. Zhou, J. Wang, and K. Li. Spark2: Top-k keyword query in relational databases. *IEEE Transactions on Knowledge and Data Engineering*, 23(12):1763–1780, 2011.

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

[18] M. Meila and M. Jordan. Learning with mixtures of trees. *The Journal of Machine Learning Research*, 1:1–48, 2001.

[19] V. Poosala, P. Haas, Y. Ioannidis, and E. Shekita. Improved histograms for selectivity estimation of range predicates. *SIGMOD*, 25(2):294–305, 1996.

[20] W. Shen, A. Doan, J. F. Naughton, and R. Ramakrishnan. Declarative information extraction using datalog with embedded extraction predicates. In *VLDB*, pages 1033–1044, 2007.

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

[22] B. Taskar, E. Segal, and D. Koller. Probabilistic classification and clustering in relational data. In *IJCAI*, pages 870–876, 2001.

[23] K. Tzoumas, A. Deshpande, and C. S. Jensen. Lightweight graphical models for selectivity estimation without independence assumptions. *PVLDB*, 4(11):852–863, 2011.

[24] D. Z. Wang, L. Wei, Y. Li, F. Reiss, and S. Vaithyanathan. Selectivity estimation for extraction operators over text data. In *ICDE*, pages 685–696, 2011.

[25] K. Wilkinson, C. Sayers, H. A. Kuno, and D. Reynolds. Efficient rdf storage and retrieval in jena2. In *SWDB*, pages 131–150, 2003.


## 8. APPENDIX

We present the query load used during our experiments. Note, queries for the DBLP dataset are based on [16], while IMDB queries are taken from [6]. All queries are given in RDF N3[6] notation.

**Listing 2: Queries for DBLP [16]**

```
# @prefix dc:
# http://purl.org/dc/elements/1.1/> .
# @prefix foaf:
# <http://xmlns.com/foaf/0.1/> .
# @prefix rdf:
# <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
# @prefix rdfs:
# <http://www.w3.org/2000/01/rdf-schema#> .
# @prefix dblp:
# <http://lsdis.cs.uga.edu/projects/semdis/opus#> .

# q1
?x rdfs:label "clique" .
?x dblp:last_modified_date "2002-12-09" .
?x rdf:type dblp:Article_in_Proceedings .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "nikos" .

# q2
?y rdf:type foaf:Person .
?y foaf:name "nikos" .
?y foaf:name "zotos" .

# q3
?x rdfs:label "constraint" .
?x dblp:last_modified_date "2005-02-25" .
?x rdf:type dblp:Article_in_Proceedings .
?x dblp:author ?y .
```

[6] http://www.w3.org/TeamSubmission/n3/

```
?y rdf:type foaf:Person .
?y foaf:name "chuang" .

# q4
?x rdfs:label "mining" .
?x rdfs:label "clustering" .
?x dblp:year "2005" .
?x rdf:type dblp:Article .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "nikos" .

# q5
?x rdfs:label "spatial" .
?x dblp:last_modified_date "2006-03-31" .
?x rdf:type dblp:Article_in_Proceedings .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "patel" .

# q6
?x rdf:type dblp:Article .
?x rdfs:label "middleware" .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "zhang" .

# q7
?x rdf:type dblp:Article_in_Proceedings .
?x rdfs:label "middleware" .
?x rdfs:label "optimal" .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "ronald" .

# q8
?x rdf:type dblp:Article_in_Proceedings .
?x rdfs:label "partition" .
?x rdfs:label "relational" .
?x rdfs:label "query" .

# q9
?x rdf:type dblp:Article_in_Proceedings .
?x rdfs:label "partition" .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "patel" .

# q10
?x rdf:type dblp:Proceedings .
?x rdfs:label "recognition" .
?x rdfs:label "speech" .
?x rdfs:label "software" .
?x dc:publisher ?p .

# q11
?x rdf:type dblp:Proceedings .
?x rdfs:label "data" .
?x rdfs:label "mining" .
?x dc:publisher <http://www.springer.de/> .

# q12
?x rdf:type dblp:Proceedings .
?x rdfs:label "australia" .
?x rdfs:label "stream" .
?x dc:publisher <http://www.springer.de/> .

# q13
?x dblp:year "2002" .
?x rdf:type dblp:Proceedings .
?x rdfs:label "industrial" .
?x rdfs:label "database" .
?x dc:publisher ?p .

# q14
?x rdf:type dblp:Article_in_Proceedings .
?x dblp:last_modified_date "2006-03-09" .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "jignesh" .

# q15
?x rdf:type dblp:Article_in_Proceedings .
?x rdfs:label "algorithm" .
?x rdfs:label "incomplete" .
?x rdfs:label "search" .

# q16
?x dblp:journal_name "SIGMOD" .
?x rdf:type dblp:Article .
?x rdfs:label "web" .
?x rdfs:label "search" .

# q17
?x rdf:type dblp:Article_in_Proceedings .
?x rdfs:label "semistructured" .
?x rdfs:label "search" .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "goldman" .

# q18
?x rdf:type dblp:Article_in_Proceedings .
?x rdfs:label "query" .
?x rdfs:label "cost" .
?x rdfs:label "optimization" .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "arvind" .

# q19
?x dblp:year "2007" .
?x rdfs:label "software" .
?x rdfs:label "time" .
?x rdf:type dblp:Article .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "zhu" .

# q20
?y rdf:type foaf:Person .
?y foaf:name "zhu" .
?y foaf:name "yuntao" .

# q21
?x dblp:year "2003" .
?x rdfs:label "data" .
?x rdfs:label "content" .
?x rdf:type dblp:Article_in_Proceedings .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "nikos" .

# q22
?x rdfs:label "spatial" .
?x rdf:type dblp:Article_in_Proceedings .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "jignesh" .

# q23
?x rdfs:label "algorithms" .
?x rdfs:label "parallel" .
?x rdfs:label "spatial" .
?x rdf:type dblp:Article_in_Proceedings .
?x dblp:author ?y .
?x dc:relation "conf" .
?y rdf:type foaf:Person .
?y foaf:name "patel" .

# q24
?x rdfs:label "implementation" .
```

```
?x rdfs:label "evaluation" .
?x rdf:type dblp:Article_in_Proceedings .
?x dblp:last_modified_date "2006-03-31" .
?x dblp:cites ?c .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "patel" .

# q25
?x rdfs:label "optimization" .
?x rdfs:label "query" .
?x rdf:type dblp:Article_in_Proceedings .
?x dblp:author ?y .
?x dblp:year "2003" .
?y rdf:type foaf:Person .
?y foaf:name ?n .

# q26
?x rdfs:label "xml" .
?x rdfs:label "tool" .
?x rdf:type dblp:Article_in_Proceedings .
?x dblp:year "2004" .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "patel" .

# q27
?x rdf:type dblp:Article_in_Proceedings .
?x rdfs:label "architecture" .
?x rdfs:label "web" .
?x dblp:last_modified_date "2005-09-05" .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "wu" .

# q28
?x rdf:type dblp:Article_in_Proceedings .
?x rdfs:label "language" .
?x rdfs:label "software" .
?x rdfs:label "system" .
?x dblp:year "2001" .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "roland" .

# q29
?x rdf:type dblp:Article_in_Proceedings .
?x rdfs:label "middleware" .
?x dblp:last_modified_date "2006-01-17" .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "sihvonen" .

# q30
?x rdf:type dblp:Article_in_Proceedings .
?x rdfs:label "middleware" .
?x rdfs:label "virtual" .
?x dblp:year "2001" .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "kwang" .

# q31
?x rdf:type dblp:Article .
?x rdfs:label "java" .
?x rdfs:label "code" .
?x rdfs:label "program" .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "roland" .

# q32
?x rdf:type dblp:Article .
?x rdfs:label "signal" .
?x rdfs:label "space" .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "zheng" .

# q33
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "fagin" .
?y foaf:name "roland" .

# q34
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "zheng" .
?y foaf:name "qui" .

# q35
?x rdf:type dblp:Article_in_Proceedings .
?x rdfs:label "processing" .
?x rdfs:label "query" .

# q36
?x rdf:type dblp:Article_in_Proceedings .
?x rdfs:label "xml" .
?x rdfs:label "processing" .

# q37
?x rdf:type dblp:Article_in_Proceedings .
?x rdfs:label "biological" .
?x rdfs:label "sequence" .
?x dblp:last_modified_date "2007-08-21" .
?x dblp:author ?y .
?y rdf:type foaf:Person .
?y foaf:name "jignesh" .

# q38
?x rdf:type dblp:Book .
?x rdfs:label "decision" .
?x rdfs:label "intelligent" .
?x rdfs:label "making" .
?x dc:publisher <http://www.springer.de/> .

# q39
?x rdf:type dblp:Proceedings .
?x rdfs:label "databases" .
?x rdfs:label "biological" .
?x dc:publisher <http://www.springer.de/> .

# q40
?x rdf:type dblp:Book .
?x rdfs:label "mining" .
?x rdfs:label "data" .

# q41
?x rdf:type dblp:Book .
?x rdfs:label "mining" .
?x rdfs:label "data" .
?x dc:publisher <http://www.springer.de/> .
?x dc:relation "trier.de" .
?x dc:relation "books" .

# q42
?x rdf:type dblp:Book .
?x rdfs:label "intelligence" .
?x rdfs:label "computational" .
?x dc:publisher <http://www.springer.de/> .
?x dc:relation "trier.de" .
?x dblp:year "2007" .

# q43
?x rdf:type dblp:Book .
?x rdfs:label "biologically" .
?x rdfs:label "inspired" .
?x rdfs:label "methods" .

# q44
?x rdf:type dblp:Book .
```

```
?x rdfs:label "networks" .
?x rdfs:label "neural" .

# q45
?x rdf:type dblp:Book .
?x rdfs:label "learning" .
?x rdfs:label "machine" .
?x dc:publisher <http://www.springer.de/> .

# q46
?x rdf:type dblp:Book .
?x rdfs:label "software" .
?x rdfs:label "system" .
?x dc:publisher <http://www.springer.de/> .

# q47
?x rdf:type dblp:Book .
?x rdfs:label "architecture" .
?x rdfs:label "computer" .

# q48
?x rdf:type dblp:Book .
?x rdfs:label "web" .
?x dblp:year "2006" .
?x dc:publisher ?p .
?x dblp:editor ?e .
?e foaf:name "kandel" .
?e foaf:name "abraham" .

# q49
?x rdf:type dblp:Book .
?x rdfs:label "theoretical" .
?x rdfs:label "science" .
?x dc:publisher <http://www.elsevier.nl/> .

# q50
?x rdf:type dblp:Book_Chapter .
?x rdfs:label "search" .
?x rdfs:label "semantic" .

# q51
?x rdf:type dblp:Article .
?x rdfs:label "search" .
?x rdfs:label "concept" .
?x rdfs:label "based" .

# q52
?x dblp:journal_name "sigmod" .
?x rdf:type dblp:Article .
?x rdfs:label "model" .
?x rdfs:label "information" .

# q53
?x dblp:journal_name "sigmod" .
?x rdf:type dblp:Article .
?x rdfs:label "dynamic" .
?x rdfs:label "networks" .

# q54
?x rdf:type dblp:Article_in_Proceedings .
?x rdfs:label "storage" .
?x rdfs:label "adaptive" .
?x dblp:author ?y .
?x dblp:year "2003" .
?y rdf:type foaf:Person .
?y foaf:name "jignesh" .
```

**Listing 3: Queries for IMDB [6]**

```
# @prefix imdb:
# <http://imdb/predicate/> .
# @prefix imdb_class:
# <http://imdb/class/> .
# @prefix rdf:
# <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

# q1
?x rdf:type imdb_class:name .
?x imdb:name "washington" .
?x imdb:name "denzel" .

# q2
?x rdf:type imdb_class:name .
?x imdb:name "eastwood" .
?x imdb:name "clint" .

# q3
?x rdf:type imdb_class:name .
?x imdb:name "john" .
?x imdb:name "wayne" .

# q4
?x rdf:type imdb_class:name .
?x imdb:name "smith" .
?x imdb:name "will" .

# q5
?x rdf:type imdb_class:name .
?x imdb:name "ford" .
?x imdb:name "harrison" .

# q6
?x rdf:type imdb_class:name .
?x imdb:name "julia" .
?x imdb:name "roberts" .

# q7
?x rdf:type imdb_class:name .
?x imdb:name "tom" .
?x imdb:name "hanks" .

# q8
?x rdf:type imdb_class:name .
?x imdb:name "johnny" .
?x imdb:name "depp" .

# q9
?x rdf:type imdb_class:name .
?x imdb:name "angelina" .
?x imdb:name "jolie" .

# q10
?x rdf:type imdb_class:name .
?x imdb:name "freeman" .
?x imdb:name "morgan" .

# q11
?x rdf:type imdb_class:title .
?x imdb:title "gone" .
?x imdb:title "with" .
?x imdb:title "the" .
?x imdb:title "wind" .

# q12
?x rdf:type imdb_class:title .
?x imdb:title "wars" .
?x imdb:title "star" .

# q13
?x rdf:type imdb_class:title .
?x imdb:title "casablanca" .

# q14
?x rdf:type imdb_class:title .
?x imdb:title "the" .
?x imdb:title "lord" .
?x imdb:title "rings" .

# q15
?x rdf:type imdb_class:title .
```

```
?x imdb:title "the" .
?x imdb:title "sound" .
?x imdb:title "music" .

# q16
?x rdf:type imdb_class:title .
?x imdb:title "wizard" .
?x imdb:title "oz" .

# q17
?x rdf:type imdb_class:title .
?x imdb:title "the" .
?x imdb:title "notebook" .

# q18
?x rdf:type imdb_class:title .
?x imdb:title "forrest" .
?x imdb:title "gump" .

# q19
?x rdf:type imdb_class:title .
?x imdb:title "the" .
?x imdb:title "princess" .
?x imdb:title "bride" .

# q20
?x rdf:type imdb_class:title .
?x imdb:title "the" .
?x imdb:title "godfather" .

# q21
?x imdb:title ?t .
?x rdf:type imdb_class:title .
?x imdb:cast_info ?z .
?r rdf:type imdb_class:char_name .
?r imdb:name "finch" .
?r imdb:name "atticus" .
?z rdf:type imdb_class:cast_info .
?z imdb:role ?r .

# q22
?x imdb:title ?t .
?x rdf:type imdb_class:title .
?x imdb:cast_info ?z .
?z rdf:type imdb_class:cast_info .
?r imdb:name "indiana" .
?r imdb:name "jones" .
?z imdb:role ?r .
?r rdf:type imdb_class:char_name .

# q23
?x imdb:title ?t .
?x rdf:type imdb_class:title .
?x imdb:cast_info ?z .
?z rdf:type imdb_class:cast_info .
?z imdb:role ?r .
?r rdf:type imdb_class:char_name .
?r imdb:name "james" .
?r imdb:name "bond" .

# q24
?x imdb:title ?t .
?x rdf:type imdb_class:title .
?x imdb:cast_info ?z .
?z rdf:type imdb_class:cast_info .
?z imdb:role ?r .
?r rdf:type imdb_class:char_name .
?r imdb:name "rick" .
?r imdb:name "blaine" .

# q25
?x imdb:title ?t .
?x imdb:cast_info ?z .
?z rdf:type imdb_class:cast_info .
?z imdb:role ?r .
?r rdf:type imdb_class:char_name .
?r imdb:name "kaine" .
?r imdb:name "will" .

# q26
?x imdb:title ?t .
?x rdf:type imdb_class:title .
?x imdb:cast_info ?z .
?z rdf:type imdb_class:cast_info .
?z imdb:role ?r .
?r rdf:type imdb_class:char_name .
?r imdb:name "dr." .
?r imdb:name "hannibal" .
?r imdb:name "lecter" .

# q27
?x imdb:title ?t .
?x rdf:type imdb_class:title .
?x imdb:cast_info ?z .
?z rdf:type imdb_class:cast_info .
?z imdb:role ?r .
?r rdf:type imdb_class:char_name .
?r imdb:name "norman" .
?r imdb:name "bates" .

# q28
?x imdb:title ?t .
?x rdf:type imdb_class:title .
?x imdb:cast_info ?z .
?z rdf:type imdb_class:cast_info .
?z imdb:role ?r .
?r rdf:type imdb_class:char_name .
?r imdb:name "darth" .
?r imdb:name "vader" .

# q29
?x imdb:title ?t .
?x rdf:type imdb_class:title .
?x imdb:cast_info ?z .
?z rdf:type imdb_class:cast_info .
?z imdb:role ?r .
?r rdf:type imdb_class:char_name .
?r imdb:name "the␣" .
?r imdb:name "wicked" .
?r imdb:name "witch" .
?r imdb:name "the" .
?r imdb:name "west" .

# q30
?x imdb:title ?t .
?x rdf:type imdb_class:title .
?x imdb:cast_info ?z .
?z rdf:type imdb_class:cast_info .
?z imdb:role ?r .
?r rdf:type imdb_class:char_name .
?r imdb:name "nurse" .
?r imdb:name "ratched" .

# q31
?x imdb:title ?t .
?x rdf:type imdb_class:title .
?x imdb:movie_info ?i .
?i rdf:type imdb_class:movie_info .
?i imdb:info "frankly" .
?i imdb:info "dear" .
?i imdb:info "don't" .
?i imdb:info "give" .
?i imdb:info "damn" .

# q32
?x imdb:title ?t .
?x rdf:type imdb_class:title .
?x imdb:movie_info ?i .
?i rdf:type imdb_class:movie_info .
?i imdb:info "going" .
?i imdb:info "make" .
?i imdb:info "offer" .
```

```
?i imdb:info "can't" .
?i imdb:info "refuse" .

# q33
?x imdb:title ?t .
?x rdf:type imdb_class:title .
?x imdb:movie_info ?i .
?i rdf:type imdb_class:movie_info .
?i imdb:info "understand" .
?i imdb:info "class" .
?i imdb:info "contender" .
?i imdb:info "coulda" .
?i imdb:info "somebody" .
?i imdb:info "instead" .
?i imdb:info "bum" .

# q34
?x imdb:title ?t .
?x rdf:type imdb_class:title .
?x imdb:movie_info ?i .
?i rdf:type imdb_class:movie_info .
?i imdb:info "toto" .
?i imdb:info "feeling" .
?i imdb:info "not" .
?i imdb:info "kansas" .
?i imdb:info "anymore" .

# q35
?x imdb:title ?t .
?x rdf:type imdb_class:title .
?x imdb:movie_info ?i .
?i rdf:type imdb_class:movie_info .
?i imdb:info "here's" .
?i imdb:info "looking" .
?i imdb:info "kid" .

# q36
?x rdf:type imdb_class:title .
?c rdf:type imdb_class:cast_info .
?x imdb:cast_info ?c .
?c imdb:role ?r .
?r rdf:type imdb_class:char_name .
?r imdb:name "skywalker" .
?c imdb:person ?p .
?p rdf:type imdb_class:name .
?p imdb:name "hamill" .

# q37
?x imdb:year "2004" .
?x rdf:type imdb_class:title .
?x imdb:title ?t .
?x imdb:cast_info ?c .
?c rdf:type imdb_class:cast_info .
?c imdb:person ?p .
?p rdf:type imdb_class:name .
?p imdb:name "hanks" .

# q38 #
?r imdb:name ?rn .
?r rdf:type imdb_class:char_name .
?x rdf:type imdb_class:title .
?x imdb:title "yours" .
?x imdb:title "mine" .
?x imdb:title "ours" .
?x imdb:cast_info ?c .
?c rdf:type imdb_class:cast_info .
?c imdb:role ?r .
?c imdb:person ?p .
?p rdf:type imdb_class:name .
?p imdb:name "henry" .
?p imdb:name "fonda" .

# q39
?x rdf:type imdb_class:title .
?x imdb:title "gladiator" .
?x imdb:cast_info ?c .
?c rdf:type imdb_class:cast_info .
?c imdb:role ?r .
?r imdb:name ?rn .
?r rdf:type imdb_class:char_name .
?c imdb:person ?p .
?p rdf:type imdb_class:name .
?p imdb:name "russell" .
?p imdb:name "crowe" .

# q40
?x rdf:type imdb_class:title .
?x imdb:title "star" .
?x imdb:title "trek" .
?x imdb:cast_info ?c .
?r rdf:type imdb_class:char_name .
?r imdb:name ?rn .
?c rdf:type imdb_class:cast_info .
?c imdb:role ?r .
?c imdb:person ?p .
?p rdf:type imdb_class:name .
?p imdb:name "spiner" .
?p imdb:name "brent" .

# q41
?x imdb:year "1951" .
?x imdb:title ?t .
?x rdf:type imdb_class:title .
?x imdb:cast_info ?c .
?c rdf:type imdb_class:cast_info .
?c imdb:person ?p .
?p rdf:type imdb_class:name .
?p imdb:name "audrey" .
?p imdb:name "hepburn" .

# q42
?p rdf:type imdb_class:name .
?p imdb:name ?n .
?c imdb:person ?p .
?c rdf:type imdb_class:cast_info .
?c imdb:role ?r .
?r rdf:type imdb_class:char_name .
?r imdb:name "jacques" .
?r imdb:name "clouseau" .

# q43
?p rdf:type imdb_class:name .
?p imdb:name ?n .
?c imdb:person ?p .
?c rdf:type imdb_class:cast_info .
?c imdb:role ?r .
?r rdf:type imdb_class:char_name .
?r imdb:name "jack" .
?r imdb:name "ryan" .

# q44
?p rdf:type imdb_class:name .
?p imdb:name "stallone" .
?c imdb:person ?p .
?c rdf:type imdb_class:cast_info .
?c imdb:role ?r .
?r rdf:type imdb_class:char_name .
?r imdb:name "rocky" .

# q45
?p rdf:type imdb_class:name .
?p imdb:name ?n .
?c imdb:person ?p .
?c rdf:type imdb_class:cast_info .
?c imdb:role ?r .
?r rdf:type imdb_class:char_name .
?r imdb:name "terminator" .

# omitted q46 to q49

# q50
?a rdf:type imdb_class:title .
```

```
?a imdb:title "lost" .
?a imdb:title "ark" .
?a imdb:cast_info ?ca .
?ca rdf:type imdb_class:cast_info .
?ca imdb:person ?p .
?p rdf:type imdb_class:name .
?p imdb:name ?n .
?ci rdf:type imdb_class:cast_info .
?ci imdb:person ?p .
?i rdf:type imdb_class:title .
?i imdb:cast_info ?ci .
?i imdb:title "indiana" .
?i imdb:title "jones" .
?i imdb:title "last" .
?i imdb:title "crusade" .
```