Improving cardinality estimation of transitive closures in graphs

Punter, Wieger

Award date:
2020

Improving cardinality estimation of transitive closures in graphs

W. R. Punter
N. Yakovets, T. Mulder

July 4, 2020

Abstract

This thesis contains a study on the estimation of cardinality per path length of a transitive closure for a binary relation in a graph database. An accuracy problem in an existing synopsis based method in the case of a binary relation is identified. An approach to improve this method is proposed. We propose a method that splits the transitive closures in two cases and predicts accurate for the most frequent case. A direction for further research in the other case is also stated. This research has the goal to contribute to a more efficient computation of a transitive closure in graph databases.
Keywords: graphs; transitive closure; synopsis method; cardinality estimation

1 Introduction

When people use a database, they use *declarative programming languages*, such as SQL or Datalog. These languages are a way to communicate to the database what information the user wants to receive, without specifying the exact procedure for receiving this information. This communication is done by *queries*, lines of code that represent a question.
The actual execution of a query is called a query plan. Multiple query plans can answer the same query. It is the task of the database optimizer to choose the most efficient query plan to minimize running time.
One of the factors the optimizer takes into account is the *cardinality* of a query: how many unique rows are in the (intermediate) result of the query? For some query plans, it could be the case that the execution will be performing unnecessary early large joins.
The problem with this cardinality is that the computation of the true cardinality of a query often takes longer than the query itself. It is possible to provide an estimate of the cardinality that uses information that can possibly be computed in less time than computing the true value.
This thesis will focus on cardinality estimation for a *transitive closure* query, which will be clarified later. This estimation is dependent on the underlying data structure. For relational databases, this concept is well-researched. In the context of *graph databases*, there is less well-researched.

1.1 Research question

This thesis aims to contribute to the improvement of existing estimation methods of the cardinality of a transitive closure in a graph database. Only simple *binary relations* are considered.
We will lay focus on the path length between vertices in the closure. The prediction of the cardinality for concatenated new pairs of vertices might differ in accuracy between the first and i-th concatenation. If an accurate estimator of the cardinality for this closure is found, it might be possible to apply this estimator in a computation of the transitive closure. It would be valuable to check whether the improvement is only more efficient in theory or would also be performing better in practice.
The research question will be formulated as follows:

What is an accurate method for estimating the cardinality per path length of a transitive closure in graph databases for a simple binary relation?

This question can be divided into two sub questions.

1. What are accuracy related **problems** in existing methods for estimating the cardinality per path length of a transitive closure in graph databases for a simple binary relation?

2. What are **solutions** to improve the accuracy in existing methods for estimating the cardinality per path length of a transitive closure in graph databases for a simple binary relation?

1.2 Contributions

- This thesis has identified a **bias** in the existing synopsis based method for the estimation in the considered case of a simple binary relation.

- This thesis proposes a direction for the improvement of the accuracy in the existing synopsis based method.

1.3 Structure

The structure of the coming sections is as follows. The preliminary knowledge section contains the definition of a graph, binary relation, transitive closure and cardinality. The related work section includes three existing methods for the computation of a transitive closure and how an estimate of cardinality would contribute to the computation. It also contains an existing estimation method, based on some statistics of the base relation. The problem that arises from the specific case of this thesis is described after the explanation of the existing method. An approach for the solution for the problem is proposed. To test this approach, three experiments are performed. The results and an analysis of the results follow after this. The conclusion contains a summary of the main findings. The final section is the discussion on limitations of this research and possible further improvements.

2 Preliminary knowledge

2.1 Graph definition

A graph G is a data structure that consists of a set vertices (V), a set edges (E) and a function λ. λ is a function that assigns a label (predicate) to every edge in E, specifying the meaning of the relationship between two vertices. Vertices are called nodes as well. There is a distinction between source and target nodes in an edge.
An example is a graph with vertices v_1 = "Alice" and v_2 = "Bob". They could be connected by an edge (v_1, v_2) with label "*knows*". Node v_1 is a source node and v_2 is a target node in edge (v_1, v_2).

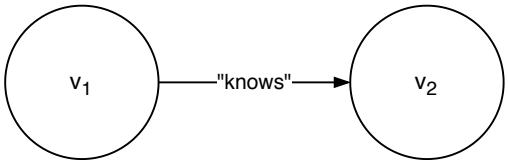

Figure 1: Graph example

2.2 Binary relation

A binary relation is a set $R \subseteq V \times V$, with pairs of nodes that are connected in E. A binary relation can satisfy a regular expression if all edges in paths between nodes in the relation have labels that satisfy a regular expression. This thesis is only considers simple binary relations. These relations satisfy a regular expression of the form $(a)^+$ where a is a label.
Suppose we add Claire (v_3) who lives next to Bob and knows Alice. As indicated with green arrows in figure 2, R with regular expression ("*knows*")$^+$ would contain (v_1, v_2) and (v_3, v_1). Notice that (v_2, v_3) is not in R, because Bob and Claire do not know each other according to the graph.

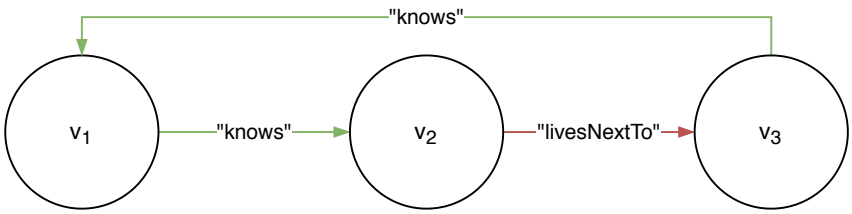

Figure 2: Binary relation example

An example of a more complex binary relation R_c would be if the regular expression was ("knows"/"livesNextTo")*. In this case, R_c would have pairs of nodes with a path labeled with this specific sequence. R_c would be (v_1, v_3) in the graph in figure 2.

2.3 Transitive closure

To explain the transitive closure query, we take a graph $G = \{V, E, \lambda\}$ and binary relation $R \subseteq V \times V$. The transitive *closure* (R^+) is the smallest binary relation in G that contains R and is transitive. According to (Bonifati et al., 2018), R is *transitive* on a graph G if for all nodes s, v, t $\in V$, the following proposition holds: $(s, v) \in$ R $\wedge$ $(v, t) \in R \implies (s, t) \in R$.
The closure must also contain all pairs of nodes in the base relation, which are all pairs $(s, v) \in R$.
To summarize, the transitive closure can be seen as a set that contains:

1. All pairs of nodes in the base relation of R.
2. All pairs of nodes with a path of edges between them whose labels satisfy the regular expression of the base relation.

The transitive closure of our relation R would contain (v_1, v_2), (v_3, v_1) and by the transitive property also (v_3, v_2).

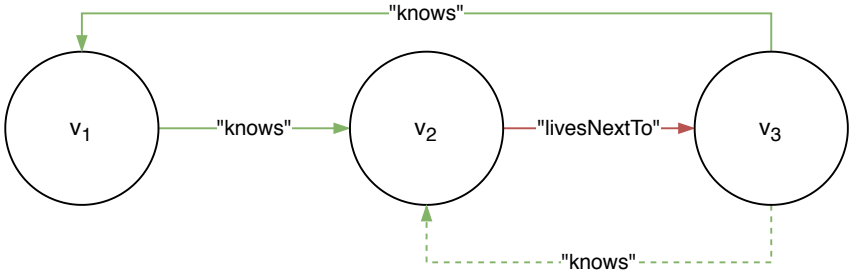

Figure 3: Transitive closure example

2.4 Cardinality

The cardinality of a relation is defined as the number of unique pairs of nodes in a relation. The source and target node cardinality in a relation is the number of unique source or target nodes in a relation.

- The cardinality of a transitive closure is the number of unique pairs of nodes in the closure.
- The cardinality per path length i in a transitive closure of a relation that satisfies predicate $(a)^*$ is the number of distinct pairs of nodes with paths labeled $(a)^i$.

The cardinality of the base relation in figure 3 is 2. The cardinality at path length 2 is 1.

3 Related work

3.1 Computation of transitive closures

The transitive closure can be computed in several ways that differ in complexity (Bonifati et al., 2018). Three methods are briefly discussed in how they evolve in their computation.

Algorithm 1 Naive TC

1: $C_0 := \mathrm{B}$
2: $C_1 := join(B, C_0)$
3: $i := 0$
4: **while** $C_{i+1} \setminus C_i \neq \emptyset$ **do**
5: $\quad i := i + 1$
6: $\quad C_{i+1} := extend(C_i, B)$
return $B^+ = C_{i+1}$

Algorithm 2 Semi-Naive TC

1: $C_0 := \emptyset$
2: $\Delta_0^R := B$
3: $i := 0$
4: **while** $\Delta_i^R \neq \emptyset$ **do**
5: $\quad C_{i+1} := cache(C_i, \Delta_i^R)$
6: $\quad \Delta_{i+1} := crank(B, \Delta_i^R)$
7: $\quad \Delta_i^R := reduce(\Delta_i, C_{i+1})$
8: $\quad i := i + 1$
return $B^+ = C_i$

3.1.1 Naive TC

The first method is the naive computation of the closure, which can be found in algorithm 1. Suppose we have a base relation B as input.

1. The base relation is copied to C_0. C_i is the partial closure up to path length $i + 1$. This is why the partial closure at path length 1 is the base relation.
2. The naive computation joins B with itself to C_1 on the predicate that the target nodes in B are the same as source nodes in C_0. This set C_1 contains all pairs of nodes $(s, t) \in B$ and the pairs that are related by two edges. This is the partial transitive closure up to path length two.
3. In the while loop, the *extend* operation is performed. This operation can be written as:

$$C_{i+1} \leftarrow C_i \cup \pi_{C_i.s, B.t}(C_i \bowtie_{C_i.t=B.s} B)$$

   The partial closure (C_i) is joined with the base relation to find the new pairs of the next path length ($i + 2$). These new pairs are added to the partial closure to define the partial closure of C_{i+1}.
4. The while loop ends if $C_{i+1} \setminus C_i = \emptyset$, which means that there are no new pairs added in the join. At that moment, the partial closure is the complete transitive closure.

This method performs potential **redundant** work by joining the partial closure with the base relation. Especially in **cyclic graphs**, this could be an endless recursive query, because pairs of nodes can be added to the closure infinitely many times. In non-cyclic graphs, the method still performs redundant work. This is because pairs of nodes are rediscovered in every iteration.
Suppose the label between v_2 and v_3 in our example is changed from "livesNextTo" to "knows".

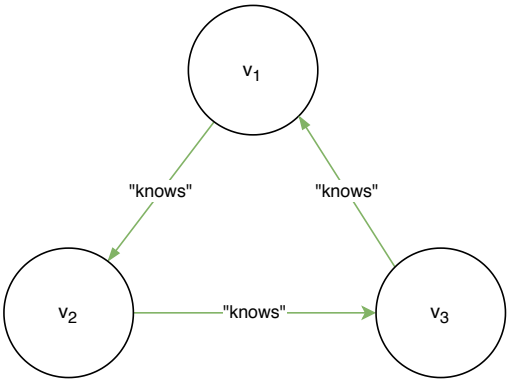

Figure 4: Example graph

The naive computation of the transitive closure on $R_{(\text{"}knows\text{"})^*}$ in figure 4 rediscovers the base relation in the third iteration, as depicted in figure 5. This figure shows the pairs that will be discovered by the transitive property in the first three iterations of the naive computation. The computation does not terminate, because of the cycle.

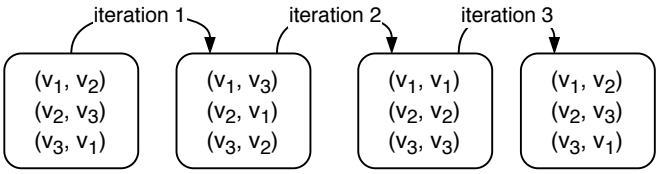

Figure 5: Discovered pairs in iterations of the naive computation

3.1.2 Semi-naive TC

The Semi-Naive computation (algorithm 2) is an improvement on the naive computation, because it guarantees termination on graphs that contain cycles. It explores every path only once. The computation uses the differential relation (Δ_i^R) to store pairs discovered at iteration i of the algorithm.

1. The partial closure (C_0) is empty at the start. Initially, the differential relation (Δ_0^R) is the base relation.
2. As long as there are new pairs ($\Delta_i^R \neq \emptyset$), the while loop performs three methods.
   (a) The *cache* method integrates Δ_i^R into C_{i+1}.
   (b) After this, new pairs of the next path length are found in the *crank* operation. This operation joins the current Δ_i^R with the base relation into Δ_{i+1}.
   (c) In the *reduce* operation, duplicates are checked. Pairs in Δ_{i+1} are dropped if they are already in the partial closure C_i or are not unique in Δ_{i+1} itself. This results in a new Δ_i^R.
3. The algorithm terminates when there are no new pairs left to discover. This is when $\Delta_I^R = \emptyset$.

So, the join in the *crank* function is contrary to naive not performed by the base relation and the partial closure, but the base relation and the differential relation. The method terminates if there are no new pairs found after the join. This will cause graphs with cycles to terminate and prevent redundant pairs in the transitive closure. Figure 6 shows the differential relation per path length of the transitive closure of $R_{(\text{"}knows\text{"})^*}$ in figure 4. Notice that for the naive computation, there would be new pairs in iteration 3. This is not the case for Semi-Naive.

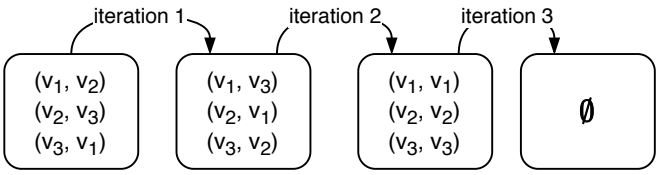

Figure 6: Differential relation Semi-Naive computation

3.1.3 SMART TC

In the previous computations, the join in the while loop is performed on the base relation. This way, the path length increases linear per iteration. In the SMART computation (algorithm 3), this is not the case. The SMART algorithm makes use of the closure depth $d_{s,t}$. This depth is defined recursively as:

- The depth $d_{s,t}$ for all pairs of nodes in the base relation is 1.
- For all nodes s, v, t in V the following holds: $(d_{s,v} = k) \wedge (d_{v,t} = l) \Rightarrow d_{s,t} = k + l$.

Algorithm 3 SMART TC

1: $\Delta_0^R := init(B)$
2: $C_0 := \Delta_0$
3: $i := 0$
4: **while** $\Delta_i^R \neq \emptyset$ **do**
5: $\quad \Delta_{i+1} := crank(\Delta_i^R, C_i)$
6: $\quad \Delta_{i+1}^R := reduce(\Delta_{i+1}, C_i)$
7: $\quad C_{i+1} := cache(C_i, \Delta_i^R)$
8: $\quad i := i + 1$
return $B^+ = C_i$

The partial closure in the SMART algorithm is stored as a set of triples ($< s, t, d_{s,t} >$) with pairs of nodes (s, t) and a corresponding closure depth ($d_{s,t}$). The SMART computation of algorithm 3 works as follows.

1. Initially, the differential relation (Δ_0^R) contains the base relation. All triples have depth 1 here. The partial closure (C_0) is initially defined as the same differential relation.
2. While the differential relation is not empty, the *crank*, *reduce* and *cache* procedures are executed.
   (a) The *crank* operation is defined as follows:

   $$\Delta_{i+1} \leftarrow \pi_{C_i.s, \Delta_i.t}(C_i \bowtie_{C_i.t=\Delta_i.s} \sigma_{d=2^i}(\Delta_i)).$$

   Here, the partial closure C_i is joined with all triples in C_i that have closure depth 2^i. This means that Δ_{i+1} contains all triples in the transitive closure that have closure depth 1 to $2 * 2^i$.
   (b) The *reduce* operation reduces duplicates in the differential relation, as in Semi-Naive.
   (c) The *cache* stores the partial closure at $i + 1$ as C_i and Δ_i^R.

In figure 7, the computation in Semi-Naive (left) is compared to SMART (right). The closure depth indicates the path length for which the pairs are computed in a step of the algorithm. Suppose n is the number of iterations to compute a transitive closure in Semi-Naive. Then, the SMART computation computes the complete transitive closure in $log_2(n)$.

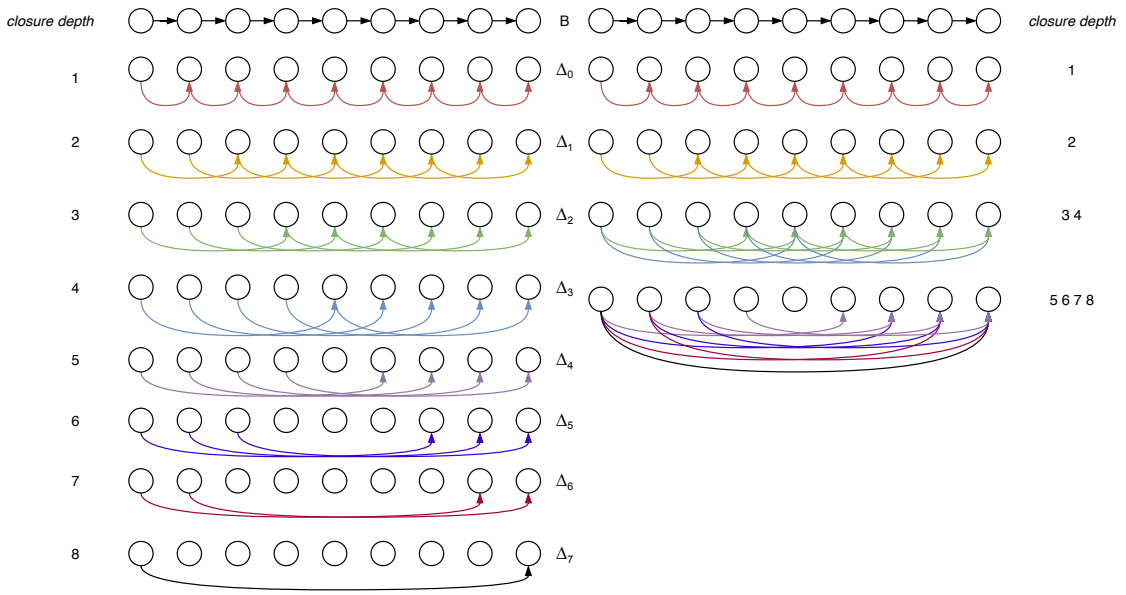

Figure 7: Comparing Semi-Naive (left) vs. SMART (right) (Bonifati et al., 2018)

So, SMART uses fewer iterations than Semi-Naive. This does not mean that SMART is always more efficient than Semi-Naive. SMART joins larger relations in those iterations. This could have an impact on the time the *reduce* operation takes. Depending on the structure of a graph, it might be the case that Semi-Naive would be more efficient than SMART (Bonifati et al., 2018). The WAVEGUIDE procedure in (Bonifati et al., 2018) is not discussed, because the method only improves the efficiency for more complex relations than a simple binary relation.

3.1.4 Importance of estimation

The computation of transitive closures is a process that is worth optimizing. As can be seen in the SMART method, it is possible to compute the closure in a non-linear way. The drawback of this method is that a lot of working memory is needed if closures on big graph databases are computed. If there would be an accurate estimate of the size of the differential relation Δ_i^R at concatenation i, it would be possible to prevent unnecessary large joins. In the case that a query is only interested in pairs at path length 4, it could be the case that Δ_4^R is computed faster by joining Δ_2^R with itself than joining Δ_3^R with the base relation. The database would then choose a query plan with this earlier join, to optimize the efficiency.

3.2 Cardinality estimation

As explained in section 2.4, the cardinality of a transitive closure is the number of unique pairs of nodes in that closure. According to (Bonifati et al., 2018), the true cardinality can be computed by summing the number of new pairs in every iteration of the Semi-Naive computation to the transitive closure.
In this method, the computation of the true cardinality has the same complexity as the computation of the closure itself. To use the cardinality to compute the transitive closure more efficient, the cardinality needs to be estimated. There are five main methods (Bonifati et al., 2018) for cardinality estimation: synopses, sketches, histograms, sampling and wavelets. Yakovets mentions a synopsis based method to estimate this cardinality in a graph context.

3.2.1 Synopsis based method

The method in (Yakovets, 2016) estimates the cardinality of new pairs at every path length $i+1$, based on estimates of the previous path length i and statistics on the graph. The statistics are stored in two **synopses**, explained in table 1. The estimate of the base case, path length 2, is based on the true values for the base relation and these synopses.

Two assumptions are made on the graph, according to (Bonifati et al., 2018). These assumptions allow us to estimate the cardinality of a join of two relations.

- The first is the **uniformity** assumption. Here, it is assumed that all nodes j in a differential relation Δ_i have the same number of tuples associated with them in both relations.
- The second assumption is the **independence** assumption. This assumes that predicates on sources and targets are mutually independent in both relations.

Synopsis	Name	Description
SYN1	out	The number of nodes in G which have outgoing edge labeled with l.
	in	The number of nodes in G which have incoming edge labeled with l.
	paths	The number of paths in G labeled with l.
	pairs	The number of distinct node pairs connected with paths labeled with l.
SYN2	out	The number of nodes in G which have outgoing path labeled with l_1/l_2.
	in	The number of nodes in G which have incoming path labeled with l_1/l_2.
	middle	The number of nodes in G which have incoming edge labeled l_1 and outgoing edge labeled l_2.
	paths	The number of paths in G labeled with l_1/l_2.
	pairs	The number of distinct node pairs connected with paths labeled with l_1/l_2.
	one	The number of paths labeled l_1 from nodes in **out** to nodes in **middle**.
	two	The number of paths labeled l_2 from nodes in **middle** to nodes in **in**.

Table 1: Statistical information stored in (joint-)frequency synopses SYN1 and SYN2 (Bonifati et al., 2018)

The existing synopsis based method is described in algorithm 4 and works as follows.

1. The algorithm starts with the base relation and computes synopsis 1 and 2 for a label a.
2. After this, the true cardinality of the base relation (j_1) is computed. So is the true cardinality of the source (s_1) and target nodes (t_1) in the base relation.
3. Because *uniformity* and *independence* are assumed, the source, pair and target cardinalities at path length can be estimated in line 4, 5 and 6. The reasoning behind these formulas can be found in section 6.5 of (Yakovets, 2016).
4. After the estimation of the base case, cardinalities for higher path lengths can be estimated in line 10, 11 and 12 in the while loop.
5. This terminates if the estimated cardinality of the next iteration is zero or lower, or if there are too many iterations.

Algorithm 4 Synopsis method

Input: Graph $G(V, E, \lambda)$**, label** a

1: $B = V \times V$ with edge labels a.
2: syn1 = syn1(a); syn2 = syn2(a, a)
3: s_1 = unique source nodes in B; j_1 = unique pairs in B; t_1 = unique target nodes in B.
4: $\hat{s}_2 = s_1 \cdot \frac{syn2.middle}{syn1.in}$
5: $\hat{j}_2 = j_1 \cdot \frac{syn2.Two}{syn1.in}$
6: $\hat{t}_2 = t_1 \cdot \frac{syn2.in}{syn1.in}$
7: $i = 2$
8: $count = 0$
9: **while** $\hat{j}_i > 0$ and $count < 1000$ **do**
10: $\quad \hat{s}_{i+1} = \hat{s}_i \cdot \frac{syn2.middle}{syn1.in}$
11: $\quad \hat{j}_{i+1} = \frac{syn2.middle}{syn1.in} \cdot \frac{(\hat{j}_i)^2}{\hat{s}_i}$
12: $\quad \hat{t}_{i+1} = \hat{t}_i \cdot \frac{syn2.in}{syn1.in}$
13: $\quad i+ = 1$; $count+ = 1$
14: **return** $\hat{S} = \hat{s}_2...\hat{s}_i$; $\hat{J} = \hat{j}_2...\hat{j}_i$; $\hat{T} = \hat{t}_2...\hat{t}_i$

3.3 Bias

The previous method is focused on complex regular expressions. In the simple $(a)^*$ expression that is considered in this thesis, there is a bias. This is not immediately considered a problem.
In other cases, the statistics stored in both synopses of an iteration depend on the last labels of paths. These labels are the same for every iteration in this case. This causes the estimates to be only dependent on the estimates of the previous iteration instead of also on changes in the synopses.
There is a bias in the estimation algorithm 4, because the factor used in line 11 and 12 ($\frac{SYN2.middle}{SYN1.in}$) is less than or equal to 1.
This factor is less than or equal to 1, because $A_{SYN1.in} \subseteq B_{SYN2.middle}$. Here, $A_{SYN1.in}$ is the set of nodes in G with incoming edge labeled with a. This set has size $SYN1.in$. $B_{SYN2.middle}$ is the set of nodes in G with incoming edge labeled with a *and* outgoing edge labeled with a. This set has size $SYN2.middle$. A contradiction will occur if a node $v \in B_{SYN2.middle}$ and $v \notin A_{SYN1.in}$ is assumed. So, we can conclude the following.

$$\frac{SYN2.middle}{SYN1.in} \leq 1 \tag{1}$$

The presence of a problem is dependent on the relation between the source node cardinality of the base relation s_1 and the cardinality of the base relation j_1. Every unique source node participates in at least one unique pair. Therefore, there are two cases, either case 1: $j_1 = s_1$ or case 2: $j_1 > s_1$.

3.3.1 Case 1: $j_1 = s_1$

If $j_1 = s_1$, the estimated cardinality of the differential relation at path length i: $\hat{j_{i+1}}$ will also be equal to the estimated source node cardinality of the differential relation at path length i: $\hat{s_{i+1}}$ with $i > 1$ and $i \in \mathbb{R}$. In base case $j_1 = s_1$, it holds that $\hat{j_2} = \hat{s_2}$.

- From line 5 in algorithm 4, we have $\hat{j_2} = \frac{(j_1)^2}{s_1} \cdot \frac{SYN2.middle}{SYN1.in}$. This becomes $\hat{j_2} = j_1 \cdot \frac{SYN2.middle}{SYN1.in}$, if $j_1 = s_1$.
- From line 4 in algorithm 4, we have $\hat{s_2} = s_1 \cdot \frac{SYN2.middle}{SYN1.in}$. $\hat{j_2} = \hat{s_2}$ if $j_1 = s_1$.

In iteration i, it holds that $\hat{j_{i+1}} = \hat{s_{i+1}}$ if $\hat{j_i} = \hat{s_i}$ according to the same reasoning used in the base case, but with line 10 and 11 of algorithm 4.
Because of equation 1, the estimated amount of source nodes in the join ($\hat{s_i}$) is either constant or decreases with each iteration in line 10. This is not necessarily a problem, because the cardinality of source nodes cannot increase over iterations. The estimate of the cardinality of new pairs in the join ($\hat{j_i}$) are also not necessarily biased by this factor. The estimate can only remain constant or decrease. Because the true cardinality can also only remain constant or decrease, we state that the estimator and the true distribution behave in the same way. However, it would be interesting know if the true cardinality and source node cardinality are also the same for every iteration in this specific case.

We want to prove that if $s_1 = j_1$ is true, $s_i = j_i$ for $i > 1$ and $i \in \mathbb{R}$. The case where $s_i > j_i$ is not possible, following the same reasoning $s_1 \leq j_1$. To disprove that $s_i < j_i$ can be true, we need a proof by contradiction.

1. Assume $s_1 = j_1$.
2. This implies that every source node in the base relation only has one outgoing edge.
3. Base case
   (a) Assume a base case $j_2 > s_2$
   (b) This implies a source node v in the base relation that has a path that satisfies $(a)^2$ to two nodes x and y with $x \neq y$. There are two situations in which this is possible
      - v might have two outgoing edges to nodes n and m where n has an edge to x and m has an edge to y.
      - v could also have one outgoing edge to a node m that has two edges to x and y.
   (c) In both cases, there would be a contradiction between statement 2 and either v or m having two outgoing edges.
4. In an iterative step, if $s_i = j_i$, we also find a contradiction if $s_{i+1} < j_{i+1}$ is assumed. If $s_{i+1} < j_{i+1}$, at least one source node is needed with two outgoing nodes. This is not possible.
5. So, if $s_1 = j_1$ is true, $s_i = j_i$ for $i > 1$ and $i \in \mathbb{R}$.

Because of this, the true cardinality j_i can only stay constant or decrease per iteration. The estimate of $\hat{j}_i$ also has this behaviour.

3.3.2 Case 2: $j_1 > s_1$

In the case where $j_1 > s_1$, the relation of $\hat{j}_i$ and $\hat{j}_{i+1}$ with $i > 1$ and $i \in \mathbb{R}$ is dependent on the relation between $\hat{j}_i$ and $\hat{s}_i \cdot \frac{SYN1.in}{SYN2.middle}$. This factor $\hat{s}_i \cdot \frac{SYN1.in}{SYN2.middle}$ is the inverse of the factor $\frac{SYN2.middle}{SYN1.in \cdot \hat{s}_i}$ which is multiplied with $\hat{j}_i$ in line 11 of algorithm 4 to estimate the cardinality of the next path length. So essentially, if $\hat{j}_i$ is larger than the factor, the estimate of the cardinality at the next iteration (j_{i+1}) increases. The behaviour of the estimator needs to be analyzed in the three base cases where j_1 is lower than, equal to or higher than the factor.

Case a: $j_1 > s_1 \cdot \frac{SYN1.in}{SYN2.middle}$

- $j_1 > s_1 \cdot \frac{SYN1.in}{SYN2.middle} \implies \hat{j_2} > j_1$, following line 5 of algorithm 4.
- The estimated source node cardinality $\hat{s_2}$ can only be equal or less than the source cardinality of the base relation s_1, following line 4.
- Therefore, $\hat{j}_2 > \hat{s}_2 \cdot \frac{SYN1.in}{SYN2.middle}$.
- In an iterative step, it holds that if $\hat{j}_i > \hat{s}_i \cdot \frac{SYN1.in}{SYN2.middle}$, $\hat{j}_{i+1} > \hat{j}_i$.
- Since this is the case and $\hat{s}_{i+1} \leq \hat{s}_i$, it will also hold that $\hat{j}_{i+1} > \hat{s}_{i+1} \cdot \frac{SYN1.in}{SYN2.middle}$.
- Therefore, all $\hat{j_{i+1}} > \hat{j}_i$ with $i \geq 1$ and $i \in \mathbb{R}$ if $j_1 > s_1 \cdot \frac{SYN1.in}{SYN2.middle}$

Case b: $j_1 = s_1 \cdot \frac{SYN1.in}{SYN2.middle}$

- $j_1 = s_1 \cdot \frac{SYN1.in}{SYN2.middle} \implies \hat{j_2} = j_1$.
- From $j_1 = s1 \cdot \frac{SYN1.in}{SYN2.middle}$, we can express $\frac{SYN2.middle}{SYN1.in}$ as $\frac{s_1}{j_1}$.
- Following the previous and line 10 and 11 of algorithm 4, we are able to express the following.
  - $\hat{s}_{i+1} = \frac{s_1^{(i+1)}}{j_1^i}$.
  - $\hat{j}_{i+1} = \hat{j}_i^2 \cdot \frac{j_1^{(i-2)}}{s_1^{(i-1)}}$
- With these expressions and the assumption that $j_1 > s_1$, we can see that the difference between $\hat{j}_{i+1}$ and $\hat{j}_i$ can only become larger after this iteration.
- In an iterative step, $\hat{j}_{i+1} > \hat{j}_i$.

Case c: $j_1 < s_1 \cdot \frac{SYN1.in}{SYN2.middle}$

- $j_1 < s_1 \cdot \frac{SYN1.in}{SYN2.middle} \implies \hat{j_2} < j_1$.
- It is not necessarily the case that $\hat{j_2} < \hat{s_2} \cdot \frac{SYN1.in}{SYN2.middle}$, because $\hat{s_2}$ also decreases with a factor of $\frac{SYN2.middle}{SYN1.in}$.
- So, if $j_1 < s_1 \cdot \frac{SYN1.in}{SYN2.middle}$, $\hat{j}_{i+1} < \hat{j}_i$ does not always hold.

$\hat{j}_i > \hat{s}_i$ still holds. If $\hat{j}_i$ ever becomes greater than $\hat{s}_i \cdot \frac{SYN1.in}{SYN2.middle}$, it will stay greater in next iterations.

After analyzing these three cases, we can state that the behaviour of the estimate $\hat{j}_i$ has a **positive bias** if $j_1 > s_1$. The estimated cardinality per path length will go to infinity. The estimation algorithm only converges because after 1000 iterations, the while loop in algorithm 4 is terminated. Unlike the behaviour of j_i when $j_1 = s_1$, the true cardinality per path length j_i does not behave the same way as the estimate in this case. The true j_i will always converge to zero at some iteration, because the graph is finite and cycles are only explored once. The behaviour of j_i also does not seem to be dependent on the relation between j_1 and $s_1 \cdot \frac{SYN1.in}{SYN2.middle}$. The existing estimator is **inaccurate** in the case that the number of unique pairs in the base relation is greater than the number of unique source nodes in the base relation. This is a problem in need of a solution.

4 Approach

The problem with the application of the existing synopsis based method in estimating the cardinality for simple relations, needs a solution. An approach that this thesis applies that is not applied in (Yakovets, 2016), is to make use of information at path length 2 of the closure. In the application of the estimator, one will always compute the transitive closure at path length 2. After this iteration, it might be more efficient to join path length 2 with itself to prevent an unnecessary large join at path length 3. Because of this approach, there is more knowledge on the behaviour of a specific graph.
From section 3.3, we know that the existing estimator behaves similar to the true cardinality if the cardinality of the base relation is the same as the source node cardinality of the base relation. This is because both are only decreasing: the cardinality distribution has no maximum.

We argue that if a factor can be found that accurately represents the decrease in cardinality per path length for transitive closures that only decrease, we are able to improve the existing method. This is only interesting if we can do the following.

1. Find if there is a significant amount of transitive closures that have no maximum.
2. Classify whether a transitive closure is likely to have no maximum.

The information on the cardinality of the base relation and path length 2 is useful for this classification.

To find a way to use this information, patterns need to be found between the true cardinality and relative variables from the synopses and the cardinalities of the first two path lengths. The exact experiments and variables are specified in table 2.

5 Methods

5.1 Data description

To inspect the behaviour of transitive closures and their cardinality, DBPedia ("The DBPedia Knowledge Base.", n.d.) is used. DBPedia contains at least 623 predicates with a simple binary transitive closure of at least path length 3. These predicates are interesting for a transitive closure analysis, because the cardinality of path length 3 or higher needs to be estimated.
In figure 8, the distribution of the predicates is shown in three categories.

1. Category one entails transitive closures with a cardinality distribution that decreases every path length. Therefore, these closures have zero maxima. This category occurs in 88.4% of the 623 predicates.
2. The second category consists of transitive closures with a cardinality distribution that has one maximum. This occurs in 8.2% of the predicates.
3. The third category consists of transitive closures with multiple maxima in the cardinality distribution. This occurs in 3.4% of the cases

The transitive closures were computed by a Semi-Naive computation in PostgreSQL.

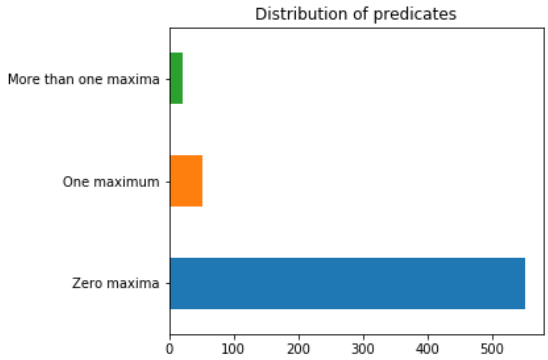

Figure 8: Distribution of predicates

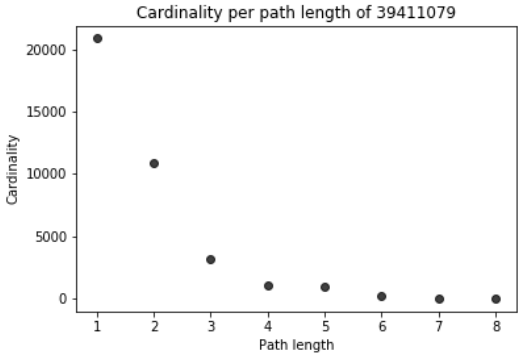

Figure 9: Cardinality plot of predicate with zero maxima

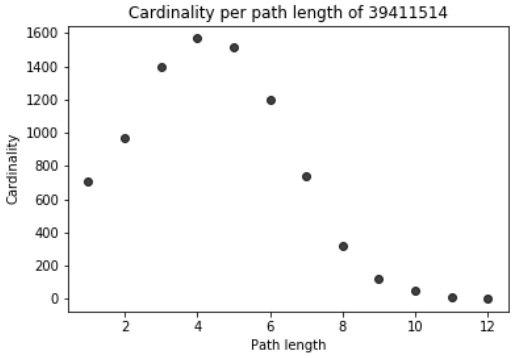

Figure 10: Cardinality plot of predicate with one maximum

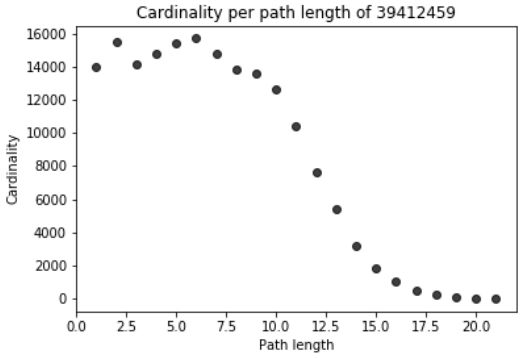

Figure 11: Cardinality plot of predicate with multiple maxima

5.2 Experiment setup

The most frequent cardinality distribution in a transitive closure has zero maxima. The estimator can use this information in combination with information on the relation between the cardinality at path length 2 (j_2) and the cardinality of the base relation (j_1). Three experiments are done to gain more understanding of the behaviour of these cardinalities and possible estimators. The general idea is to split predicates in two sets. Set 1 has predicates where $j_2 < j_1$. Set 2 has predicates where $j_2 \geq j_1$.

5.2.1 Experiment 1

For set 1, the assumption is made that since the cardinality is already decreasing, it will have no maximum. An experiment 1 is set up to model the decrease in cardinality. Four models are used that all predict the cardinality at the next path length $\hat{j}_{i+1}$ as as product of $\hat{j}_i$ and a factor below one.

1. Model 1: $\hat{j}_{i+1} = \hat{j}_i \cdot \frac{j_2}{j_1}$
2. Model 2: $\hat{j}_{i+1} = \hat{j}_i \cdot \sqrt{\frac{j_2}{j_1}}$
3. Model 3: $\hat{j}_{i+1} = \hat{j}_i \cdot \frac{j_2}{j_1})^{1/(i-1)}$
4. Model 4: $\hat{j}_{i+1} = \hat{j}_i \cdot \frac{j_2}{j_1})^{(i-1)}$

The mean absolute error (MAE) and the mean squared error (MSE) are computed per predicate. After this, the mean and sample standard deviations of the MAE and MSE over all predicates were computed. This is done to gain an intuition on what direction the development of a final model should head. Both the MAE and the MSE are negatively oriented scores, which mean that a lower error is better for the accuracy of a model. The difference between the two measures is that the MSE gives a high weight to large errors, where the MAE gives every error equal weight.

5.2.2 Experiment 2

In experiment 2, the Spearman's linear correlation test ("Spearman Rank Correlation Coefficient", 2008) is performed to find out if one of the relative variables named in table 2 correlates with whether a transitive closure has either one or multiple maxima. This test has H_0 of no correlation between the variables in 2 and the classification of either having one or having more than one maxima. Important to denote is that the test does not assume that data is normally distributed. An α of 0.05 is used to test for significance. The variables were chosen, because they are relative and can thus be used in general for all transitive closures. These variables capture the ratio of change in size for all synopses between path length one and two.

5.2.3 Experiment 3

Experiment 3 searches in transitive closures with one maximum for correlations between the path length of the maximum and the variables in table 2 on the graph. The Spearman's linear correlation test ("Spearman Rank Correlation Coefficient", 2008) is used to test for significance, as well as in experiment 2.

$\frac{j_2}{j_1}$	Cardinality at path length 2 divided by cardinality at path length 1.
$\frac{s_2}{s_1}$	Source node cardinality at path length 2 divided by source node cardinality at path length 1.
$\frac{t_2}{t_1}$	Target node cardinality at path length 2 divided by target node cardinality at path length 1.
$\frac{SYN2.in}{SYN1.in}$	Number of nodes with incoming path a/a divided by number of nodes with incoming path a.
$\frac{SYN2.out}{SYN1.out}$	Number of nodes with outgoing path a/a divided by number of nodes with outgoing path a.
$\frac{SYN2.middle}{SYN1.in}$	Number of nodes with incoming path a and outgoing path a divided by number of nodes with incoming path a.
$\frac{SYN2.middle}{SYN1.out}$	Number of nodes with incoming path a and outgoing path a divided by number of nodes with outgoing path a.
$\frac{SYN2.paths}{SYN1.paths}$	The number of paths labeled with a/a divided by the number of paths labeled with a.
$\frac{SYN2.pairs}{SYN1.pairs}$	The number of distinct node pairs connected by paths labeled a/a divided by distinct node pairs connected by an edge labeled a.
$\frac{SYN2.one}{SYN1.paths}$	Number of paths between nodes with outgoing path a and nodes with incoming path a and outgoing path a divided by the number of paths labeled with a/a.

Table 2: Independent variables in experiments 2 and 3

6 Results

6.1 Experiment 1

Model	Mean MAE	Std MAE	Mean MSE	Std MSE
1	351	2477	571	4239
2	514	3120	811	5256
3	495	2921	705	4512
4	351	2484	593	4282

Table 3: Results experiment 1

The statistics in the table hint towards a preference for model 1 or 4, because of a lower mean MAE and MSE. However, the standard deviation is too high in every model to draw a significant conclusion. The experiment provides the intuition that more steeply downward models have a lower error and lower standard deviation. Figure 12 shows the true cardinality (black) and the model 1 prediction (red) per path length for one predicate.

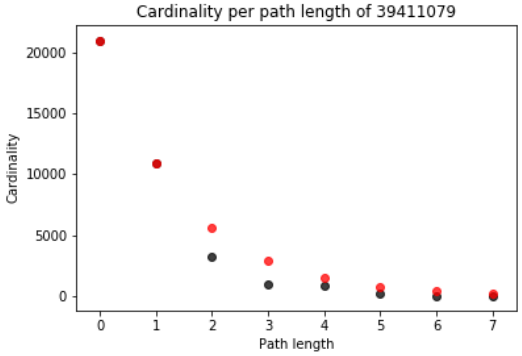

Figure 12: True and estimated cardinality of one predicate with model 1

6.2 Experiment 2

Variable	Correlation	P-value
$\frac{j_2}{j_1}$	-0.025	0.882
$\frac{s_2}{s_1}$	-0.182	0.261
$\frac{t_2}{t_1}$	-0.176	0.278
$\frac{SYN2.in}{SYN1.in}$	-0.188	0.245
$\frac{SYN2.out}{SYN1.out}$	-0.182	0.261
$\frac{SYN2.middle}{SYN1.in}$	-0.146	0.370
$\frac{SYN2.middle}{SYN1.out}$	-0.182	0.261
$\frac{SYN2.one}{SYN1.paths}$	-0.206	0.202
$\frac{SYN2.paths}{SYN1.paths}$	-0.152	0.350
$\frac{SYN2.pairs}{SYN1.pairs}$	-0.097	0.551

Table 4: Results experiment 2

None of the variables has a significant linear correlation with the presence of one or more maxima.

6.3 Experiment 3

Variable	Correlation	P-value
$\frac{j_2}{j_1}$	0.220	0.212
$\frac{s_2}{s_1}$	0.095	0.591
$\frac{t_2}{t_1}$	0.054	0.760
$\frac{SYN2.in}{SYN1.in}$	0.053	0.767
$\frac{SYN2.out}{SYN1.out}$	0.102	0.565
$\frac{SYN2.middle}{SYN1.in}$	0.338	0.051
$\frac{SYN2.middle}{SYN1.out}$	0.204	0.247
$\frac{SYN2.one}{SYN1.paths}$	0.365	0.034
$\frac{SYN2.paths}{SYN1.paths}$	0.146	0.410
$\frac{SYN2.pairs}{SYN1.pairs}$	0.178	0.314

Table 5: Results experiment 3

The only significant linear correlation is that between $\frac{SYN2.one}{SYN1.paths}$ and the position of the maximum. This correlation is 0.365.

7 Analysis

From the results of experiment 1, we conclude that it is possible to estimate the decline in cardinality in a more accurate way than the existing method in (Yakovets, 2016). The results of experiment 2 and 3 were interesting for a further research. The application of these findings in a general estimation algorithm is as follows:

1. First, the partial closure at path length 2 has to be computed.
2. If the cardinality at path length 2 is **lower** than the cardinality of the base relation, the assumption is made that this transitive closure has no maximum. In this case, the cardinality at other path lengths is predicted via model 1: $\hat{j}_{i+1} = \hat{j}_i \cdot \frac{j_2}{j_1}$.
3. If the cardinality at path length 2 is **higher** than the cardinality of the base relation, the cardinality at other path lengths is estimated via the existing method. The cause is the lack of a model that predicts the cardinality more accurate.
    - We did find a correlation that might be worth for further investigation. This is a significant correlation between the position of a maximum if there is only one maximum and the ratio between the number of nodes that have path a/a and the number of nodes that have path a ($\frac{SYN2.one}{SYN1.paths}$).
    - We did not find a significant correlation that could help to classify whether a transitive closure has either only one or has more than one maxima.

8 Conclusion

In this research, the synopsis based method of Yakovets is investigated with an application in simple binary relations. These simple binary relations satisfy a regular expression $(a)^*$, where a is a queried predicate. The main findings in this research are that:

- A bias is found in the existing synopsis method when relations that satisfy a regular expression in the form $(a)^*$ are queried. This bias causes an accuracy related problem. The problem is that the estimator predicts a cardinality of infinity in the case the cardinality of the base relation is higher than the source node cardinality of the base relation.
- The solution to this problem is studied in three experiments that contributed to a general model which estimates the cardinality of a transitive closure for the gross of the relations used in this thesis.

The estimation algorithm is performing better than the existing method in our specific case, which can be applied in a more efficient computation of the transitive closure in the case of a regular expression $(a)^*$.

The answer to the research question is that the synopsis method can be improved into a more accurate method for estimating the cardinality per path length of a transitive closure in graph databases for a simple binary relation. This thesis detected a problem in the current estimator and made an attempt on improving the estimator in a specific situation. This improvement is in need of further development.

9 Discussion

9.1 Practical

In further research, the experiments should be run on more than one database. Because this is not done in this research, the experiments could be subject to overfitting since the DBPedia database could differ in context from other databases. The conclusion that the transitive closures have zero maxima in 88% of the cases, could be wrong.

For experiment 1, there might be a model that captures the decline in a more accurate way. A model that includes statistics from the synopses would be a starting point in exploring other models.
The correlation tests in experiment 2 and 3 are only testing for linear correlation. A test for non-linear correlation with the setup of experiment 2 and 3 would be useful to gain more understanding on the dependencies.
The set of variables in the experiments 2 and 3 could also be extended by interaction variables between the synopses and true cardinalities.
For the further research on the classification of either one or more maxima, it would be interesting to take a machine learning based approach. This could help in exploring relations between the variables and whether the transitive closure has one or more maxima. This would also be an advice in an approach for finding the position of the maxima.

9.2 Theoretical

Because this research has an extensive focus on a synopsis based method, other methods named in section 3.2 could turn out to be valuable methods for the estimation. The focus on the synopses method was chosen because of the bottom-up approach used in the algorithm. This causes the method to be able to estimate the cardinality at certain path lengths.
An interesting angle to investigate is how this research could combine with a query where one queries one specific source or target node v_1 and wants all nodes and path lengths that that is related with v_1.

References

Bonifati, A., Fletcher, G., Voigt, H., Yakovets, N., Claude, U., & Lyon, B. (2018). *Querying Graphs* (H. V. Jagadish, Ed.). Michigan: Morgan & Claypool publishers. doi: https://doi.org/10.2200/S00873ED1V01Y201808DTM051

The DBPedia Knowledge Base. (n.d.). Retrieved from `http://dbpedia.org/`

Spearman Rank Correlation Coefficient. (2008). In *The concise encyclopedia of statistics* (pp. 502–505). New York, NY: Springer New York. Retrieved from `https://doi.org/10.1007/978-0-387-32833-1_379` doi: 10.1007/978-0-387-32833-1{_}379

Yakovets, N. (2016). Optimization of regular path queries in graph databases.