Context-Free Path Querying by Matrix Multiplication

Rustam Azimov
Saint Petersburg State University
St. Petersburg, Russia
rustam.azimov19021995@gmail.com

Semyon Grigorev
Saint Petersburg State University
St. Petersburg, Russia
Semen.Grigorev@jetbrains.com

ABSTRACT

Graph data models are widely used in many areas, for example, bioinformatics, graph databases. In these areas, it is often required to process queries for large graphs. One of the most common graph queries are navigational queries. The result of query evaluation is a set of implicit relations between nodes of the graph, i.e. paths in the graph. A natural way to specify these relations is by specifying paths using formal grammars over the alphabet of edge labels. An answer to a context-free path query in this approach is usually a set of triples (A, m, n) such that there is a path from node m to node n, whose labeling is derived from a non-terminal A of the given context-free grammar. This type of queries is evaluated using the *relational query semantics*. Another example of path query semantics is *single-path query semantics* which requires to present a single path from node m to node n, whose labeling is derived from a non-terminal A for all triples (A, m, n) evaluated using relational query semantics. There is a number of algorithms for query evaluation which use these semantics but all of them perform poorly on a large graphs. One of the most common technique for efficient big data processing is GPGPU, but these algorithms do not allow to use this technique efficiently. In this paper we show how the context-free path query evaluation using these query semantics can be reduced to the calculation of the matrix transitive closure. Also we propose an algorithm for context-free path query evaluation which uses relational query semantics and is based on matrix operations that make it possible to speed up computations by using GPGPU.

KEYWORDS

Transitive closure, CFPQ, graph databases, context-free grammar, GPGPU, matrix multiplication

1 INTRODUCTION

Graph data models are widely used in many areas, for example, bioinformatics [2], graph databases [10]. In these areas, it is often required to process queries for large graphs. The most common among graph queries are navigational queries. The result of query evaluation is a set of implicit relations between nodes of the graph, i.e. paths in the graph. A natural way to specify these relations is by specifying paths using formal grammars (regular expressions, context-free grammars) over the alphabet of edge labels. Context-free grammars are actively used in graphs queries because of the limited expressive power of regular expressions.

The result of context-free path query evaluation is usually a set of triples (A, m, n) such that there is a path from node m to node n, whose labeling is derived from a non-terminal A of the given context-free grammar. This type of query is evaluated using the *relational query semantics* [6]. Another example of path query semantics is *single-path query semantics* [7] which requires to present a single path from node m to node n whose labeling is derived from a non-terminal A for all triples (A, m, n) evaluated using relational query semantics. There is a number of algorithms for context-free path query evaluation using these semantics [5, 6, 17].

Existing algorithms for context-free path query evaluation w.r.t. these semantics demonstrate poor performance when applied to big data. One of the most common technique for efficient big data processing is *GPGPU* (General-Purpose computing on Graphics Processing Units), but these algorithms do not allow to use this technique efficiently. The algorithms for context-free language recognition had a similar problem until Valiant [14] proposed a parsing algorithm which computes a recognition table by computing matrix transitive closure. Thus, the active use of matrix operations (such as matrix multiplication) in the process of a transitive closure computation makes it possible to efficiently apply GPGPU computing techniques [3].

We address the problem of creating an algorithm for context-free path query evaluation using the relational and the single-path query semantics which allows us to speed up computations with GPGPU by using the matrix operations.

The main contribution of this paper can be summarized as follows:

- We show how the context-free path query evaluation w.r.t. relational and single-path query semantics can be reduced to the calculation of matrix transitive closure.
- We introduce an algorithm for context-free path query evaluation w.r.t. relational query semantics which is based on matrix operations that make it possible to speed up computations by means of GPGPU.
- We provide a formal proof of correctness of the proposed algorithm.
- We show the practical applicability of the proposed algorithm by running different implementations of our algorithm on real-world data.

2 PRELIMINARIES

In this section, we introduce the basic notions used throughout the paper.

Let Σ be a finite set of edge labels. Define an *edge-labeled directed graph* as a tuple $D = (V, E)$ with a set of nodes V and a directed edge-relation $E \subseteq V \times \Sigma \times V$. For a path π in a graph D we denote the unique word obtained by concatenating the labels of the edges along the path π as $l(\pi)$. Also, we write $n\pi m$ to indicate that a path π starts at node $n \in V$ and ends at node $m \in V$.

Following Hellings [6], we deviate from the usual definition of a context-free grammar in *Chomsky Normal Form* [4] by not including a special starting non-terminal, which will be specified in the path queries to the graph. Since every context-free grammar can be transformed into an equivalent one in Chomsky Normal Form and checking that an empty string is in the language is trivial it is sufficient to consider only grammars of the following type. A *context-free grammar* is a triple $G = (N, \Sigma, P)$, where N is a finite set of non-terminals, Σ is a finite set of terminals, and P is a finite set of productions of the following forms:

- $A \to BC$, for $A, B, C \in N$,
- $A \to x$, for $A \in N$ and $x \in \Sigma$.

Note that we omit the rules of the form $A \to \varepsilon$, where ε denotes an empty string. This does not restrict the applicability of our algorithm because only the empty paths $m\pi m$ correspond to an empty string ε.

We use the conventional notation $A \xrightarrow{*} w$ to denote that a string $w \in \Sigma^*$ can be derived from a non-terminal A by some sequence of applications of the production rules from P. The *language* of a grammar $G = (N, \Sigma, P)$ with respect to a start non-terminal $S \in N$ is defined by

$$L(G_S) = \{w \in \Sigma^* \mid S \xrightarrow{*} w\}.$$

For a given graph $D = (V, E)$ and a context-free grammar $G = (N, \Sigma, P)$ we define *context-free relations* $R_A \subseteq V \times V$, for every $A \in N$, such that

$$R_A = \{(n, m) \mid \exists n\pi m \,(l(\pi) \in L(G_A))\}.$$

We define a binary operation $(\,\cdot\,)$ on arbitrary subsets N_1, N_2 of N with respect to a context-free grammar $G = (N, \Sigma, P)$ as

$$N_1 \cdot N_2 = \{A \mid \exists B \in N_1, \exists C \in N_2 \text{ such that } (A \to BC) \in P\}.$$

Using this binary operation as a multiplication of subsets of N and union of sets as an addition, we can define a *matrix multiplication*, $a \times b = c$, where a and b are matrices of a suitable size that have subsets of N as elements, as

$$c_{i,j} = \bigcup_{k=1}^{n} a_{i,k} \cdot b_{k,j}.$$

According to Valiant [14], we define the *transitive closure* of a square matrix a as $a^+ = a_+^{(1)} \cup a_+^{(2)} \cup \cdots$ where $a_+^{(1)} = a$ and

$$a_+^{(i)} = \bigcup_{j=1}^{i-1} a_+^{(j)} \times a_+^{(i-j)}, \; i \geq 2.$$

We enumerate the positions in the input string s of Valiant's algorithm from 0 to the length of s. Valiant proposes the algorithm for computing this transitive closure only for upper triangular matrices, which is sufficient since for Valiant's algorithm the input is essentially a directed chain and for all possible paths $n\pi m$ in a directed chain $n < m$. In the context-free path querying input graphs can be arbitrary. For this reason, we introduce an algorithm for computing the transitive closure of an arbitrary square matrix.

For convenience of further reasoning, we introduce another definition of the transitive closure of an arbitrary square matrix a as $a^{cf} = a^{(1)} \cup a^{(2)} \cup \cdots$ where $a^{(1)} = a$ and

$$a^{(i)} = a^{(i-1)} \cup (a^{(i-1)} \times a^{(i-1)}), \; i \geq 2.$$

To show the equivalence of these two definitions of transitive closure, we introduce the partial order $\geq$ on matrices with fixed size that have subsets of N as elements. For square matrices a, b of the same size we denote $a \geq b$ iff $a_{i,j} \supseteq b_{i,j}$, for every i, j. For these two definitions of transitive closure, the following lemmas and theorem hold.

Lemma 2.1. *Let $G = (N, \Sigma, P)$ be a grammar, let a be a square matrix. Then $a^{(k)} \geq a_+^{(k)}$ for any $k \geq 1$.*

Proof. (Proof by Induction)

Basis: The statement of the lemma holds for $k = 1$, since

$$a^{(1)} = a_+^{(1)} = a.$$

Inductive step: Assume that the statement of the lemma holds for any $k \leq (p-1)$ and show that it also holds for $k = p$ where $p \geq 2$. For any $i \geq 2$

$$a^{(i)} = a^{(i-1)} \cup (a^{(i-1)} \times a^{(i-1)}) \Rightarrow a^{(i)} \geq a^{(i-1)}.$$

Hence, by the inductive hypothesis, for any $i \leq (p-1)$

$$a^{(p-1)} \geq a^{(i)} \geq a_+^{(i)}.$$

Let $1 \leq j \leq (p-1)$. The following holds

$$(a^{(p-1)} \times a^{(p-1)}) \geq (a_+^{(j)} \times a_+^{(p-j)}),$$

since $a^{(p-1)} \geq a_+^{(j)}$ and $a^{(p-1)} \geq a_+^{(p-j)}$. By the definition,

$$a_+^{(p)} = \bigcup_{j=1}^{p-1} a_+^{(j)} \times a_+^{(p-j)}$$

and from this it follows that

$$(a^{(p-1)} \times a^{(p-1)}) \geq a_+^{(p)}.$$

By the definition,

$$a^{(p)} = a^{(p-1)} \cup (a^{(p-1)} \times a^{(p-1)}) \Rightarrow a^{(p)} \geq (a^{(p-1)} \times a^{(p-1)}) \geq a_+^{(p)}$$

and this completes the proof of the lemma. □

Lemma 2.2. *Let $G = (N, \Sigma, P)$ be a grammar, let a be a square matrix. Then for any $k \geq 1$ there is $j \geq 1$, such that $(\bigcup_{i=1}^{j} a_+^{(i)}) \geq a^{(k)}$.*

Proof. (Proof by Induction)

Basis: For $k = 1$ there is $j = 1$, such that

$$a_+^{(1)} = a^{(1)} = a.$$

Thus, the statement of the lemma holds for $k = 1$.

Inductive step: Assume that the statement of the lemma holds for any $k \leq (p-1)$ and show that it also holds for $k = p$ where $p \geq 2$. By the inductive hypothesis, there is $j \geq 1$, such that

$$(\bigcup_{i=1}^{j} a_+^{(i)}) \geq a^{(p-1)}.$$

By the definition,

$$a_+^{(2j)} = \bigcup_{i=1}^{2j-1} a_+^{(i)} \times a_+^{(2j-i)}$$

and from this it follows that

$$(\bigcup_{i=1}^{2j} a_+^{(i)}) \geq (\bigcup_{i=1}^{j} a_+^{(i)}) \times (\bigcup_{i=1}^{j} a_+^{(i)}) \geq (a^{(p-1)} \times a^{(p-1)}).$$

The following holds

$$(\bigcup_{i=1}^{2j} a_+^{(i)}) \geq a^{(p)} = a^{(p-1)} \cup (a^{(p-1)} \times a^{(p-1)}),$$

since

$$(\bigcup_{i=1}^{2j} a_+^{(i)}) \geq (\bigcup_{i=1}^{j} a_+^{(i)}) \geq a^{(p-1)}$$

and

$$(\bigcup_{i=1}^{2j} a_+^{(i)}) \geq (a^{(p-1)} \times a^{(p-1)}).$$

Therefore there is $2j$, such that

$$(\bigcup_{i=1}^{2j} a_+^{(i)}) \geq a^{(p)}$$

and this completes the proof of the lemma. □

Theorem 1. *Let $G = (N, \Sigma, P)$ be a grammar, let a be a square matrix. Then $a^+ = a^{cf}$.*

Proof. By the lemma 2.1, for any $k \geq 1$, $a^{(k)} \geq a_+^{(k)}$. Therefore

$$a^{cf} = a^{(1)} \cup a^{(2)} \cup \cdots \geq a_+^{(1)} \cup a_+^{(2)} \cup \cdots = a^+.$$

By the lemma 2.2, for any $k \geq 1$ there is $j \geq 1$, such that

$$(\bigcup_{i=1}^{j} a_+^{(i)}) \geq a^{(k)}.$$

Hence

$$a^+ = (\bigcup_{i=1}^{\infty} a_+^{(i)}) \geq a^{(k)},$$

for any $k \geq 1$. Therefore

$$a^+ \geq a^{(1)} \cup a^{(2)} \cup \cdots = a^{cf}.$$

Since $a^{cf} \geq a^+$ and $a^+ \geq a^{cf}$,

$$a^+ = a^{cf}$$

and this completes the proof of the theorem. □

Further in this paper we use the transitive closure a^{cf} instead of a^+ and, by the theorem 1, algorithm for computing a^{cf} also computes Valiant's transitive closure a^+.

3 RELATED WORKS

Our work is inspired by Valiant [14], who proposed an algorithm for general context-free recognition in less than cubic time. This algorithm computes the same parsing table as the Cocke-Kasami-Younger algorithm [8, 16] but does this by offloading the most intensive computations into calls to a Boolean matrix multiplication procedure. This approach not only provides an asymptotically more efficient algorithm but it also permits to effectively apply GPGPU computing techniques. Valiant's algorithm computes the transitive closure a^+ of a square upper triangular matrix a. Valiant also showed that the matrix multiplication operation is essentially the same as $|N|^2$ Boolean matrix multiplications, where $|N|$ is the number of non-terminals of the given context-free grammar in Chomsky normal form.

Hellings [6] presented an algorithm for context-free path query evaluation using the relational query semantics. According to Hellings, for a given graph $D = (V, E)$ and a grammar $G = (N, \Sigma, P)$ the context-free path query evaluation using the relational query semantics reduces to a calculation of the relations R_A. Thus, in this paper, we focus on the calculation of these context-free relations. Also, Hellings [6] presented an algorithm for context-free path query evaluation using the single-path query semantics that evaluates paths of minimal length for all triples (A, m, n), but also noted that this is not necessary. Thus, in this paper, we evaluate an arbitrary paths for all triples (A, m, n).

Yannakakis [15] analyzed the reducibility of various path querying problems to the calculation of the transitive closure. He formulated a problem of Valiant's technique generalization to the context-free path query evaluation w.r.t. relational query semantics. Also, he assumed that this technique cannot be generalized for arbitrary graphs, though it does for acyclic graphs.

Thus, the possibility of reducing the context-free path query evaluation using the relational and the single-path query semantics to the calculation of the transitive closure is an open problem.

4 CONTEXT-FREE PATH QUERYING BY THE CALCULATION OF TRANSITIVE CLOSURE

In this section, we show how the context-free path query evaluation using the relational query semantics can be reduced to the calculation of matrix transitive closure a^{cf}, prove the correctness of this reduction, introduce an algorithm for computing the transitive closure a^{cf}, and provide a step-by-step demonstration of this algorithm on a small example.

4.1 Reducing context-free path querying to transitive closure

In this section, we show how the context-free relations R_A can be calculated by computing the transitive closure a^{cf}.

Let $G = (N, \Sigma, P)$ be a grammar and $D = (V, E)$ be a graph. We enumerate the nodes of the graph D from 0 to $(|V| - 1)$. We initialize the elements of $|V| \times |V|$ matrix a with $\varnothing$. Further, for every i and j we set

$$a_{i,j} = \{A_k \mid ((i, x, j) \in E) \wedge ((A_k \to x) \in P)\}.$$

Finally, we compute the transitive closure

$$a^{cf} = a^{(1)} \cup a^{(2)} \cup \cdots$$

where

$$a^{(i)} = a^{(i-1)} \cup (a^{(i-1)} \times a^{(i-1)}),$$

for $i \geq 2$ and $a^{(1)} = a$. For the transitive closure a^{cf}, the following statements hold.

Lemma 4.1. *Let $D = (V, E)$ be a graph, let $G = (N, \Sigma, P)$ be a grammar. Then for any i, j and for any non-terminal $A \in N$, $A \in a_{i,j}^{(k)}$ iff $(i, j) \in R_A$ and $i\pi j$, such that there is a derivation tree of the height $h \leq k$ for the string $l(\pi)$ and a context-free grammar $G_A = (N, \Sigma, P, A)$.*

Proof. (Proof by Induction)

Basis: Show that the statement of the lemma holds for $k = 1$. For any i, j and for any non-terminal $A \in N$, $A \in a_{i,j}^{(1)}$ iff there is $i\pi j$ that consists of a unique edge e from node i to node j and $(A \to x) \in P$ where $x = l(\pi)$. Therefore $(i, j) \in R_A$ and there is a derivation tree of the height $h = 1$, shown in Figure 1, for the string x and a context-free grammar $G_A = (N, \Sigma, P, A)$. Thus, it has been shown that the statement of the lemma holds for $k = 1$.

Inductive step: Assume that the statement of the lemma holds for any $k \leq (p - 1)$ and show that it also holds for $k = p$ where $p \geq 2$. For any i, j and for any non-terminal $A \in N$,

$$A \in a_{i,j}^{(p)} \text{ iff } A \in a_{i,j}^{(p-1)} \text{ or } A \in (a^{(p-1)} \times a^{(p-1)})_{i,j},$$

since

$$a^{(p)} = a^{(p-1)} \cup (a^{(p-1)} \times a^{(p-1)}).$$

Let $A \in a_{i,j}^{(p-1)}$. By the inductive hypothesis, $A \in a_{i,j}^{(p-1)}$ iff $(i, j) \in R_A$ and there exists $i\pi j$, such that there is a derivation tree of the height $h \leq (p-1)$ for the string $l(\pi)$ and a context-free grammar $G_A = (N, \Sigma, P, A)$. The statement of the lemma holds

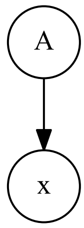

Figure 1: The derivation tree of the height $h = 1$ for the string $x = l(\pi)$.

for $k = p$, since the height h of this tree is also less than or equal to p.

Let $A \in (a^{(p-1)} \times a^{(p-1)})_{i,j}$. By the definition of the binary operation on arbitrary subsets, $A \in (a^{(p-1)} \times a^{(p-1)})_{i,j}$ iff there are r, $B \in a^{(p-1)}_{i,r}$ and $C \in a^{(p-1)}_{r,j}$, such that $(A \to BC) \in P$. Hence, by the inductive hypothesis, there are $i\pi_1 r$ and $r\pi_2 j$, such that $(i,r) \in R_B$ and $(r,j) \in R_C$, and there are the derivation trees T_B and T_C of heights $h_1 \le (p-1)$ and $h_2 \le (p-1)$ for the strings $w_1 = l(\pi_1)$, $w_2 = l(\pi_2)$ and the context-free grammars G_B, G_C respectively. Thus, the concatenation of paths π_1 and π_2 is $i\pi j$, where $(i,j) \in R_A$ and there is a derivation tree of the height $h = 1 + max(h_1, h_2)$, shown in Figure 2, for the string $w = l(\pi)$ and a context-free grammar G_A.

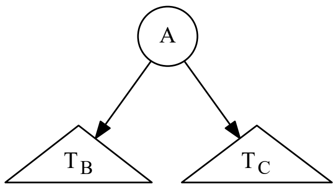

Figure 2: The derivation tree of the height $h = 1 + max(h_1, h_2)$ for the string $w = l(\pi)$, where T_B and T_C are the derivation trees for strings w_1 and w_2 respectively.

The statement of the lemma holds for $k = p$, since the height $h = 1 + max(h_1, h_2) \le p$. This completes the proof of the lemma. □

Theorem 2. *Let $D = (V, E)$ be a graph and let $G = (N, \Sigma, P)$ be a grammar. Then for any i, j and for any non-terminal $A \in N$, $A \in a^{cf}_{i,j}$ iff $(i,j) \in R_A$.*

Proof. Since the matrix $a^{cf} = a^{(1)} \cup a^{(2)} \cup \cdots$, for any i, j and for any non-terminal $A \in N$, $A \in a^{cf}_{i,j}$ iff there is $k \ge 1$, such that $A \in a^{(k)}_{i,j}$. By the lemma 4.1, $A \in a^{(k)}_{i,j}$ iff $(i,j) \in R_A$ and there is $i\pi j$, such that there is a derivation tree of the height $h \le k$ for the string $l(\pi)$ and a context-free grammar $G_A = (N, \Sigma, P, A)$. This completes the proof of the theorem. □

We can, therefore, determine whether $(i,j) \in R_A$ by asking whether $A \in a^{cf}_{i,j}$. Thus, we show how the context-free relations R_A can be calculated by computing the transitive closure a^{cf} of the matrix a.

4.2 The algorithm

In this section we introduce an algorithm for calculating the transitive closure a^{cf} which was discussed in Section 4.1.

Let $D = (V, E)$ be the input graph and $G = (N, \Sigma, P)$ be the input grammar.

Algorithm 1 Context-free recognizer for graphs

1: **function** contextFreePathQuerying(D, G)
2: $n \leftarrow$ a number of nodes in D
3: $E \leftarrow$ the directed edge-relation from D
4: $P \leftarrow$ a set of production rules in G
5: $T \leftarrow$ a matrix $n \times n$ in which each element is $\varnothing$
6: **for all** $(i, x, j) \in E$ **do** ▷ Matrix initialization
7: $T_{i,j} \leftarrow T_{i,j} \cup \{A \mid (A \to x) \in P\}$
8: **while** matrix T is changing **do**
9: $T \leftarrow T \cup (T \times T)$ ▷ Transitive closure T^{cf} calculation
10: **return** T

Note that the matrix initialization in lines **6-7** of the Algorithm 1 can handle arbitrary graph D. For example, if a graph D contains multiple edges (i, x_1, j) and (i, x_2, j) then both the elements of set $\{A \mid (A \to x_1) \in P\}$ and the elements of a set $\{A \mid (A \to x_2) \in P\}$ will be added to $T_{i,j}$.

We need to show that the Algorithm 1 terminates in a finite number of steps. Since each element of the matrix T contains no more than $|N|$ non-terminals, the total number of non-terminals in the matrix T does not exceed $|V|^2|N|$. Therefore, the following theorem holds.

Theorem 3. *Let $D = (V, E)$ be a graph and let $G = (N, \Sigma, P)$ be a grammar. Algorithm 1 terminates in a finite number of steps.*

Proof. It is sufficient to show, that the operation in the line **9** of the Algorithm 1 changes the matrix T only finite number of times. Since this operation can only add non-terminals to some elements of the matrix T, but not remove them, it can change the matrix T no more than $|V|^2|N|$ times. □

Denote the number of elementary operations executed by the algorithm of multiplying two $n \times n$ Boolean matrices as $BMM(n)$. According to Valiant, the matrix multiplication operation in the line **9** of the Algorithm 1 can be calculated in $O(|N|^2 BMM(|V|))$. Denote the number of elementary operations executed by the matrix union operation of two $n \times n$ Boolean matrices as $BMU(n)$. Similarly, it can be shown that the matrix union operation in the line **9** of the Algorithm 1 can be calculated in $O(|N|^2 BMU(n))$. Since the line **9** of the Algorithm 1 is executed no more than $|V|^2|N|$ times, the following theorem holds.

Theorem 4. *Let $D = (V, E)$ be a graph and let $G = (N, \Sigma, P)$ be a grammar. Algorithm 1 calculates the transitive closure T^{cf} in $O(|V|^2|N|^3(BMM(|V|) + BMU(|V|)))$.*

4.3 An example

In this section, we provide a step-by-step demonstration of the proposed algorithm. For this, we consider the classical *same-generation query* [1].

The **example query** is based on the context-free grammar $G = (N, \Sigma, P)$ where:

- A set of non-terminals $N = \{S\}$.
- A set of terminals $\Sigma = \{subClassOf, subClassOf^{-1}, type, type^{-1}\}$.

- A set of production rules P is presented in Figure 3.

$$\begin{array}{rlll} 0: & S & \rightarrow & subClassOf^{-1}\ S\ subClassOf \\ 1: & S & \rightarrow & type^{-1}\ S\ type \\ 2: & S & \rightarrow & subClassOf^{-1}\ subClassOf \\ 3: & S & \rightarrow & type^{-1}\ type \end{array}$$

Figure 3: Production rules for the example query grammar.

Since the proposed algorithm processes only grammars in Chomsky normal form, we first transform the grammar G into an equivalent grammar $G' = (N', \Sigma', P')$ in normal form, where:

- A set of non-terminals $N' = \{S, S_1, S_2, S_3, S_4, S_5, S_6\}$.
- A set of terminals $\Sigma' = \{subClassOf, subClassOf^{-1}, type, type^{-1}\}$.
- A set of production rules P' is presented in Figure 4.

$$\begin{array}{rlll} 0: & S & \rightarrow & S_1\ S_5 \\ 1: & S & \rightarrow & S_3\ S_6 \\ 2: & S & \rightarrow & S_1\ S_2 \\ 3: & S & \rightarrow & S_3\ S_4 \\ 4: & S_5 & \rightarrow & S\ S_2 \\ 5: & S_6 & \rightarrow & S\ S_4 \\ 6: & S_1 & \rightarrow & subClassOf^{-1} \\ 7: & S_2 & \rightarrow & subClassOf \\ 8: & S_3 & \rightarrow & type^{-1} \\ 9: & S_4 & \rightarrow & type \end{array}$$

Figure 4: Production rules for the example query grammar in normal form.

We run the query on a graph presented in Figure 5.

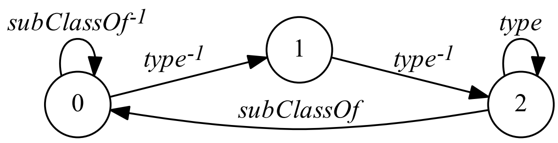

Figure 5: An input graph for the example query.

We provide a step-by-step demonstration of the work with the given graph D and grammar G' of the Algorithm 1. After the matrix initialization in lines **6-7** of the Algorithm 1 we have a matrix T_0 presented in Figure 6.

$$T_0 = \begin{pmatrix} \{S_1\} & \{S_3\} & \varnothing \\ \varnothing & \varnothing & \{S_3\} \\ \{S_2\} & \varnothing & \{S_4\} \end{pmatrix}$$

Figure 6: Initial matrix for the example query.

We denote T_i as a matrix T after i-th loop iteration in lines **8-9** of the Algorithm 1. The calculation of the matrix T_1 is shown in Figure 7.

$$T_0 \times T_0 = \begin{pmatrix} \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \{S\} \\ \varnothing & \varnothing & \varnothing \end{pmatrix}$$

$$T_1 = T_0 \cup (T_0 \times T_0) = \begin{pmatrix} \{S_1\} & \{S_3\} & \varnothing \\ \varnothing & \varnothing & \{S_3, S\} \\ \{S_2\} & \varnothing & \{S_4\} \end{pmatrix}$$

Figure 7: The first iteration of computing the transitive closure for the example query.

When the algorithm at some iteration finds new paths in the graph D, then it adds corresponding nonterminals to the matrix T. For example, after the first loop iteration, non-terminal S is added to the matrix T. This non-terminal is added to the element with a row index $i = 1$ and a column index $j = 2$. This means that there is $i\pi j$ (a path π from node 1 to node 2), such that $S \xrightarrow{*} l(\pi)$. For example, such a path consists of two edges with labels $type^{-1}$ and $type$, and thus $S \xrightarrow{*} type^{-1}\ type$.

The calculation of the transitive closure is completed after k iterations when a fixpoint is reached: $T_{k-1} = T_k$. For the example query, $k = 6$, since $T_6 = T_5$. The remaining iterations of computing the transitive closure are presented in Figure 8.

$$T_2 = \begin{pmatrix} \{S_1\} & \{S_3\} & \varnothing \\ \{S_5\} & \varnothing & \{S_3, S, S_6\} \\ \{S_2\} & \varnothing & \{S_4\} \end{pmatrix}$$

$$T_3 = \begin{pmatrix} \{S_1\} & \{S_3\} & \{S\} \\ \{S_5\} & \varnothing & \{S_3, S, S_6\} \\ \{S_2\} & \varnothing & \{S_4\} \end{pmatrix}$$

$$T_4 = \begin{pmatrix} \{S_1, S_5\} & \{S_3\} & \{S, S_6\} \\ \{S_5\} & \varnothing & \{S_3, S, S_6\} \\ \{S_2\} & \varnothing & \{S_4\} \end{pmatrix}$$

$$T_5 = \begin{pmatrix} \{S_1, S_5, S\} & \{S_3\} & \{S, S_6\} \\ \{S_5\} & \varnothing & \{S_3, S, S_6\} \\ \{S_2\} & \varnothing & \{S_4\} \end{pmatrix}$$

Figure 8: Remaining states of the matrix T.

Thus, the result of the Algorithm 1 for the example query is the matrix $T_5 = T_6$. Now, after constructing the transitive closure, we can construct the context-free relations R_A. These relations for each non-terminal of the grammar G' are presented in Figure 9.

$$\begin{array}{rcl} R_S & = & \{(0,0), (0,2), (1,2)\}, \\ R_{S_1} & = & \{(0,0)\}, \\ R_{S_2} & = & \{(2,0)\}, \\ R_{S_3} & = & \{(0,1), (1,2)\}, \\ R_{S_4} & = & \{(2,2)\}, \\ R_{S_5} & = & \{(0,0), (1,0)\}, \\ R_{S_6} & = & \{(0,2), (1,2)\}. \end{array}$$

Figure 9: Context-free relations for the example query.

By the context-free relation R_S, we can conclude that there are paths in a graph D only from node 0 to node 0, from node 0 to node 2 or from node 1 to node 2, corresponding to the context-free grammar G_S. This conclusion is based on the fact that a grammar G'_S is equivalent to the grammar G_S and $L(G_S) = L(G'_S)$.

5 CONTEXT-FREE PATH QUERYING USING SINGLE-PATH SEMANTICS

In this section, we show how the context-free path query evaluation using the single-path query semantics can be reduced to the calculation of matrix transitive closure a^{cf} and prove the correctness of this reduction.

At the first step, we show how the calculation of matrix transitive closure a^{cf} which was discussed in Section 4.1 can be modified to compute the length of some path $i\pi j$ for all $(i, j) \in R_A$. This is sufficient to solve the problem of context-free path query evaluation using the single-path query semantics, since the path of a fixed length from node i to node j can be found by a simple search.

Let $G = (N, \Sigma, P)$ be a grammar and $D = (V, E)$ be a graph. We enumerate the nodes of the graph D from 0 to $(|V| - 1)$. We initialize $|V| \times |V|$ matrix a with $\varnothing$. We associate each nonterminal in matrix a with the corresponding path length. For convenience, each nonterminal A in the $a_{i,j}$ is represented as a pair (A, k) where k is an associated path length. For every i and j we set

$$a_{i,j} = \{(A_k, 1) \mid ((i, x, j) \in E) \wedge ((A_k \to x) \in P)\},$$

since initially all path lengths are equal to 1. Finally, we compute the transitive closure a^{cf} and if non-terminal A is added to $a^{(p)}_{i,j}$ by using the production rule $(A \to BC) \in P$ where $(B, l_B) \in a^{(p-1)}_{i,k}$, $(C, l_C) \in a^{(p-1)}_{k,j}$, then the path length l_A associated with non-terminal A is calculated as $l_A = l_B + l_C$. Therefore $(A, l_A) \in a^{(p)}_{i,j}$. Note that if some non-terminal A with an associated path length l_1 is in $a^{(p)}_{i,j}$, then the non-terminal A is not added to the $a^{(k)}_{i,j}$ with an associated path length l_2 for all $l_2 \neq l_1$ and $k \geq p$. For the transitive closure a^{cf}, the following statements hold.

Lemma 5.1. *Let $D = (V, E)$ be a graph, let $G = (N, \Sigma, P)$ be a grammar. Then for any i, j and for any non-terminal $A \in N$, if $(A, l_A) \in a^{(k)}_{i,j}$, then $i\pi j$, such that the length of π is equal to l_A.*

Proof. (Proof by Induction)

Basis: Show that the statement of the lemma holds for $k = 1$. For any i, j and for any non-terminal $A \in N$, $(A, l_A) \in a^{(1)}_{i,j}$ iff $l_A = 1$ and there is $i\pi j$ that consists of a unique edge e from node i to node j and $(A \to x) \in P$ where $x = l(\pi)$. Therefore there is $i\pi j$, such that the length of π is equal to l_A. Thus, it has been shown that the statement of the lemma holds for $k = 1$.

Inductive step: Assume that the statement of the lemma holds for any $k \leq (p - 1)$ and show that it also holds for $k = p$ where $p \geq 2$. For any i, j and for any non-terminal $A \in N$, $(A, l_A) \in a^{(p)}_{i,j}$ iff $(A, l_A) \in a^{(p-1)}_{i,j}$ or $(A, l_A) \in (a^{(p-1)} \times a^{(p-1)})_{i,j}$, since $a^{(p)} = a^{(p-1)} \cup (a^{(p-1)} \times a^{(p-1)})$.

Let $(A, l_A) \in a^{(p-1)}_{i,j}$. By the inductive hypothesis, there is $i\pi j$, such that the length of π is equal to l_A. Therefore the statement of the lemma holds for $k = p$.

Let $(A, l_A) \in (a^{(p-1)} \times a^{(p-1)})_{i,j}$. By the definition, $(A, l_A) \in (a^{(p-1)} \times a^{(p-1)})_{i,j}$ iff there are r, $(B, l_B) \in a^{(p-1)}_{i,r}$ and $(C, l_C) \in a^{(p-1)}_{r,j}$, such that $(A \to BC) \in P$ and $l_A = l_B + l_C$. Hence, by the inductive hypothesis, there are $i\pi_1 r$ and $r\pi_2 j$, such that the length of π_1 is equal to l_B and the length of π_2 is equal to l_C. Thus, the concatenation of paths π_1 and π_2 is $i\pi j$, where the length of π is equal to l_A. Therefore the statement of the lemma holds for $k = p$ and this completes the proof of the lemma. □

Theorem 5. *Let $D = (V, E)$ be a graph and let $G = (N, \Sigma, P)$ be a grammar. Then for any i, j and for any non-terminal $A \in N$, $(A, l_A) \in a^{cf}_{i,j}$ iff $(i, j) \in R_A$ and there is $i\pi j$, such that the length of π is equal to l_A.*

Proof. Since the matrix $a^{cf} = a^{(1)} \cup a^{(2)} \cup \cdots$, for any i, j and for any non-terminal $A \in N$, $(A, l_A) \in a^{cf}_{i,j}$ iff there is $k \geq 1$, such that $A \in a^{(k)}_{i,j}$. By the lemma 4.1 and lemma5.1, if $(A, l_A) \in a^{(k)}_{i,j}$, then $(i, j) \in R_A$ and there is $i\pi j$, such that the length of π is equal to l_A. This completes the proof of the theorem. □

We can, therefore, determine whether $(i, j) \in R_A$ and there is $i\pi j$, such that the length of π is equal to l_A, by asking whether $(A, l_A) \in a^{cf}_{i,j}$. Therefore, we can find some path from node i to node j by a simple search. Thus, we show how the context-free path query evaluation using the single-path query semantics can be reduced to the calculation of matrix transitive closure a^{cf}.

6 EVALUATION

To show the practical applicability of the proposed algorithm, we implement this algorithm using different optimizations and apply these implementations to the navigation query problem for a dataset of popular ontologies taken from [17]. We also compare the performance of our implementations with existing analogues from [5, 17]. These analogues use more complex algorithms, while our algorithm uses only simple matrix operations.

Since our algorithm works with graphs, each RDF file from a dataset was converted to an edge-labeled directed graph as follows. For each triple (o, p, s) from a RDF file, we added edges (o, p, s) and (s, p^{-1}, o) to the graph. We also constructed synthetic graphs g_1, g_2 and g_3 by simple repeating the existing graphs.

All tests were run on a PC with the following characteristics:

- OS: Microsoft Windows 10 Pro
- System Type: x64-based PC
- CPU: Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz, 3601 Mhz, 4 Core(s), 4 Logical Processor(s)
- RAM: 16 GB
- GPU: NVIDIA GeForce GTX 1070
  - CUDA Cores: 1920
  - Core clock: 1556 MHz
  - Memory data rate: 8008 MHz
  - Memory interface: 256-bit
  - Memory bandwidth: 256.26 GB/s
  - Dedicated video memory: 8192 MB GDDR5

We denote the implementation of the algorithm from a paper [5] as *GLL*. The algorithm presented in this paper is implemented in F# programming language [13] and is available on GitHub[1]. We denote our implementations of the proposed algorithm as follows:

- dGPU (dense GPU) — an implementation using row-major order for general matrix representation and a GPU for matrix operations calculation. For calculations of matrix

[1]GitHub repository of the YaccConstructor project: https://github.com/YaccConstructor/YaccConstructor.

operations on a GPU, we use a wrapper for the CUBLAS library from the managedCuda[2] library.
- sCPU (sparse CPU) — an implementation using CSR format for sparse matrix representation and a CPU for matrix operations calculation. For sparse matrix representation in CSR format, we use the Math.Net Numerics[3] package.
- sGPU (sparse GPU) — an implementation using the CSR format for sparse matrix representation and a GPU for matrix operations calculation. For calculations of the matrix operations on a GPU, where matrices represented in a CSR format, we use a wrapper for the CUSPARSE library from the managedCuda library.

Since a dense matrix representation leads to a significant performance degradation with the graph size growth, we omit *dGPU* performance on graphs g_1, g_2 and g_3.

We evaluate two classical *same-generation query* [1] which, for example, are applicable in bioinformatics.

Query 1 is based on the grammar G_S^1 for retrieving concepts on the same layer, where:

- A grammar $G^1 = (N^1, \Sigma^1, P^1)$.
- A set of non-terminals $N^1 = \{S\}$.
- A set of terminals $\Sigma^1 = \{subClassOf, subClassOf^{-1}, type, type^{-1}\}$.
- A set of production rules P^1 is presented in Figure 10.

$$\begin{array}{llll} 0: & S & \rightarrow & \mathit{subClassOf}^{-1}\ S\ \mathit{subClassOf} \\ 1: & S & \rightarrow & \mathit{type}^{-1}\ S\ \mathit{type} \\ 2: & S & \rightarrow & \mathit{subClassOf}^{-1}\ \mathit{subClassOf} \\ 3: & S & \rightarrow & \mathit{type}^{-1}\ \mathit{type} \end{array}$$

Figure 10: Production rules for the query 1 grammar.

A grammar G^1 is transformed into an equivalent grammar in normal form, which is necessary for our algorithm. This transformation is the same as in Section 4.3. Let R_S be a context-free relation for a start non-terminal in the transformed grammar.

The result of query 1 evaluation is presented in Table 1, where #triples is a number of triples (o, p, s) in a RDF file, and #results is a number of pairs (n, m) in the context-free relation R_S. We can determine whether $(i, j) \in R_S$ by asking whether $S \in a_{i,j}^{cf}$, where a^{cf} is a transitive closure calculated by the proposed algorithm. All implementations in Table 1 have the same #results and demonstrate up to 1000 times better performance as compared to the algorithm presented in [17] for Q_1. Our implementation *sGPU* demonstrates a better performance than *GLL*. We also can conclude that acceleration from the *GPU* increases with the graph size growth.

Query 2 is based on the grammar G_S^2 for retrieving concepts on the adjacent layers, where:

- A grammar $G^2 = (N^2, \Sigma^2, P^2)$.
- A set of non-terminals $N^2 = \{S, B\}$.
- A set of terminals $\Sigma^2 = \{subClassOf, subClassOf^{-1}\}$.
- A set of production rules P^2 is presented in Figure 11.

A grammar G^2 is transformed into an equivalent grammar in normal form. Let R_S be a context-free relation for a start non-terminal in the transformed grammar.

[2]GitHub repository of the managedCuda library: https://kunzmi.github.io/managedCuda/.
[3]The Math.Net Numerics WebSite: https://numerics.mathdotnet.com/.

$$\begin{array}{llll} 0: & S & \rightarrow & B\ \mathit{subClassOf} \\ 1: & S & \rightarrow & \mathit{subClassOf} \\ 2: & B & \rightarrow & \mathit{subClassOf}^{-1}\ B\ \mathit{subClassOf} \\ 3: & B & \rightarrow & \mathit{subClassOf}^{-1}\ \mathit{subClassOf} \end{array}$$

Figure 11: Production rules for the query 2 grammar.

The result of the query 2 evaluation is presented in Table 2. All implementations in Table 2 have the same #results. On almost all graphs *sGPU* demonstrates a better performance than *GLL* implementation and we also can conclude that acceleration from the *GPU* increases with the graph size growth.

As a result, we conclude that our algorithm can be applied to some real-world problems and it allows us to speed up computations by means of GPGPU.

7 CONCLUSION AND FUTURE WORK

In this paper, we shown how the context-free path query evaluation w.r.t. relational and single-path query semantics can be reduced to the calculation of matrix transitive closure. Also, we provided a formal proof of the correctness of the proposed reduction. In addition, we introduced an algorithm for computing this transitive closure, which allows us to efficiently apply GPGPU computing techniques. Finally, we shown the practical applicability of the proposed algorithm by running different implementations of our algorithm on real-world data.

We can identify several open problems for further research. In this paper we have considered only two semantics of context-free path querying but there are other important semantics, such as *all-path* semantics [7] which requires to present all paths for all triples (A, m, n). Context-free path querying implemented with algorithm [5] can answer the queries in all-path query semantics by constructing a parse forest. It is possible to construct a parse forest for a linear input by matrix multiplication [12]. Whether it is possible to generalize this approach for a graph input is an open question.

In our algorithm, we calculate the matrix transitive closure naively, but there are algorithms for the transitive closure calculation, which are asymptotically more efficient. Therefore, the question is whether it is possible to apply these algorithms for the matrix transitive closure calculation to the problem of context-free path querying.

Also, there are Boolean grammars [11], which have more expressive power than context-free grammars. Boolean path querying is an undecidable problem [6] but our algorithm can be trivially generalized to work on boolean grammars because parsing with boolean grammars can be expressed by matrix multiplication [12]. It is not clear what a result of our algorithm applied to Boolean grammars would look like. Our hypothesis is that it would produce the upper approximation of a solution.

From a practical point of view, matrix multiplication in the main loop of the proposed algorithm may be performed on different GPGPU independently. It can help to utilize the power of multi-GPU systems and increase the performance of context-free path querying.

There is an algorithm [9] for transitive closure calculation on directed graphs which generalized to handle graph sizes inherently larger then the DRAM memory available on the GPU.

Table 1: Evaluation results for Query 1

Ontology	#triples	#results	GLL(ms)	dGPU(ms)	sCPU(ms)	sGPU(ms)
skos	252	810	10	56	14	12
generations	273	2164	19	62	20	13
travel	277	2499	24	69	22	30
univ-bench	293	2540	25	81	25	15
atom-primitive	425	15454	255	190	92	22
biomedical-measure-primitive	459	15156	261	266	113	20
foaf	631	4118	39	154	48	9
people-pets	640	9472	89	392	142	32
funding	1086	17634	212	1410	447	36
wine	1839	66572	819	2047	797	54
pizza	1980	56195	697	1104	430	24
g_1	8688	141072	1926	—	26957	82
g_2	14712	532576	6246	—	46809	185
g_3	15840	449560	7014	—	24967	127

Table 2: Evaluation results for Query 2

Ontology	#triples	#results	GLL(ms)	dGPU(ms)	sCPU(ms)	sGPU(ms)
skos	252	1	1	10	2	1
generations	273	0	1	9	2	0
travel	277	63	1	31	7	10
univ-bench	293	81	11	55	15	9
atom-primitive	425	122	66	36	9	2
biomedical-measure-primitive	459	2871	45	276	91	24
foaf	631	10	2	53	14	3
people-pets	640	37	3	144	38	6
funding	1086	1158	23	1246	344	27
wine	1839	133	8	722	179	6
pizza	1980	1262	29	943	258	23
g_1	8688	9264	167	—	21115	38
g_2	14712	1064	46	—	10874	21
g_3	15840	10096	393	—	15736	40

Therefore, the question is whether it is possible to apply this approach to the matrix transitive closure calculation in the problem of context-free path querying.

ACKNOWLEDGMENTS

We are grateful to Dmitri Boulytchev, Ekaterina Verbitskaia, Marina Polubelova, Dmitrii Kosarev and Dmitry Koznov for their careful reading, pointing out some mistakes, and invaluable suggestions. This work is supported by grant from JetBrains Research.

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. [n. d.]. Foundations of Databases, 1995. ([n. d.]).

[2] James WJ Anderson, Ádám Novák, Zsuzsanna Sükösd, Michael Golden, Preeti Arunapuram, Ingolfur Edvardsson, and Jotun Hein. 2013. Quantifying variances in comparative RNA secondary structure prediction. *BMC bioinformatics* 14, 1 (2013), 149.

[3] Shuai Che, Bradford M Beckmann, and Steven K Reinhardt. 2016. Programming GPGPU Graph Applications with Linear Algebra Building Blocks. *International Journal of Parallel Programming* (2016), 1–23.

[4] Noam Chomsky. 1959. On certain formal properties of grammars. *Information and control* 2, 2 (1959), 137–167.

[5] Semyon Grigorev and Anastasiya Ragozina. 2016. Context-Free Path Querying with Structural Representation of Result. *arXiv preprint arXiv:1612.08872* (2016).

[6] J. Hellings. 2014. Conjunctive context-free path queries. (2014).

[7] Jelle Hellings. 2015. Querying for Paths in Graphs using Context-Free Path Queries. *arXiv preprint arXiv:1502.02242* (2015).

[8] Tadao Kasami. 1965. *AN EFFICIENT RECOGNITION AND SYNTAXANALYSIS ALGORITHM FOR CONTEXT-FREE LANGUAGES.* Technical Report. DTIC Document.

[9] Gary J Katz and Joseph T Kider Jr. 2008. All-pairs shortest-paths for large graphs on the GPU. In *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*. Eurographics Association, 47–55.

[10] A. Mendelzon and P. Wood. 1995. Finding Regular Simple Paths in Graph Databases. *SIAM J. Computing* 24, 6 (1995), 1235–1258.

[11] Alexander Okhotin. 2004. Boolean grammars. *Information and Computation* 194, 1 (2004), 19–48.

[12] Alexander Okhotin. 2014. Parsing by matrix multiplication generalized to Boolean grammars. *Theoretical Computer Science* 516 (2014), 101–120.

[13] Don Syme, Adam Granicz, and Antonio Cisternino. 2012. *Expert F# 3.0.* Springer.

[14] Leslie G Valiant. 1975. General context-free recognition in less than cubic time. *Journal of computer and system sciences* 10, 2 (1975), 308–315.

[15] Mihalis Yannakakis. 1990. Graph-theoretic methods in database theory. In *Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. ACM, 230–242.

[16] Daniel H Younger. 1967. Recognition and parsing of context-free languages in time n3. *Information and control* 10, 2 (1967), 189–208.

[17] X. Zhang, Z. Feng, X. Wang, G. Rao, and W. Wu. 2016. Context-free path queries on RDF graphs. In *International Semantic Web Conference*. Springer, 632–648.