Estimating Selectivity for Joined RDF Triple Patterns

Hai Huang
Faculty of ICT
Swinburne University of Technology
VIC 3122, Australia
hhuang@swin.edu.au

Chengfei Liu
Faculty of ICT
Swinburne University of Technology
VIC 3122, Australia
cliu@swin.edu.au

ABSTRACT

A fundamental problem related to RDF query processing is selectivity estimation, which is crucial to query optimization. In this paper, we focus on selectivity estimation for SPARQL graph patterns. The previous work takes the join uniformity assumption when estimating the joined triple patterns. This assumption would lead to highly inaccurate estimations in the cases where properties in SPARQL graph patterns are correlated. We take into account the dependencies among properties in SPARQL graph patterns and propose a more accurate estimation model. Since star and chain query patterns are common in SPARQL graph patterns, we first focus on these two basic patterns and propose to use Bayesian networks and chain histograms respectively for estimating the selectivity of them. Then, for estimating the selectivity of an arbitrary SPARQL graph pattern, we design algorithms for maximally using the precomputed statistics of the star paths and chain paths. The experiments show that our method outperforms existing approaches in accuracy.

Categories and Subject Descriptors
H.2.4 [Systems]: Query processing

General Terms
Performance

Keywords
SPARQL Query Processing, Selectivity Estimation

1. INTRODUCTION

The Resource Description Framework (RDF) is a standard format for encoding machine-readable information on the Semantic Web. Recently, more and more data is being stored in RDF format. RDF data is a set of triples and each triple called statement is of the form (subject, property, object). RDF data can be represented as a graph with nodes representing resources or their property values and labeled arcs representing properties. This data representation is general and flexible. However, this fine-grained model leads to queries on RDF data with a large number of joins, which is an inherent characteristic of querying RDF data [1].

Since the use of RDF to represent data has grown dramatically over the last few years, query processing on RDF data becomes an important issue in realizing the semantic web vision. Some query languages such as SPARQL have been developed. As we know, accurate estimation of the result size of queries is crucial to query processing. Cost-based query optimizers use estimated intermediate result size to choose the optimal query execution plan.

As a SPARQL query has a large number of joins, estimating precisely the joined triple pattern is very important. Some work has been done in this area. In [4, 9] the join uniformity assumption is made when estimating the joined triple patterns with bounded subjects or objects (i.e., the subjects or objects are concrete values). They assume that each triple satisfying a triple pattern is equally likely to join with the triples satisfying the other triple pattern. However, this assumption does not hold in many cases. And when the data are inconsistent with this assumption, it could cause a highly inaccurate estimation.

For example, in Figure 1, a SPARQL query is posed on an RDF database, which retrieves academic staff members and the courses they teach with some conditions. Suppose we want to estimate the selectivity of the joined triple pattern with the formula (1) proposed in [4] as follows:

$$\text{sel}(t_1 \bowtie t_3) = \frac{S_P}{|T|} \times \text{sel}(\text{Income}, ' \leq 70K') \times \text{sel}(\text{Position}, '\text{Prof.}$$

where $S_P$ is the result upper bound of the joined triple pattern $(?Z, \text{Income}, ?W)(?Z, \text{Position}, ?Y)$ and $|T|$ is the number of triples in the database. sel(\text{Income}, ' \leq 70K') and sel(\text{Position}, '\text{Prof.}$$) are the object selectivities of $t_1$ and $t_3$. It assumes that each triple satisfying triple pattern $t_1$ is equally likely to join with triples satisfying triple pattern $t_3$. But in fact, the triple matching $t_3$ which indicates that the person is a professor, who is supposed to have higher income ($> 70K$). Thus, the triples matching $t_1$ are less likely to join with triples matching $t_3$.

There are two observations from this example. First, we...
can observe that the join uniformity assumption is not applicable to those cases where properties have correlations (dependencies) with each other. In fact, this assumption is rarely satisfied in real data, so we need a more accurate model to relax this assumption for estimating the result size of a RDF triple. Based on the observations, we propose new methods for selectivity estimation of a set of literals. In this paper, we construct Bayesian networks to compactly represent the joint probability distribution over values of correlated properties. And for chain query patterns, we build the chain histogram, which can obtain a good balance between the estimation accuracy and space cost.

- For an arbitrary SPARQL query represented as a composite graph pattern, we propose algorithms for maximally combining the statistics of chain paths and star paths that we have precomputed to estimate the overall selectivity of the graph pattern.

- We conduct experiments on both synthetic and real-world datasets show the effectiveness and performance of our approach.

The remainder of this paper is organized as follows. Section 2 introduces some preliminary knowledge. In Section 3 and Section 4, we propose two methods to estimate the selectivity of star and chain query patterns, respectively. Section 5 presents the algorithms for estimating the selectivity of arbitrary composite SPARQL graph patterns. Section 6 describes an experimental evaluation of our approach. Some related work is discussed in Section 7. At last, in Section 8 we conclude our work in this paper.

2. PRELIMINARY

A triple \((s, p, o)\) \(\in (I \cup B) \times (I \cup B) \times (I \cup B \cup L)\) is called an RDF triple, where \(I\) is a set of IRIls (Internationalized URIs), \(B\) a set of blank nodes and \(L\) a set of literals. In the triple, \(s\) is called subject, \(p\) the property (or predicate), and \(o\) the object or property value. An RDF triple pattern \((s, p, o)\) \(\in (I \cup V) \times (I \cup V) \times (I \cup V \cup L)\), where \(V\) is a set of variables disjoint from the sets \(I, B\) and \(L\). An RDF graph pattern \(G = (q_1, q_2, \ldots, q_n), q_i \in T\), where \(T\) is a set of triple patterns. A SPARQL query can be represented as an RDF graph pattern and called a SPARQL graph pattern.

Problem Definition. The problem that we tackle in this paper can be summarized as follows: Given an RDF database \(D\) and a SPARQL graph pattern \(Q\), we estimate the selectivity \(sel(Q)\) of \(Q\). \(sel(Q)\) stands for the count of results in database \(D\) satisfying \(Q\). In this paper, our interest is to estimate the selectivity of joined triple patterns, which will be discussed in the following sections.

3. ESTIMATION FOR STAR QUERY PATTERNS

In this section, we present the method of using Bayesian networks to estimate the selectivity of star query patterns.

3.1 Estimation for star patterns using Bayesian networks

Star query pattern that commonly occurs in SPARQL queries is used to retrieve entities with some constraints. It has the form of a number of triple patterns with different properties sharing the same subject variable. Normally, the properties involved in a star query patterns are correlated. For example, Figure 2 shows a star query pattern \(Q\) and the properties \(Income, Education, Position, TeacherOf\) are correlated. Thus, as shown in the introduction, the join uniformity assumption would cause a highly inaccurate selectivity estimation for star query patterns.

For estimating the selectivities of star query patterns precisely, we need some helpful statistics. In this paper, we precompute statistics for star paths. A star path is a set of properties which share the same domain. Given a star path with properties \(prop_1, prop_2, \ldots, prop_n\), we use \((prop_1, prop_2, \ldots, prop_n)\) to denote. In Figure 2, \(Q\) \((Income, Education, Position, TeacherOf)\) is a star path. Since the number of possible star paths could be huge \(2^n\), where \(n\) is the number of properties in the database, it is impossible to precompute statistics for all star paths. In this paper, we target the star paths in which properties are strongly correlated.

We assume that if a star path appears in the data graph frequently then the properties contained in this star path are strongly correlated. This assumption is also made in data mining field to find association rules. Using the algorithm presented in [9], we can find frequent star paths. A frequent star path is logically associated with a table \(R\) called cluster-property table [2] that contains a set of entities to be queried. For example, Figure 2 shows a cluster-property table of star path \(Q\). Each row of the table is an academic staff with values of properties \(Education, Income, Position\) and \(TeacherOf\). Note that different with [2], we do not store cluster-property tables for frequent star paths. More compressed structures would be constructed for selectivity estimation.

Given a frequent star path \(Q\) \((prop_1, prop_2, \ldots, prop_n)\) and its cluster-property table \(R\), the selectivity of star query \(Q\) \((?x, prop_1, o_1) \&(?x, prop_2, o_2) \&\cdots \&(?x, prop_n, o_n)\) denoted
by \(sel(Q)\), can be computed as follows:

\[
sel(Q) = \Pr(prop_1 = o_1, \ldots, prop_m = o_m) \cdot |R|
\]

where \(\Pr(prop_1 = o_1, \ldots, prop_m = o_m)\) denotes the joint probability distribution over object values of properties \(prop_1, \ldots, prop_m\) in table \(R\) and \(|R|\) is the number of rows in the cluster-property table \(R\).

Since the number of rows \(|R|\) is easy to know, we will focus on computing the joint probability distribution \(\Pr(prop_1 = o_1, \ldots, prop_m = o_m)\). Unfortunately, it is impossible to explicitly store \(\Pr(prop_1 = o_1, \ldots, prop_m = o_m)\) because the possible combinations of property values would be exponential. Thus, we need an appropriate structure to approximately store the joint probability distribution information.

Figure 2: Star path and star query pattern

The Bayesian network [3] can approximately represent the probability distribution over a set of variables using a little space. Bayesian networks make use of Bayes’ Rule and conditional independence assumption to compactly represent the full joint probability distribution. Let \(X, Y, Z\) be three discrete valued random variables. We say that \(X\) is conditionally independent of \(Y\) given \(Z\) if the probability distribution of \(X\) is independent of the value of \(Y\) given a value for \(Z\); that is:

\[
\Pr(X = x_i | Y = y_j, Z = z_k) = \Pr(X = x_i | Z = z_k)
\]

where \(x_i, y_j, z_k\) are values of variables \(X, Y, Z\). The conditional independence assumptions associated with a Bayesian network and conditional probability tables (CPTs), determine a joint probability distribution. For example, in Figure 3, a Bayesian network is constructed on cluster-property table in Figure 2. We can see that properties Education and Income are conditionally independent given condition Position, which means if we already know the position of some person, knowing his education information does not make any difference to our beliefs about his income.

For a star query pattern \(Q\) with properties and object values \(prop_1 = o_1, \ldots, prop_m = o_m\), given a Bayesian network \(\beta\), we have:

\[
Pr_{\beta}(prop_1 = o_1, \ldots, prop_m = o_m) = \prod_{i=1}^{m} \Pr(prop_i = o_i | Parents(prop_i) = o_i) \cdot |R|
\]

where \(Parents(prop_i)\) denotes the set of immediate predecessors of \(prop_i\) in the network and \(o_i\) denotes the set of values of \(Parents(prop_i)\). Note that for computing \(\Pr(prop_i | Parents(prop_i) = o_i)\), we only need to know the values of \(prop_i\)'s parent properties, which would save a lot of space in practice. So given the Bayesian network \(\beta\), we can use \(Pr_{\beta}(prop_1 = o_1, \ldots, prop_m = o_m)\) to approximately represent \(Pr_{\beta}(prop_1 = o_1, \ldots, prop_m = o_m)\).

We have:

\[
\Pr(prop_1 = o_1, \ldots, prop_m = o_m) \cdot |R| = \prod_{i=1}^{m} \Pr(prop_i = o_i | Parents(prop_i) = o_i) \cdot |R|
\]

**Example 1.** Given the star pattern \(Q(\exists \, Z, income \leq 70K)\) (\(\exists \, Z, education \equiv \text{'PhD'}\) (\(\exists \, Z, position \equiv \text{'Prof.'}\) (\(\exists Z, TeacherOf, \text{ ‘Adc’}\)) shown in Figure 2 and Bayesian network described in Figure 3, we compute the selectivity of \(Q\) as follows:

\[
\Pr(Edu = \text{‘PhD’}, Pos = \text{‘Prof.’}, Inc \leq 70K', TOf = \text{‘Adc’}) \cdot |R|
\]

\[
\Pr(Edu = \text{‘PhD’}) \cdot \Pr(Inc \leq 70K | Pos = \text{‘Prof.’}) \cdot \Pr(TOf = \text{‘Adc’} | Pos = \text{‘Prof.’}) \cdot \Pr(Edu = \text{‘PhD’}) \cdot |R|
\]

\[
= 0.4 \cdot 0.01 \cdot 0.7 \cdot 0.3 \cdot |R| = 0.00084 \cdot |R|
\]

**3.2 Learning Bayesian networks**

To approximately represent the joint distribution of property values for selectivity estimation, we first generate the cluster-property tables for frequent star paths, and then Bayesian networks can be learned from these tables. From the Aprior property, we know that if a star path is frequent then its sub paths are also frequent. Given a frequent star path \((p_1, p_2, p_3)\) and one of its sub path \((p_1, p_3)\), if they have equal frequencies, namely \(|(p_1, p_2, p_3)| = |(p_1, p_3)|\), it indicates that they have the same instance set. In this case, the information contained in the cluster-property table of path \((p_1, p_3)\) is fully covered by the cluster-property table of path \((p_1, p_2, p_3)\) and we do not need to generate the cluster-property table for path \((p_1, p_3)\). Note that we will drop all cluster-property tables once Bayesian networks have been constructed.

Before building Bayesian networks, the property values
are first clustered into equi-width subsets (abstracted values). Bayesian network construction algorithms can be grouped into two categories: one category of algorithms uses heuristic searching methods to construct a model and then evaluates it using a scoring method [5, 6, 8]. The other category of algorithms constructs Bayesian networks by analyzing dependency relationships among nodes. The dependency relationships are measured by using some kind of conditional independence (CI) test. In this paper we adopt the algorithm in [7], which employs mutual information to measure how close the relationship between two variables. The mutual information \( I(X, Y) \) of two variables \( X, Y \) is defined as follows:

\[
I(X, Y) = \sum_{x,y} \Pr(x, y) \log \frac{\Pr(x, y)}{\Pr(x) \cdot \Pr(y)}
\]

and the conditional mutual information is defined as

\[
I(X, Y|M) = \sum_{x,y,m} \Pr(x, y, m) \log \frac{\Pr(x, y|m)}{\Pr(x|m) \cdot \Pr(y|m)}
\]

where \( M \) is a set of observed variables. When \( I(X, Y|M) \) is smaller than a threshold \( \epsilon \), \( X \) and \( Y \) are conditionally independent and we also call \( X \) and \( Y \) are \( d \)-separated by the condition set \( M \).

The construction algorithm contains three steps. The first step is to compute mutual information of each pair of nodes as a measure of closeness, and creates a draft graph based on this information. The draft graph is a singly connected graph (a graph without loops). In the second phase, the algorithm adds edges when the pairs of nodes cannot be \( d \)-separated. In the third step, each edge of the Independent-map is examined using CI tests and will be removed if the two nodes of the edge can be \( d \)-separated. At the end of the third phase, the algorithm also carries out a procedure to orient the edges of the graph. For details on specific three steps, please see [7]. The time complexity of this algorithm is \( O(n^3) \), where \( n \) is the number of properties involved in the cluster property table. In our problem, it requires that the conditional probability tables learned fit in a small amount of memory, therefore we restrict the threshold \( \epsilon \) for space limit when building Bayesian networks.

4. ESTIMATION FOR CHAIN PATTERNS

The other kind of correlated properties commonly occur in SPARQL graph patterns is chain path. A chain path consists of a sequence of triple patterns where the object of a triple pattern is also the subject of the next triple pattern. For example, in Figure 4, \((\text{\texttt{?course}}, \text{\texttt{TakenBy}}, \text{\texttt{?student}})(\text{\texttt{?student}}, \text{\texttt{Age}, ?\texttt{age}})\) is a chain path. Note that triple patterns of a chain path have only unbound subjects and objects. Given a chain path with a sequence of properties \( p_1, \ldots, p_n \), we use \((p_1, p_2, \ldots, p_n)\) to denote it. For instance, in Figure 4, \((\text{\texttt{TakenBy-Age}})\) is a chain path and tuples in the data graph matching this chain path fall in the subject-property matrix table [10] of this chain path.

A chain query pattern is a chain path with possible literal constraints on the start node (the subject of the first triple pattern) or end node (the object of the last triple pattern) in the chain path. For example, \((\text{\texttt{?course}}, \text{\texttt{TakenBy}}, \text{\texttt{?student}})(\text{\texttt{?student}}, \text{\texttt{Age}, '22'})\) shown in Figure 4 is a chain query pattern. The straightforward way to estimate the selectivity of a chain query pattern is combining the selectivity of each triple pattern in the product form. Obviously, it would be highly inaccurate if the properties are correlated.

4.1 Constructing the histogram

We first select frequent chain paths and construct the histogram for them. For these frequent chain paths, we can construct a chain frequency table \(\text{CF T}\) (shown in Figure 5), which has two attributes: Abstracted chain query pattern and Frequency. The values of attribute Abstracted chain query pattern are abstracted chain query pattern descriptors consisting of frequent chain paths with possible value combinations of the start node and end node. Note that the possible values occurring on the start node and end node of paths are clustered into equi-width subsets called abstracted values. Each value of attribute Frequency is the count of tuples that match the abstracted chain pattern query in the data graph. We can see that each row of \(\text{CF T}\) indicates an abstracted chain query pattern with its frequency, so it is easy to get the selectivity of abstracted chain patterns from \(\text{CF T}\). For example, the first tuple \((\text{\texttt{Pre/p1/20-22}}, 26)\) of chain frequency table shown in Figure 5 indicates a chain query pattern \((\text{\texttt{Pre}}, \text{\texttt{TakenBy}}, \text{\texttt{?student}})(\text{\texttt{?student}}, \text{\texttt{Age, '20-22'}})\) with its frequency 26. However, chain frequency table \(\text{CF T}\) could be too large to fit in a small amount of main memory. Thus, we construct the new data structure called chain histogram to approximate the selectivity of chain query patterns.

![Figure 4: Chain path and chain query pattern](image1)

![Figure 5: An example of the chain frequency table and the chain histogram. “?” indicates a variable.](image2)
age frequency and its abstracted chain query pattern members. Given an abstracted chain query pattern and which bucket it belongs to, we can easily get the approximate frequency of this chain query pattern.

To space-efficiently store the elements of each bucket and accelerate the membership query processing, we employ the bloom filter technique. Bloom filter [11, 12] is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set. Here we use bloom filters to store buckets and construct the chain histogram. An example of chain histogram is shown in Figure 5.

4.2 Estimation for chain patterns using the histogram

Given a chain query pattern \( C \) and a histogram \( H \), for estimating the selectivity of \( C \), we first map the basic level values of start node and end node of \( C \) to their abstracted values and generate the abstracted chain query pattern \( C_a \). For example, if a chain query pattern \( C(\text{?course, Takenby, ?student}) \text{(?student, Age, ’22’)} \) is posed on the database, we map the value ’22’ to its abstracted value ’20-22’ and get the abstracted chain query pattern \( C_a(\text{?y, Takenby, ?student}) \text{(?student, Age, ’20-22’)} \). The uniformity assumption is made here when computing the coefficient \( \eta \) between the selectivities of \( C \) and \( C_a \). We can acquire the approximate selectivity of chain query pattern \( C_a \) from the chain histogram \( H \). Note that though it is rare, it is possible that the pattern \( C_a \) is reported to fall in multiple buckets due to the false positive error of the bloom filter. In this case, we simply return the count of the first bucket which reports that the pattern \( C_a \) is its member. This process is given in Algorithm 1 GetChainSel. The time complexity of obtaining the selectivity of a chain pattern is \( O(KB) \), where \( K \) is the maximal number of hash functions used in a bloom filter and \( B \) is the number of buckets (or bloom filters) in the chain histogram.

Algorithm 1 GetChainSel

**Input:** Chain pattern \( C \), histogram \( H \);  
**Output:** The selectivity set of chain pattern \( C \);

1: Map chain query pattern \( C \) to \( C_a \); \( \{C_a \text{ is abstracted chain pattern of } C\} \)
2: Compute the coefficient \( \eta \) between \( C \) and \( C_a \);
3: for all buckets \( b_j \) in \( H \) do
4: if \( C_a \) falls into a certain bucket \( b_j \) then
5: \( \text{sel}_a = b_j \times \text{frequency}; \{\text{sel}_a \text{ is the selectivity of } C_a\} \)
6: return \( \text{sel} = \eta \times \text{sel}_a; \)
7: return 0.

For example, given a chain query pattern \( (?y, \text{Takenby, ?student}) \text{(?student, Age, ’22’)} \), we compute the selectivity of this chain query pattern as follows:

\[
\text{sel} = \frac{1}{3} \times \text{sel}(?/p1/20-22) = \frac{1}{3} \times 74 \approx 25;
\]

5. ESTIMATION FOR COMPOSITE GRAPH PATTERNS

In this section, we discuss how to estimate the selectivity of an arbitrary composite graph pattern with the maximum use of the precomputed statistics of star and chain paths.

5.1 Decomposition of SPARQL graph patterns

From previous sections, we know that the precomputed statistics of the star and chain paths can help to estimate the selectivity of star and chain query patterns respectively. As a SPARQL graph pattern \( SC \) (an example is shown in Figure 6) is a composite graph pattern, we decompose it into a set of star and chain query patterns and then use the precomputed statistics of star and chain paths to obtain the overall selectivity. For maximally using precomputed statistics, we hope to obtain a set of edge-disjoint precomputed star and chain paths as building blocks that constitute a subgraph \( SC' \) of graph pattern \( SC \), which has the largest number of edges.

There is a special case we need to consider. A SPARQL query may include non-property-bound triple patterns. For this case, we remove such triple patterns from graph patterns (reserving join nodes) and estimate the selectivities of these triple patterns as single triple patterns. At last, we combine the selectivity of the whole graph pattern with the independent assumption.

We use \( PS \) to refer to the set of star and chain paths whose statistics have been precomputed. Removing a path \( p \) from graph \( Q \), denoted by \( Q \setminus p \), consists of removing all edges of \( p \) from \( Q \), followed by removing all stand-alone nodes. Given a query pattern \( Q \) and precomputed path set \( PS \), the maximum path cover of \( Q \) with respect to \( PS \) denoted by \( Cov_{PS}(Q) \) is a subset of edge-disjoint paths in \( PS \), which can constitute a subgraph \( Q' \) of \( Q \) such that \( Q' \) has the largest number of edges. Note that \( Q' \) could be \( Q \) itself.

Algorithm 2 GreedyFindpathCover

**Input:** Path set \( PS \), Query \( Q \);  
**Output:** A maximum path cover set \( Cov_{PS}(Q) \);

1: priority queue \( PQ = \Omega; Cov_{PS}(Q) = \Omega; \)
2: for all path \( p_k \) in \( PS \) do
3: if \( p_k \subseteq Q \) then
4: insert \( p_k \) to the priority queue \( PQ \); \{according to \( p_k \)’s length\}
5: while not empty(\( PQ \)) do
6: remove(\( PQ, p_k \));
7: if \( p_k \subseteq Q \) then
8: \( Cov_{PS}(Q) = Cov_{PS}(Q) \cup p_k; \)
9: \( Q = Q \setminus p_k; \)
10: if \( Q \) is empty then
11: return \( Cov_{PS}(Q); \)
12: return \( Cov_{PS}(Q); \)

Our problem is to obtain \( Cov_{PS}(Q) \). It is similar to the classic 0-1 knapsack problem: Given a set of items, each
with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than a given limit and the total value is as large as possible.

In our problem, the items are precomputed chain paths and star paths. For a chain or star path item, the value of this item is the length of the path and the weight is just the path since we use this path as a building block. Now, we need to determine the status for each path item ("picked" or "not picked") so that the picked edge-disjoint paths (or sub-paths) can constitute a subgraph $Q'$ of $Q$, which has the largest number of edges. The weight limit here is the query graph pattern $Q$.

**Algorithm 3 OptimalFC**

**Input:** A path array $PS[1..n]$, A boolean array $flag[1..n]$, Graph pattern $Q$;  
**Output:** A path cover set $Cov_{PS}(Q)$ of $Q$;

1: Initial flag $[1..n]=false$; 
2: int $maxNum = maxCov(n, Q)$; {get the number of edges covered, function $maxCov(n, Q)$ is defined below} 
3: for $k := 1$ to $n$ do 
4: if $flag[k] = true$ then 
5: add $PS[k]$ to $Cov_{PS}(Q)$; 
6: return $Cov_{PS}(Q)$; 
1: Function int $maxCov(n, Q)$; 
2: if $n = 0$ or $Q = \emptyset$ then 
3: return 0; 
4: if path $PS[i] \notin Q$ then 
5: return $maxCov(n-1, Q)$; 
6: if $maxCov(n-1, Q) \leq edgeNum(PS[i]) + maxCov(n-1, Q \setminus PS[i])$ then 
7: $Q = Q \setminus PS[i]$; 
8: $flag[i] = true$; {indicating that path $PS[i]$ is picked} 
9: return $maxCov(n-1, Q \setminus PS[i])$; 
10: else 
11: return $maxCov(n-1, Q)$;

The straightforward way of finding $Cov_{PS}(Q)$ is to adopt greedy strategy (presented in Algorithm 2 GreedyFindPathCover), which is eager to use the longest paths in $PS$ to constitute a subgraph of $Q$. The time complexity of the algorithm is $O(|PS| \cdot |E|)$, where $|PS|$ is the number of paths in $PS$ and $|E|$ is the number of edges of $Q$. Due to the local optimum nature of the algorithm, GreedyFindPathCover would fail to achieve the global optimum (i.e., the maximum path cover of $Q$) in some cases. However, this algorithm is still of interest because it provides an efficient way to find a path cover of $Q$ when graph pattern $Q$ is huge and dense.

Now we propose an optimal algorithm based on dynamic programming. Dynamic programming is a method for solving those problems which exhibit the property of optimal substructures by breaking them into overlapping subproblems. It is also applicable to our problem. Given a graph pattern $Q$, we represent $PS$ as a path array $PS[1..n]$ that contains all precomputed chain paths, star paths. Let $maxCov(i, Q)$ be the number of edges of subgraph $Q'$ of $Q$ such that $Q'$ consists of edge-disjoint paths from $PS[1..i]$ and has the largest number of edges. Here we use the last item $i$ and the remaining graph pattern (current weight limit) to index subproblems. $edgeNum(PS[i])$ indicates the edge number of path item $j$ in $PS$. $maxCov(i, Q)$ can be obtained through combining the optimal solutions of its subproblems.

We have the following properties:

\[ \begin{align*} 
& . maxCov(0, Q) = 0; \\
& . maxCov(i, Q) = 0, \text{ if } Q \text{ is null graph}; \\
& . maxCov(i, Q) = maxCov(i-1, Q), \text{ if } PS[i] \notin Q; \\
& . maxCov(i, Q) = max(\text{maxCov}(i-1, Q), \text{edgeNum}(PS[i]) + \text{maxCov}(i-1, Q \setminus PS[i])); 
\end{align*} \]

Obviously, when $i = 0$ or graph pattern $Q$ is a null graph, $maxCov(i, Q) = 0$. If path $PS[i]$ is not a subgraph of $Q$, we drop $PS[i]$ and $maxCov(i, Q)$ is equal to its subproblem $maxCov(i-1, Q)$; otherwise, $maxCov(i, Q)$ is computed through comparing the values in two cases (where $PS[i]$ is picked or not picked). Based on these properties, we present the algorithm in Algorithm 3 OptimalFC. The time complexity of our optimal algorithm is $O(|PS| \cdot |Q|)$, where $|Q|$ is the number of subgraphs of $Q$, which is exponential to the number of edges of $Q$. In practice, if query graph pattern $Q$ is sparse (i.e., the constituent vertices are of low degree) and not large, we can employ this optimal algorithm; otherwise, the greedy algorithm proposed can be used instead.

### 5.2 Joining a star pattern with a chain pattern

After a composite graph pattern is decomposed into a set of star and chain query patterns, how do we compute the selectivity of this graph pattern? To address this problem, let us first begin with a simple case shown in Figure 6.

In Figure 6, the composite graph pattern $SC$ can be decomposed into a star query pattern $S$ and a chain query pattern $C$ whose corresponding star path and chain path have been computed. The variable $Y$ is the join node. We use $val(CY)$ to refer to the value set of variable $CY$ and $val(SY)$ for variable $SY$. Our aim here is to estimate the selectivity of $SC$ denoted by $sel(SC)$.

Using the Bayesian networks and the chain histogram constructed, we can get $sel(S)$ and $sel(C)$. However, estimating precisely the selectivity of $SC$ is not trivial. A tuple $s$ matching star query pattern $S$ may not join with a tuple $c$ matching chain query pattern $C$, since they may have different values on variable $Y$. For estimating precisely the selectivity of $SC$, it is crucial to know the probability distributions over $val(SY)$ and $val(CY)$. However, obtaining the exact probability distributions of values over $val(SY)$ and $val(CY)$ is hard without executing the star query pattern $S$ and chain query pattern $C$. Fortunately, the probability distribution of abstracted values on the join variable $Y$ (target variable) can be inferred from the Bayesian network.

For example, suppose we have the Bayesian network shown in Figure 3 and we can infer the probability distribution of abstracted values over $val(SY)$ (i.e., the probability distribution over the object values of property $TeacherOf(TOF)$ for short) as follows:

\[
Pr(\text{TOF} = 'Adc'| \text{Edu} = 'PhD', \text{Pos} = 'Prof.', \text{Inc} \leq '70K')
= Pr(\text{TeacherOf} = 'Adc'| \text{Pos} = 'Prof.') = 0.7
\]

\[
Pr(\text{TOF} = 'Pre'| \text{Edu} = 'PhD', \text{Pos} = 'Prof.', \text{Inc} \leq '70K')
= Pr(\text{TOF} = 'Pre'| \text{Pos} = 'Prof.') = 0.3
\]

We can see that 70 percent of tuples matching pattern $S$ have value “Adc” and 30 percent of tuples have value “Pre”. This shows when $s$ joins with $c$, the result is affected by variable $Y$.
"Prc" on variable ?Y. Using the probability distribution over \( val(S.Y) \), we can easily compute the selectivity of SC. Recall that we use the chain histogram to estimate the selectivities of chain query patterns. Now we take the values "Add" and "PrC" as the values of start node of chain C separately, then test chain query patterns "Add/C", "PrC/C" in the chain histogram and get the selectivities \( sel(Add/C) \) and \( sel(PrC/C) \). Since bloom filter is efficient for answering set membership queries, this process is time efficient. At last, we can obtain the selectivity of SC as \( 0.7 \cdot sel(Add/C) \cdot sel(Add/C) + 0.3 \cdot sel(Add/C) \cdot sel(PrC/C) \). This process is described in Algorithm 4 ComSelStar-Chain.

Algorithm 4 ComSelStar-Chain

Input: A composite graph pattern \( SC(S \bowtie C) \) consisting of a chain pattern \( C \) and a star pattern \( S \); Join node ?Y;

Output: The selectivity of chain pattern \( SC \);

1: count = 0;
2: for all abstracted values \( a_i \) over the join node ?Y do
3: \( \text{Compute the probability } Pr(Y = a_i | M(S)) \); \( M(S) \) are the observed values in \( S \);
4: \( count_1 = Pr(Y = a_i | M(S)) \times sel(S) \); \{compute sel(\( S \) using formula (3))\}
5: \( count_2 = GetChainSel(a_i, C) \);
6: count = count_1 \times count_2;
7: return count;

Algorithm 5 ComSelStar-Chain

Input: A composite graph pattern \( S(S_1 \bowtie S_2) \) consisting of two star pattern \( S_1 \) and \( S_2 \); Join node ?Y;

Output: The selectivity of \( S \);

1: count = 0;
2: for all abstracted values \( a_i \) over the join node ?Y do
3: \( \text{Compute the probability } Pr(Y = a_i | M(S_1), M(S_2)) \); \( M(S_1), M(S_2) \) are the observed values in \( S_1, S_2 \);
4: \( count_1 = Pr(Y = a_i | M(S_1)) \times sel(S_1) \);
5: \( count_2 = Pr(Y = a_i | M(S_2)) \times sel(S_2) \);
6: count = count_1 \times count_2;
7: return count;

5.3 Other cases

We have discussed the case where a composite graph pattern can be decomposed into one star pattern and one chain pattern. What if a composite graph pattern is decomposed into two star patterns (Figure 7(a)) or two chain patterns (Figure 7(b))? If a graph pattern \( S(S_1 \bowtie S_2) \) is decomposed into two star patterns \( S_1 \) and \( S_2 \), we go through each possible abstracted value \( a_i \) on the join node ?Y of two star paths. Through inference on the Bayesian networks constructed, it is easy to acquire the probabilities \( Pr(Y = a_i | M(S_1)) \) and \( Pr(Y = a_i | M(S_2)) \) such that \( M(S_1) \) and \( M(S_2) \) are observed object values in two star patterns. We then get the selectivity of the composite graph pattern through cumulating the count of tuples from two star patterns which have the same values on the join node. The algorithm is described in Algorithm 5 ComSelStar-Star.

In the case where the graph pattern \( C(C_1 \bowtie C_2) \) is decomposed into two chain query patterns \( C_1, C_2 \) that join on variable ?Y, the method is similar (presented in Algorithm 6 ComSelChain-Chain). We go through each possible abstracted value \( a_i \) of the join node, and the selectivities of two chain query patterns with value \( a_i \) can be obtained using the chain histogram. \( sel(C_1 \bowtie C_2 | Y = a_i) \) is computed through combining \( sel(a_i/C_1) \) and \( sel(a_i/C_2) \) in the product form. At last, we cumulate \( sel(C_1 \bowtie C_2 | Y = a_i) \) with all possible abstracted values. Since bloom filter is quick to get the selectivities of two chain patterns, this method is also time efficient.

Algorithm 6 ComSelChain-Chain

Input: A composite graph pattern \( C(C_1 \bowtie C_2) \) consisting of two chain patterns \( C_1, C_2 \); Join node ?Y;

Output: The selectivity of chain pattern \( C \);

1: count = 0;
2: for all abstracted values \( a_i \) over the join node ?Y do
3: \( count_1 = GetChainSel(a_i, C_1) \);
4: \( count_2 = GetChainSel(a_i, C_2) \);
5: \( count = count_1 \times count_2 \);
6: return count;

Figure 7: Join between chain patterns and star patterns, where "?Y" is the join node.

Now we extend our method to the more general case where the path cover set \( CovPS(Q) \) of query \( Q \) consists of multiple star patterns and chain patterns. We select two patterns \( p_i \) and \( p_j \) from \( CovPS(Q) \) that can join together. Compute the selectivity for the composite pattern \( p_{ij} = p_i \bowtie p_j \) maintain
a selectivity distribution table $T_{pq}$ indicating the selectivity distributions over the possible abstracted values of the join node in $p_{ij}$. Iterate this process until there is only one composite graph pattern left in $CoreSP(Q)$. For the part of $Q$ that is not covered by $CoreSP(Q)$, we deal with it as a set of single triple patterns and combine the results of them in the product form with the independence assumption. The formal description of this algorithm is presented in Algorithm 7 ComSelectivity.

6. EXPERIMENTS

6.1 Experiment Setup

We implemented the proposed algorithms. All algorithms are run on a windows XP professional operating system. The hardware is a PC with Intel Pentium 4 3.0GHz CPU, 4 GB memory.

<table>
<thead>
<tr>
<th></th>
<th>CoreSP</th>
<th>CoreSP+</th>
<th>CoreSP+1</th>
</tr>
</thead>
<tbody>
<tr>
<td># Quads</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td># Triads</td>
<td>2</td>
<td>34</td>
<td>&gt;6</td>
</tr>
<tr>
<td># Pairs</td>
<td>12</td>
<td>12</td>
<td>13</td>
</tr>
</tbody>
</table>

Figure 8: Query sets used in experiments.

Data sets. Both synthetic and real-world datasets are used in our experiments. In this paper, we present results on three datasets: (1) LUBM [13] is developed by Lehigh University; (2) SwetoDBLP [14] is a dataset describing real computer science bibliography information. (3) YAGO [15] consists of facts extracted from Wikipedia. Compared with the former two datasets, it is relatively heterogeneous. The details of three datasets are shown in Table 1.

<table>
<thead>
<tr>
<th></th>
<th>LUBM</th>
<th>DBLP</th>
<th>YAGO</th>
</tr>
</thead>
<tbody>
<tr>
<td># Triples</td>
<td>25,511,071</td>
<td>11,014,618</td>
<td>40,114,899</td>
</tr>
<tr>
<td># Instances</td>
<td>4,555,428</td>
<td>2,395,467</td>
<td>3,883,360</td>
</tr>
<tr>
<td># Properties</td>
<td>43</td>
<td>45</td>
<td>93</td>
</tr>
</tbody>
</table>

Table 1: Statistics of datasets

Query Loads. There are three kinds of query patterns used in our experiments: star, chain and composite query patterns. Figure 8 shows some information about query sets used in the experiments. We use the seed queries (part of them are shown in Appendix) with some random literal values to generate different query sets.

Previous Techniques. We compare our method with two previous works: 1) In [4], the authors propose the selectivity estimation method PF based on the probabilistic framework. They build summary statistics of joined triple patterns for RDF data; 2) In [9], the authors propose the estimation method in RDF-3X system, which computes two kinds of statistics for selectivity estimation. The first one is called specialized histograms which assumes independence among predicates. The second statistics computes frequent join paths in the data.

Evaluation Method. We use relative error $RE$ to present the performance of different techniques of selectivity estimation. Given a true selectivity $sel$ and its estimate $\tilde{sel}$, we use the relative error formula $RE(sel, \tilde{sel}) = \frac{|sel - \tilde{sel}|}{max(1,sel)}$ for positive queries ($sel > 0$) and negative queries ($sel = 0$).

6.2 Offline Evaluation

Learning Bayesian Networks. Before evaluating our methods for star query patterns, we first constructed Bayesian networks for frequent star paths in three datasets. For LUBM dataset, we constructed cluster property tables for 6 common star paths that correspond entities such as Faculty, Student, Course and so on. An entity may correspond several classes. Similarly, we constructed cluster property tables for 7 common star paths) in DBLP dataset. Then Bayesian networks are learned from these cluster property tables. Clearly, with the increase of space for storing learned Bayesian networks, the constructed Bayesian networks and CPTs are more accurate over the data. Due to the efficiency consideration, we set the space limit of each dataset for storing each Bayesian network to 16 KB. Some details about learning Bayesian networks are shown in Table 2. From Table 2, we can see that the learning time of Bayesian networks for three datasets are 127.3s, 232.2s, 2112.9s, respectively. In DBLP dataset, we have learned 7 Bayesian networks from cluster property tables and the largest cluster property table contains 362,907 tuples. However, the time of learning Bayesian network for this cluster property table is 722s, which is acceptable. For YAGO dataset, since the data is heterogeneous there are more frequent paths and we construct 100 Bayesian Networks for them.

Table 2: Some information about chain histograms

<table>
<thead>
<tr>
<th></th>
<th>LUBM</th>
<th>DBLP</th>
<th>YAGO</th>
</tr>
</thead>
<tbody>
<tr>
<td># Chain paths Contained</td>
<td>124</td>
<td>323</td>
<td>500</td>
</tr>
<tr>
<td>Length of chain paths</td>
<td>2-5</td>
<td>2-7</td>
<td>2-5</td>
</tr>
<tr>
<td>Constructing time (s)</td>
<td>179</td>
<td>357s</td>
<td>840s</td>
</tr>
<tr>
<td>Space limit (KB)</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
</tbody>
</table>

Constructing chain histograms. To employ our chain histogram based method to estimate the selectivity of chain patterns, we constructed chain histograms for chain paths of three datasets. The space limit for storing the chain histogram of each dataset is 16 KB. Some details of constructing chain histograms are shown in Table 3.

Table 3: Some information about chain histograms

<table>
<thead>
<tr>
<th></th>
<th>LUBM</th>
<th>DBLP</th>
<th>YAGO</th>
</tr>
</thead>
<tbody>
<tr>
<td># Chain paths Contained</td>
<td>124</td>
<td>323</td>
<td>500</td>
</tr>
<tr>
<td>Length of chain paths</td>
<td>2-5</td>
<td>2-7</td>
<td>2-5</td>
</tr>
<tr>
<td>Constructing time (s)</td>
<td>179</td>
<td>357s</td>
<td>840s</td>
</tr>
<tr>
<td>Space limit (KB)</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
</tbody>
</table>

6.3 Estimation Accuracy

In this set of experiments, we study and compare the accuracy of different selectivity estimation methods. Figure 9
shows the accuracy of three methods for different query sets on three datasets. In all figures, X-axis are different query sets and Y-axis is the average relative error RE.

Performance of methods for star query patterns.
In this experiment, we evaluate our method of selectivity estimation for star query patterns. For each dataset, 15 star query patterns are developed. They are grouped into 3 query sets: Qs1, Qs2, Qs3 (see Figure 8). Qs1, Qs2 and Qs3 contain star query patterns that have 2, 3 to 4, and more than 5 properties, respectively. Each query set contains 5 star query patterns. The relative errors of different methods on three datasets are shown in Figure 9(a), (b) and (c). “BNM” indicates our Bayesian network based selectivity estimation method for star query patterns.

From Figure 9(a), (b) and (c), we can see that our method for star patterns dominates the other methods because we capture the dependencies among properties in the queries. The RDF-3x method also have better results on query set Qs1 in which query patterns have the small number of properties and literal constraints. In this case, the join uniformity assumption does not affect much. Conversely, in query sets Qs2 and Qs3, queries have more properties and literal constraints. It is more likely that the join uniformity assumption affects the accuracy of estimation. Thus our Bayesian network based method obtains much better results on query sets Qs2 and Qs3 than the other two methods that adopt the join uniformity assumption.

Performance of methods for chain query patterns.
From Figure 9(d), (e) and (f), we can see performance of methods on chain query patterns. For each dataset, 15 chain query patterns are developed. They are grouped into 3 query sets: Qc1, Qc2, Qc3 (See Figure 8 for more details). “CHM” indicates our chain histogram based method for estimating the selectivity of chain patterns. Our method dominates the other methods. The RDF-3x method also have better results on query set Qc1 in which the lengths of chain query patterns are relatively small. Conversely, in query sets Qc2 and Qc3, lengths of chain query patterns are longer. It is more likely that the join uniformity assumption affect the accuracy of estimation. Thus our method obtains much better results on query sets Qc2 and Qc3 than the other two methods.

Performance of methods for composite query patterns.
In this experiment, we evaluate our method for composite query patterns. We develop 20 composite query patterns for each dataset. All these queries are also grouped into 4 query sets: Qsc1, Qsc2, Qsc3, Qsc4. Some details of query sets are shown in Figure 8. Figure 9(g), (h) and (i) show the results of four methods on three datasets, where “Greedy” and “Optimal” stand for two decomposition algorithms (shown in Algorithm 2 and Algorithm 3 respectively). Our methods obtain more accurate estimation since we construct the refined model when dealing with joined triple patterns and do not adopt join uniformity assumption. The optimal algorithm also outperforms the greedy algorithm because the latter may not find the maximum path cover of query patterns. And for the uncovered part of queries, greedy algorithm would adopt the independence assumption for selectivity estimation, which affects the estimation accuracy. However, figures show that Greedy algorithm still performs better than PF and RDF-3x methods.

6.4 Online Running Time

<table>
<thead>
<tr>
<th># Queries</th>
<th>Qsc1</th>
<th>Qsc2</th>
<th>Qsc3</th>
<th>Qsc4</th>
</tr>
</thead>
<tbody>
<tr>
<td># Edges</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>11</td>
</tr>
</tbody>
</table>

In this experiment, we study the on-line running time of our estimation methods adopting two decomposition algorithms: Greedy algorithm and Optimal algorithm on YAGO dataset which is relatively heterogeneous. Figure 10 shows the average running time of two methods as the average number of edge of query sets increases. The running time of the method with the Greedy algorithm reveals a slow growth with the increase of the average number of edge of queries set. We can observe that the time cost is reasonable even when the average number of edge of query set Qsc4 rises to 11. Compared with the Greedy algorithm, the running time of the method with the Optimal algorithm is higher because the Optimal algorithm tries to find the maximum path cover of each composite query pattern. The time costs of two cases are close when graph patterns are small (the number of edge <= 8), but the running time of Optimal algorithm rises significantly when the number of edge exceeds 8. It suggests
the Greedy decomposition algorithm is preferable for larger and dense graph patterns.

7. RELATED WORK

Selectivity estimation is well studied in relational databases. Histogram is one of the most important statistical estimation data structure in relational DBMSs. [17] offers a latest, comprehensive survey on this subject. Join Synopses [16] enable accurate estimation by summarizing the combined join and value distribution across foreign-key joins of several tables. In our work, we estimate the composite graph patterns by summing over the possible abstracted values of the joined variables and do not have the restriction of foreign-key joins.

In [4], the authors propose the framework of static Basic Graph Pattern (BGP) optimization based on selectivity estimation. They devise a number of heuristics for the selectivity estimation of joined triple patterns. The heuristics range from simple variable counting techniques to more sophisticated selectivity estimation based on the probabilistic framework. In [9] the authors propose two kinds of statistics for selectivity estimation. The first one is a histogram, which is generic and can handle any kind of triple patterns and joins with independence assumption. The second statistics computes frequent join paths in the data, which are similar to our methods. Nevertheless, [4, 9] both take the join uniformity assumption when estimating the selectivity of joined triple patterns. We avoid this assumption and propose a more accurate estimation model that takes correlations among properties into account.

8. CONCLUSION

With the increasing amount of RDF data, efficient and scalable management of RDF data has become a fundamental challenge to achieve the semantic web vision. Selectivity estimation is critical to processing RDF queries efficiently. To address this problem, we employ the Bayesian network and chain histogram techniques for estimating the selectivity of star and chain query patterns. For an arbitrary composite graph pattern, the algorithm which combines the precomputed statistics of chain and star paths to estimate the overall selectivity is proposed. The experiments show that our method outperforms existing approaches in accuracy significantly. In the future, we will investigate the appropriate way to generalize our methods to support other applications in SPARQL query processing.

9. ACKNOWLEDGMENTS

This research was supported by the Australian Research Council Discovery Projects DP110102407 and DP0878405.

10. REFERENCES


APPENDIX

A. PART OF SEED QUERIES ON YAGO.

- Qs1: ?s hasProductionLanguage '%Random_Language%'. ?s type 'Movie'.
- Qc3: ?x0 hasChild ?x1. ?x1 hasPredecessor ?x2. ?x2 hasPredecessor ?x3. ?x3 politicianOf ?x4. ?x4 hasCapital '%Random_City%'.
- Qsc2: 's type wordnet:actor_100765278. ?s actedin ?m1. ?m1 type 'movie'. ?m1 hasProductionLanguage 'English'. ?s directed ?m2. ?m2 type 'movie'. ?m2 hasProductionLanguage 'English'. ?s livesIn ?city. ?city locatedIn ?state. ?state locatedIn 'New_York'.

Figure 10: Online Running Time