Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Санкт-Петербургский государственный университет»

Математическое обеспечение и администрирование
информационных систем

Системное Программирование

Азимов Рустам Шухратуллович

Синтаксический анализ графов через умножение матриц

Магистерская диссертация

Допущена к защите.
Зав. кафедрой:
д. ф.-м. н., профессор Терехов А. Н.

Научный руководитель:
к. ф.-м. н., доцент кафедры информатики Григорьев С. В.

Рецензент:
преподаватель, кафедра информатики, Академия Або (Финляндия) Бараш М. Л.

Санкт-Петербург
2018

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems

Software Engineering

Rustam Azimov

Graph parsing by matrix multiplication

Master's Thesis

Admitted for defence.
Head of the chair:
Professor Andrey Terekhov

Scientific supervisor:
Assistant Professor Semyon Grigorev

Reviewer:
University Teacher, Department of Computer Science, Åbo Akademi University (Finland)
Mikhail Barash

Saint-Petersburg
2018

Contents

Introduction

Graph data models are widely used in many areas, for example, graph databases [18], bioinformatics [24], static analysis [10, 17], etc. In these areas, it is often required to process queries for large graphs. The most common type of graph queries is a navigational query. The result of a query evaluation is a set of implicit relations between the nodes of the graph, i.e. a set of paths. A natural way to specify these relations is to specify the paths using some form of formal grammars (regular expressions, context-free grammars) over the alphabet of edge labels. Context-free grammars are actively used in graph querying because of the limited expressive power of regular expressions. For example, classical *same-generation queries* [1] cannot be expressed using regular expressions.

The result of a context-free path query (CFPQ) evaluation is usually a set of triples (A, m, n), such that there is a path from the node m to the 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* [13]. Another example of path query semantics is the *single-path query semantics* [14] which requires presenting a single path from the node m to the node n whose labeling is derived from a non-terminal A for all triples (A, m, n) evaluated using the relational query semantics. There is a number of algorithms for CFPQ evaluation using these semantics [9, 11, 13, 14, 28].

The existing algorithms for CFPQ evaluation w.r.t. these semantics demonstrate a poor performance when applied to large graphs. The algorithms for context-free language recognition had a similar problem until Valiant [31] proposed a parsing algorithm, which computes a recognition table by computing matrix transitive closure. The algorithm works for a linear input and has the complexity, which is essentially the same as for Boolean matrix multiplication. One of the hard open problems is to generalize Valiant's matrix-based approach for context-free path query evaluation.

Proposing the first matrix-based algorithm for CFPQ evaluation using these semantics makes it possible to efficiently apply such computing techniques as *GPGPU* (General-Purpose computing on Graphics Processing

Units) and parallel computation [6]. From a practical point of view, matrix multiplication can be performed on different GPUs independently. It can help to utilize the power of multi-GPU systems and increase the performance of context-free path querying. Also, the algorithms for distributed-memory matrix multiplication make it possible to handle graph sizes inherently larger, than the memory available on the GPU [7, 29, 35].

Also, there are conjunctive grammars [22], which have more expressive power than context-free grammars. Path querying with conjunctive grammars is known to be undecidable [13]. Although there is an algorithm [36] for path querying with linear conjunctive grammars [22] which is used in static analysis and provides an over-approximation of the result. However, there is no algorithm for path querying with conjunctive grammars of an arbitrary form.

1 Problem statement

The purpose of this work is to develop an effective matrix-based algorithm for path querying with context-free and conjunctive grammars, for which it is required to solve the problems listed below.

- Introduce a matrix-based algorithm for context-free path query evaluation w.r.t. relational query semantics.
- Introduce a matrix-based algorithm for context-free path query evaluation w.r.t. single-path query semantics.
- Introduce a matrix-based algorithm for path query evaluation w.r.t. conjunctive grammars and relational query semantics.
- Show the practical applicability of our algorithms by presenting the results of their evaluation on a set of conventional benchmarks.

2 Background

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

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 the node $n \in V$ and ends at the node $m \in V$.

Following Hellings [13], we deviate from the usual definition of a context-free grammar in *Chomsky Normal Form* [8] 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 [31], 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 need to compute the transitive closure of an arbitrary square matrix.

Also, the conjunctive grammars can be used in the path querying problems. Similar to the case of the context-free grammars, we deviate from the usual definition of a conjunctive grammar in the *binary normal form* [22] by not including a special start non-terminal, which will be specified in the queries to the graph. Since every conjunctive grammar can be transformed into an equivalent one in the binary normal form [22] and checking that an empty string is in the language is trivial, then it is sufficient to only

consider grammars of the following type. A *conjunctive grammar* is 3-tuple $G = (N, \Sigma, P)$ 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:

- $A \to B_1C_1 \,\&\, \ldots \,\&\, B_mC_m$, for $m \geq 1$, $A, B_i, C_i \in N$,
- $A \to x$, for $A \in N$ and $x \in \Sigma$.

For conjunctive grammars we also use the conventional notation $A \overset{*}{\to} w$ to denote that the string $w \in \Sigma^*$ can be derived from a non-terminal A by some sequence of applying the production rules from P. The relation $\to$ is defined as follows:

- Using a rule $A \to B_1C_1 \,\&\, \ldots \,\&\, B_mC_m \in P$, any atomic subterm A of any term can be rewritten by the subterm $(B_1C_1 \,\&\, \ldots \,\&\, B_mC_m)$:

$$\ldots A \ldots \to \ldots (B_1C_1 \,\&\, \ldots \,\&\, B_mC_m) \ldots$$

- A conjunction of several identical strings in Σ^* can be rewritten by one such string: for every $w \in \Sigma^*$,

$$\ldots (w \,\&\, \ldots \,\&\, w) \ldots \to \ldots w \ldots$$

The *language* of a conjunctive grammar $G = (N, \Sigma, P)$ with respect to a start non-terminal $S \in N$ is defined by $L(G_S) = \{w \in \Sigma^* \mid S \overset{*}{\to} w\}$.

For a given graph $D = (V, E)$ and a conjunctive grammar $G = (N, \Sigma, P)$, we define *conjunctive relations* $R_A \subseteq V \times V$, for every $A \in N$, such that $R_A = \{(n, m) \mid \exists n\pi m \; (l(\pi) \in L(G_A))\}$.

3 Related works

Problems in many areas can be reduced to one of the formal-languages-constrained path problems [4]. For example, various problems of static code analysis [5, 32] can be formulated in terms of the context-free language reachability [25] or in terms of the linear conjunctive language reachability [36].

One of the well-known problems in the area of graph database analysis is the language-constrained path querying. For example, the regular language constrained path querying [26, 3, 2, 19], and the context-free language constrained path querying.

There are a number of solutions [13, 28, 9] for context-free path query evaluation w.r.t. the relational query semantics, which employ such parsing algorithms as CYK [15, 34] or Earley [12]. Other examples of path query semantics are single-path and *all-path query semantics*. The all-path query semantics requires presenting all possible paths from node m to node n whose labeling is derived from a non-terminal A for all triples (A, m, n) evaluated using the relational query semantics. Hellings [14] presented algorithms for the context-free path query evaluation using the single-path and the all-path query semantics. If a context-free path query w.r.t. the all-path query semantics is evaluated on cyclic graphs, then the query result can be an infinite set of paths. For this reason, in [14], annotated grammars are proposed as a possible solution.

In [11], the algorithm for context-free path query evaluation w.r.t. the all-path query semantics is proposed. This algorithm is based on the generalized top-down parsing algorithm — GLL [27]. This solution uses derivation trees for the result representation which is more native for grammar-based analysis. The algorithms in [11, 14] for the context-free path query evaluation w.r.t. the all-path query semantics can also be used for query evaluation using the relational and the single-path semantics.

Our work is inspired by Valiant [31], who proposed an algorithm for general context-free recognition in less than cubic time. This algorithm computes the same parsing table as the CYK algorithm 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 allows us 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 ($\times$) 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 [13] presented an algorithm for the 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 w.r.t. the relational query semantics reduces to a calculation of the context-free relations R_A. Thus, in this work, we focus on the calculation of these context-free relations. Also, Hellings [13] presented an algorithm for the context-free path query evaluation using the single-path query semantics which evaluates paths of minimal length for all triples (A, m, n), but also noted that the length of these paths is not necessarily upper bounded. Thus, in this work, we evaluate an arbitrary path for all triples (A, m, n).

Yannakakis [33] 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. the 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.

Also, there is an algorithm [36] for path querying with linear conjunctive grammars and relational query semantics. This grammars have no more than one nonterminal in each conjunct of the rule. Thus, the possibility of creating an algorithm for path query evaluation w.r.t. conjunctive grammars of an arbitrary form is an open problem.

4 An algorithm for CFPQ using relational query semantics

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 CFPQ to transitive closure

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

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 that Valiant's and this definitions of the matrix transitive closure are equivalent, we introduce the partial order $\succeq$ on matrices with the fixed size which have subsets of N as elements. For square matrices a, b of the same size, we denote $a \succeq b$ 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 1 *Let $G = (N, \Sigma, P)$ be a grammar, let a be a square matrix. Then $a^{(k)} \succeq a_+^{(k)}$ 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

$i \geq 2$

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

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

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

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

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

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

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

and from this it follows that

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

By the definition,

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

and this completes the proof of the lemma. □

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

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

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

By the definition,

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

and from this it follows that

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

The following holds

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

since

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

and

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

Therefore there is $2j$, such that

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

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 1, for any $k \geq 1$, $a^{(k)} \preceq a_+^{(k)}$. Therefore

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

By the lemma 2, for any $k \geq 1$ there is $j \geq 1$, such that

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

Hence

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

for any $k \geq 1$. Therefore

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

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

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

and this completes the proof of the theorem. □

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

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 the $|V| \times |V|$ 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 3 *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 the node i to the 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 $k = 1$.

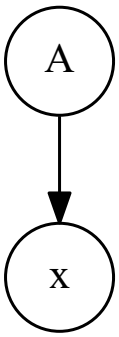

Figure 1: The derivation tree of the height $h = 1$ for the string $x = l(\pi)$.

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 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 $(\cdot)$ on arbitrary subsets, $A \in (a^{(p-1)} \times a^{(p-1)})_{i,j}$ iff there are r, $B \in a_{i,r}^{(p-1)}$ and $C \in a_{r,j}^{(p-1)}$, 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 $h_1 \leq (p-1)$ and $h_2 \leq (p-1)$ 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.

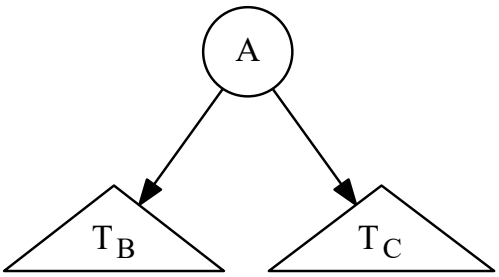

Figure 2: The derivation tree of the height $h = 1+max(h_1, h_2)$ for the string $w = l(\pi)$, where T_B and T_C are the derivation trees for strings w_1 and w_2 respectively.

The statement of the lemma holds for $k = p$ since the height $h = 1 + max(h_1, h_2) \leq p$. 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 $A \in N$, $A \in a_{i,j}^{cf}$ 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 $A \in N$, $A \in a_{i,j}^{cf}$ iff there is $k \geq 1$, such that $A \in a_{i,j}^{(k)}$. By the lemma 3, $A \in a_{i,j}^{(k)}$ iff $(i,j) \in R_A$ and there is $i\pi j$, such that there is a derivation tree of the height $h \leq k$ for the string $l(\pi)$ and a context-free grammar $G_A = (N, \Sigma, P, A)$. This completes the proof of the theorem. □

We can, therefore, determine whether $(i,j) \in R_A$ by asking whether $A \in a_{i,j}^{cf}$. 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 ← the number of nodes in D
3:      E ← the directed edge-relation from D
4:      P ← the set of production rules in G
5:      T ← the matrix n × n in which each element is ∅
6:      for all (i, x, j) ∈ E do                    ▷ Matrix initialization
7:          T_{i,j} ← T_{i,j} ∪ {A | (A → x) ∈ P}
8:      while matrix T is changing do
9:          T ← T ∪ (T × 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 the set $\{A \mid (A \to x_1) \in P\}$ and the elements of the 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. The 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. $\square$

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. The Algorithm 1 calculates the transitive closure T^{cf} in $O(|V|^2|N|^3(BMM(|V|)+ BMU(|V|)))$.*

We also provide the worst-case example, for which the time complexity in terms of the graph size provided by Theorem 4 cannot be improved. This example is based on the context-free grammar $G = (N, \Sigma, P)$ where:

- the set of non-terminals $N = \{S\}$;
- the set of terminals $\Sigma = \{a, b\}$;
- the set of production rules P is presented on Figure 3.

$$\begin{array}{lrcl} 0: & S & \rightarrow & a\ S\ b \\ 1: & S & \rightarrow & a\ b \end{array}$$

Figure 3: Production rules for the worst-case example.

Let the size $|N|$ of the grammar G be a constant. The worst-case time complexity is reached by running this query on the double-cyclic graph where:

- one of the cycles having $u = 2^k + 1$ edges labeled with a;

- another cycle having $v = 2^k$ edges labeled with b;
- the two cycles are connected via a shared node m.

A small example of such graph with $k = 1$, $u = 3$, $v = 2$, and $m = 0$ is presented on Figure 4.

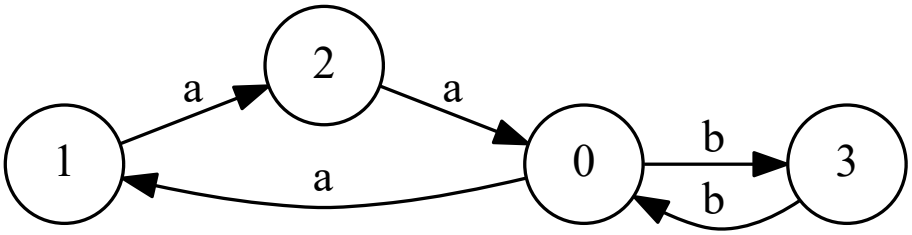

Figure 4: An example of the graph for the worst-case time complexity.

The shortest path π from the node m to the node m, whose labeling forms a string from the language $L(G_S) = \{a^n b^n; n \geq 1\}$, has a length $l = 2 * u * v$, since $u = 2^k + 1$ and $v = 2^k$ are coprime, and string s, formed by this path, consists of $u * v$ labels a and $u * v$ labels b. The string $s = l(\pi)$ has a derivation tree according to a context-free grammar G_S of the minimal height $h = 2 * u * v$ among all the paths from the node m to the node m in this double-cyclic graph. Therefore, if we run the worst-case example query on this graph, then the operation in the line **9** of the Algorithm 1 changes the matrix T at least $h = 2 * u * v$ times. Hence, the Algorithm 1 computes this query in $O(|V|^2(BMM(|V|) + BMU(|V|)))$, since $|V| = (u + v - 1) = 2 * v$ and $h = 2 * u * v > 2 * v * v = |V|^2/4 = O(|V|^2)$.

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:

- The set of non-terminals $N = \{S\}$.

- The set of terminals

$$\Sigma = \{subClassOf, subClassOf^{-1}, type, type^{-1}\}.$$

- The set of production rules P is presented in Figure 5.

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

Figure 5: 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:

- The set of non-terminals $N' = \{S, S_1, S_2, S_3, S_4, S_5, S_6\}$.

- The set of terminals

$$\Sigma' = \{subClassOf, subClassOf^{-1}, type, type^{-1}\}.$$

- The set of production rules P' is presented in Figure 6.

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

Figure 6: Production rules for the example query grammar in normal form.

We run the query on a graph presented in Figure 7.

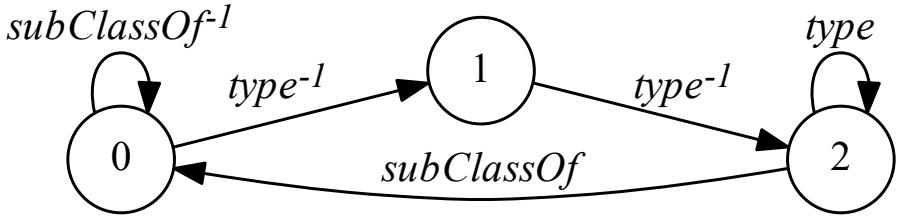

Figure 7: An input graph for the example query.

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 8.

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

Figure 8: The initial matrix for the example query.

Let T_i be the matrix T obtained after executing the loop in lines **8-9** of the Algorithm 1 i times. The calculation of the matrix T_1 is shown in Figure 9.

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

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

Figure 9: 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 the node 1 to

the 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 10.

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

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

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

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

Figure 10: 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 11.

By the context-free relation R_S, we can conclude that there are paths in a graph D only from the node 0 to the node 0, from the node 0 to the node 2 or from the node 1 to the 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 $L(G_S) = L(G'_S)$.

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

Figure 11: Context-free relations for the example query.

5 CFPQ using single-path query 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 $(i,j) \in R_A$, such that $A \xrightarrow{*} l(\pi)$. This is sufficient to solve the problem of context-free path query evaluation using the single-path query semantics since the required path of a fixed length from the node i to the node j can be found by a simple search and checking whether the labels of this path form a string which can be derived from a non-terminal A.

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 $|V| \times |V|$ matrix a with $\varnothing$. We associate each non-terminal 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 $a_{i,j}^{(p)}$ by using the

production rule $(A \rightarrow BC) \in P$ where $(B, l_B) \in a_{i,k}^{(p-1)}$, $(C, l_C) \in a_{k,j}^{(p-1)}$, then the path length l_A associated with non-terminal A is calculated as $l_A = l_B + l_C$. Therefore $(A, l_A) \in a_{i,j}^{(p)}$. Note that if some non-terminal A with an associated path length l_1 is in $a_{i,j}^{(p)}$, then the non-terminal A is not added to the $a_{i,j}^{(k)}$ 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 4 *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 $(A, l_A) \in a_{i,j}^{(k)}$, then there is $i\pi j$, such that $A \stackrel{*}{\rightarrow} l(\pi)$ and 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 $A \in N$, $(A, l_A) \in a_{i,j}^{(1)}$ iff $l_A = 1$ and there is $i\pi j$ that consists of a unique edge e from the node i to the node j and $(A \rightarrow x) \in P$ where $x = l(\pi)$. Therefore there is $i\pi j$, such that $A \stackrel{*}{\rightarrow} l(\pi)$ and 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 $A \in N$, $(A, l_A) \in a_{i,j}^{(p)}$ iff $(A, l_A) \in a_{i,j}^{(p-1)}$ 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 $(A, l_A) \in a_{i,j}^{(p-1)}$. By the inductive hypothesis, there is $i\pi j$, such that $A \stackrel{*}{\rightarrow} l(\pi)$ and 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, $(B, l_B) \in a_{i,r}^{(p-1)}$ and $(C, l_C) \in a_{r,j}^{(p-1)}$, such that $(A \rightarrow BC) \in P$ 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

$$(B \stackrel{*}{\rightarrow} l(\pi_1)) \wedge (C \stackrel{*}{\rightarrow} l(\pi_2)),$$

where 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 $A \stackrel{*}{\rightarrow} l(\pi)$ and 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 $A \in N$, if $(A, l_A) \in a^{cf}_{i,j}$, then there is $i\pi j$, such that $A \xrightarrow{*} l(\pi)$ and 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 $A \in N$, if $(A, l_A) \in a^{cf}_{i,j}$, then there is $k \geq 1$, such that $A \in a^{(k)}_{i,j}$. By the lemma 4, if $(A, l_A) \in a^{(k)}_{i,j}$, then there is $i\pi j$, such that $A \xrightarrow{*} l(\pi)$ and the length of π is equal to l_A. This completes the proof of the theorem. □

By the theorem 2, we can determine whether $(i, j) \in R_A$ by asking whether $(A, l_A) \in a^{cf}_{i,j}$ for some l_A. By the theorem 5, there is $i\pi j$, such that $A \xrightarrow{*} l(\pi)$ and the length of π is equal to l_A. Therefore, we can find such a path π of the length l_A from the node i to the 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}. Note that the time complexity of the algorithm for context-free path querying w.r.t. the single-path semantics no longer depends on the Boolean matrix multiplications since we modify the matrix representation and operations on the matrix elements.

6 A path querying algorithm using conjunctive grammars

In this section, we show how the path querying using conjunctive grammars and relational query semantics can be reduced to the calculation of the matrix transitive closure. We propose an algorithm that calculates the over-approximation of all conjunctive relations R_A, since the query evaluation using the relational query semantics and conjunctive grammars is undecidable problem [13].

We define a *conjunctive matrix multiplication*, $a \circ b = c$, where a and b are matrices of the suitable size that have subsets of N as elements, as $c_{i,j} = \{A \mid \exists(A \to B_1C_1 \ \& \ldots \& \ B_mC_m) \in P \text{ such that } (B_k, C_k) \in d_{i,j}\}$, where $d_{i,j} = \bigcup_{k=1}^{n} a_{i,k} \times b_{k,j}$ and $(\times)$ is the Cartesian product.

We define the *conjunctive transitive closure* of a square matrix a as $a^{conj} = a^{(1)} \cup a^{(2)} \cup \cdots$ where $a^{(i)} = a^{(i-1)} \cup (a^{(i-1)} \circ a^{(i-1)})$, $i \geq 2$ and $a^{(1)} = a$.

6.1 Reducing conjunctive path querying to transitive closure

In this section, we show how the over-approximation of all conjunctive relations R_A can be calculated by computing the transitive closure a^{conj}.

Let $G = (N, \Sigma, P)$ be a conjunctive grammar and $D = (V, E)$ be a graph. We number the nodes of the graph D from 0 to $(|V|-1)$ and we associate the nodes with their numbers. We initialize $|V| \times |V|$ matrix b with $\emptyset$. Further, for every i and j we set $b_{i,j} = \{A_k \mid ((i, x, j) \in E) \wedge ((A_k \to x) \in P)\}$. Finally, we compute the conjunctive transitive closure $b^{conj} = b^{(1)} \cup b^{(2)} \cup \cdots$ where $b^{(i)} = b^{(i-1)} \cup (b^{(i-1)} \circ b^{(i-1)})$, $i \geq 2$ and $b^{(1)} = b$. For the conjunctive transitive closure b^{conj}, the following statements holds.

Lemma 5 *Let $D = (V, E)$ be a graph, let $G = (N, \Sigma, P)$ be a conjunctive grammar. Then for any i, j and for any non-terminal $A \in N$, if $(i, j) \in R_A$ and $i\pi j$, such that there is a derivation tree according to the string $l(\pi)$ and a conjunctive grammar $G_A = (N, \Sigma, P, A)$ of the height $h \leq k$ then $A \in b_{i,j}^{(k)}$.*

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 $A \in N$, if $(i, j) \in R_A$ and $i\pi j$, such that there is a derivation tree according to the string $l(\pi)$ and a conjunctive grammar $G_A = (N, \Sigma, P, A)$ of the height $h \leq 1$ then there is edge e from node i to node j and $(A \to x) \in P$ where $x = l(\pi)$. Therefore $A \in b_{i,j}^{(1)}$ and 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 $p \geq 2$. Let $(i, j) \in R_A$ and $i\pi j$, such that there is a derivation tree according to the string $l(\pi)$ and a conjunctive grammar $G_A = (N, \Sigma, P, A)$ of the height $h \leq p$.

Let $h < p$. Then by the inductive hypothesis $A \in b_{i,j}^{(p-1)}$. Since $b^{(p)} = b^{(p-1)} \cup (b^{(p-1)} \circ b^{(p-1)})$ then $A \in b_{i,j}^{(p)}$ and the statement of the lemma holds for $k = p$.

Let $h = p$. Let $A \to B_1C_1 \,\&\, \ldots \,\&\, B_mC_m$ be the rule corresponding to the root of the derivation tree from the assumption of the lemma. Therefore the heights of all subtrees corresponding to non-terminals $B_1, C_1, \ldots B_m, C_m$ are less than p. Then by the inductive hypothesis $B_x \in b_{i,t_x}^{(p-1)}$ and $C_x \in b_{t_x,j}^{(p-1)}$, for $x = 1 \ldots m$ and $t_x \in V$. Let d be a matrix that have subsets of $N \times N$ as elements, where $d_{i,j} = \bigcup_{t=1}^{n} b_{i,t}^{(p-1)} \times b_{t,j}^{(p-1)}$. Therefore $(B_x, C_x) \in d_{i,j}$, for $x = 1 \ldots m$. Since $b^{(p)} = b^{(p-1)} \cup (b^{(p-1)} \circ b^{(p-1)})$ and $(b^{(p-1)} \circ b^{(p-1)})_{i,j} = \{A \mid \exists (A \to B_1C_1 \,\&\, \ldots \,\&\, B_mC_m) \in P \text{ such that } (B_k, C_k) \in d_{i,j}\}$ then $A \in b_{i,j}^{(p)}$ and the statement of the lemma holds for $k = p$. This completes the proof of the lemma. □

Theorem 6 *Let $D = (V, E)$ be a graph and let $G = (N, \Sigma, P)$ be a conjunctive grammar. Then for any i, j and for any non-terminal $A \in N$, if $(i, j) \in R_A$ then $A \in b_{i,j}^{conj}$.*

Proof: By the lemma 5, if $(i, j) \in R_A$ then $A \in b_{i,j}^{(k)}$ for some k, such that $i\pi j$ with a derivation tree according to the string $l(\pi)$ and a conjunctive grammar $G_A = (N, \Sigma, P, A)$ of the height $h \leq k$. Since the matrix $b^{conj} = b^{(1)} \cup b^{(2)} \cup \cdots$, then for any i, j and for any non-terminal $A \in N$, if $A \in b_{i,j}^{(k)}$

for some $k \geq 1$ then $A \in b_{i,j}^{conj}$. Therefore, if $(i,j) \in R_A$ then $A \in b_{i,j}^{conj}$. This completes the proof of the theorem. □

Thus, we show how the over-approximation of all conjunctive relations R_A can be calculated by computing the conjunctive transitive closure b^{conj} of the matrix b.

6.2 The algorithm

In this section we introduce an algorithm for calculating the conjunctive transitive closure b^{conj} which was discussed in Section 6.1.

The following algorithm takes on input a graph $D = (V, E)$ and a conjunctive grammar $G = (N, \Sigma, P)$.

Algorithm 2 Conjunctive recognizer for graphs

```
1:  function conjunctiveGraphParsing(D, G)
2:      n ← a number of nodes in D
3:      E ← the directed edge-relation from D
4:      P ← a set of production rules in G
5:      T ← a matrix n × n in which each element is ∅
6:      for all (i, x, j) ∈ E do                    ▷ Matrix initialization
7:          T_{i,j} ← T_{i,j} ∪ {A | (A → x) ∈ P}
8:      while matrix T is changing do
9:          T ← T ∪ (T ∘ T)                         ▷ Transitive closure calculation
10:     return T
```

Similar to the case of the context-free grammars we can show that the Algorithm 2 terminates in a finite number of steps. Since each element of the matrix T contains no more than $|N|$ non-terminals, then total number of non-terminals in the matrix T does not exceed $|V|^2|N|$. Therefore, the following theorem holds.

Theorem 7 *Let $D = (V, E)$ be a graph and let $G = (N, \Sigma, P)$ be a conjunctive grammar. Algorithm 2 terminates in a finite number of steps.*

Proof: It is sufficient to show, that the operation in line **9** of the Algorithm 2 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. □

6.3 An example

In this section, we provide a step-by-step demonstration of the proposed algorithm for path querying using conjunctive grammars. The **example query** is based on the conjunctive grammar $G = (N, \Sigma, P)$ in binary normal form where:

- The set of non-terminals $N = \{S, A, B, C, D\}$.
- The set of terminals $\Sigma = \{a, b, c\}$.
- The set of production rules P is presented in Figure 12.

$$
\begin{array}{lrcl}
0: & S & \to & AB \ \& \ DC \\
1: & A & \to & a \\
2: & B & \to & BC \\
3: & B & \to & b \\
4: & C & \to & c \\
5: & D & \to & AD \\
6: & D & \to & b
\end{array}
$$

Figure 12: Production rules for the conjunctive example query grammar.

The conjunct AB generates the language $L_{AB} = \{abc^*\}$ and the conjunct DC generates the language $L_{DC} = \{a^*bc\}$. Thus, the language generated by the conjunctive grammar $G_S = (N, \Sigma, P, S)$ is $L(G_S) = L_{AB} \cap L_{DC} = \{abc\}$. We tun the query on a graph presented in Figure 13.

We provide a step-by-step demonstration of the work with the given graph D and grammar G of the Algorithm 2. After the matrix initialization in lines **6-7** of the Algorithm 2, we have a matrix T_0 presented in Figure 14.

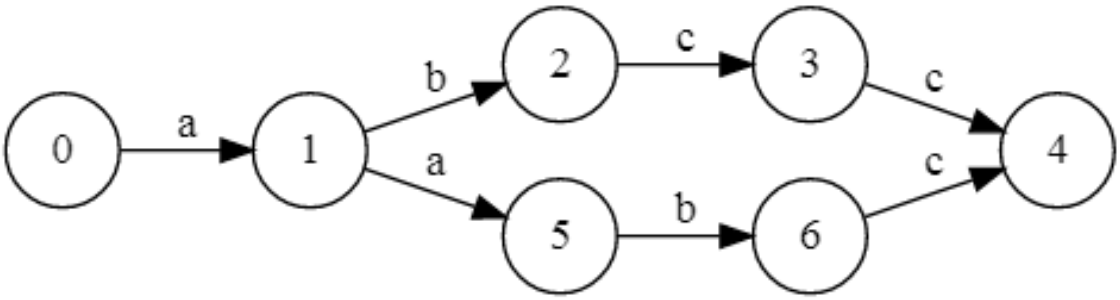

Figure 13: An input graph for the conjunctive example query.

$$T_0 = \begin{pmatrix} \varnothing & \{A\} & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \{B,D\} & \varnothing & \varnothing & \{A\} & \varnothing \\ \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \{B,D\} \\ \varnothing & \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing \end{pmatrix}$$

Figure 14: The initial matrix for the example query.

Let T_i be the matrix T obtained after executing the loop in lines **8-9** of the Algorithm 2 i times. To compute the matrix T_1 we need to compute the matrix d where $d_{i,j} = \bigcup_{k=1}^{n} T_{0_{i,k}} \times T_{0_{i,k}}$. The matrix d for the first loop iteration is presented in Figure 15. The matrix $T_1 = T_1 = T_0 \cup (T_0 \circ T_0)$ is shown in Figure 16.

$$\begin{pmatrix} \varnothing & \varnothing & \{(A,B),(A,D)\} & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \{(B,C),(D,C)\} & \varnothing & \varnothing & \{(A,B),(A,D)\} \\ \varnothing & \varnothing & \varnothing & \varnothing & \{(C,C)\} & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{(B,C),(D,C)\} & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

Figure 15: The matrix d for the first loop iteration.

When the algorithm at some iteration finds new paths from the node i to the node j for all conjuncts of some production rule, then it adds nonterminal from the left side of this rule to the set $T_{i,j}$.

$$T_1 = \begin{pmatrix} \varnothing & \{A\} & \{D\} & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \{B,D\} & \{B\} & \varnothing & \{A\} & \{D\} \\ \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{B\} & \varnothing & \{B,D\} \\ \varnothing & \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing \end{pmatrix}$$

Figure 16: The initial matrix for the example query.

The calculation of the transitive closure is completed after k iterations when a fixpoint is reached: $T_{k-1} = T_k$. For this example, $k = 4$ since $T_4 = T_3$. The remaining iterations of computing the transitive closure are presented in Figure 17.

Thus, the result of the Algorithm 2 for the example query is the matrix $T_4 = T_3$. Now, after constructing the transitive closure, we can construct the over-approximations R'_A of the conjunctive relations R_A. These approximations for each non-terminal of the grammar G are presented in Figure 18.

This example demonstrates that it is not always possible to obtain an exact solution. For example, a pair of nodes $(0, 4)$ belongs to R'_S, although there is no path from the node 0 to the node 4, which forms a string derived from the nonterminal S (only the string abc can be derived from the nonterminal S). Extra pairs of nodes are added if there are different paths from the node i to the node j, which in summary correspond to all conjuncts of one production rule, but there is no path from the node i to the node j, which at the same time would correspond to all conjuncts of this rule. For example, for the conjuncts of the rule $S \to AB \;\&\; DC$, there is a path from the node 0 to the node 4 forming the string $abcc$, and there is also a path from the node 0 to the node 4 forming the string $aabc$. The first path corresponds to the conjunct AB, since the string $abcc$ belongs to the language $L_{AB} = \{abc^*\}$, and the second path corresponds to the conjunct DC, since the string $aabc$ belongs to the language $L_{DC} = \{a^*bc\}$. However, it is obvious that there is no path from the node 0 to the node 4, which forms the string abc.

$$T_2 = \begin{pmatrix} \varnothing & \{A\} & \{D\} & \{S\} & \varnothing & \varnothing & \{D\} \\ \varnothing & \varnothing & \{B,D\} & \{B\} & \{S,B\} & \{A\} & \{D\} \\ \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{B\} & \varnothing & \{B,D\} \\ \varnothing & \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing \end{pmatrix}$$

$$T_3 = \begin{pmatrix} \varnothing & \{A\} & \{D\} & \{S\} & \{S\} & \varnothing & \{D\} \\ \varnothing & \varnothing & \{B,D\} & \{B\} & \{S,B\} & \{A\} & \{D\} \\ \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{B\} & \varnothing & \{B,D\} \\ \varnothing & \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing \end{pmatrix}$$

$$T_4 = \begin{pmatrix} \varnothing & \{A\} & \{D\} & \{S\} & \{S\} & \varnothing & \{D\} \\ \varnothing & \varnothing & \{B,D\} & \{B\} & \{S,B\} & \{A\} & \{D\} \\ \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{B\} & \varnothing & \{B,D\} \\ \varnothing & \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing \end{pmatrix}$$

Figure 17: Remaining states of the matrix T.

$$\begin{aligned} R'_S &= \{(0,3),(0,4),(1,4)\}, && (1)\\ R'_A &= \{(0,1),(1,5)\}, && (2)\\ R'_B &= \{(1,2),(1,3),(1,4),(5,4),(5,6)\}, && (3)\\ R'_C &= \{(2,3),(3,4),(6,4)\}, && (4)\\ R'_D &= \{(0,2),(0,6),(1,2),(1,6),(5,6)\}. && (5) \end{aligned}$$

Figure 18: The over-approximations of the conjunctive relations for the example query.

7 Evaluation

In this work, we do not estimate the practical value of the algorithm for the context-free path querying w.r.t. the single-path query semantics, since this algorithm have similar complexity as an algortihm for relational query semantics but it depends significant on the implementation of the path searching.

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

7.1 CFPQ using relational query semantics

To show the practical applicability of the algorithm for context-free path querying w.r.t. the relational query semantics, we implement this algorithm using a variety of optimizations and apply these implementations to the navigation query problem for a dataset of popular ontologies taken from [9]. We also compare the performance of our implementations with existing

analogs from [11, 9]. These analogs use more complex algorithms, while our algorithm uses only simple matrix operations.

Since the Algorithm 1 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 an 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, simply repeating the existing graphs.

We denote the implementation of the algorithm from a paper [11] as *GLL*. The Algorithm 1 is implemented in F# programming language [30] and is available on GitHub[1]. We denote our implementations of the Algorithm 1 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 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.

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

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

[2]GitHub repository of the managedCuda library: `https://kunzmi.github.io/managedCuda/`.

[3]The Math.Net Numerics WebSite: `https://numerics.mathdotnet.com/`.

We evaluate two classical *same-generation queries* [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:

- The grammar $G^1 = (N^1, \Sigma^1, P^1)$.
- The set of non-terminals $N^1 = \{S\}$.
- The set of terminals

$$\Sigma^1 = \{subClassOf, subClassOf^{-1}, type, type^{-1}\}.$$

- The set of production rules P^1 is presented in Figure 19.

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

Figure 19: Production rules for the query 1 grammar.

The grammar G^1 is transformed into an equivalent grammar in normal form, which is necessary for the Algorithm 1. 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 an 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 $S \in a^{cf}_{i,j}$, where a^{cf} is a transitive closure calculated by the Algorithm 1. All implementations in Table 1 have the same #results and demonstrate up to 1000 times better performance as compared to the algorithm presented in [9] 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:

Table 1: Evaluation results for Query 1 (time in ms)

Ontology	#triples	#results	GLL	dGPU	sCPU	sGPU
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	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

- The grammar $G^2 = (N^2, \Sigma^2, P^2)$.
- The set of non-terminals $N^2 = \{S, B\}$.
- The set of terminals

$$\Sigma^2 = \{subClassOf, subClassOf^{-1}\}.$$

- The set of production rules P^2 is presented in Figure 20.

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

Figure 20: Production rules for the query 2 grammar.

The 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.

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*

Table 2: Evaluation results for Query 2 (time in ms)

Ontology	#triples	#results	GLL	dGPU	sCPU	sGPU
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	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

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 the Algorithm 1 can be applied to some real-world problems and it allows us to speed up computations by means of GPGPU.

7.2 A path querying using conjunctive grammar

To show the practical applicability of the algorithm for path query evaluation w.r.t. conjunctive grammars and the relational query semantics, we implement the Algorithm 2 on a CPU and on a GPU. Also, we apply these implementations to some classical conjunctive grammars [20] and synthetic graphs.

Algorithm 2 is implemented in F# programming language [30] and is available on GitHub[4]. We denote our implementations of the Algorithm 2 as follows:

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

- onCPU — 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 package.

- onGPU — 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.

Comparison of the performance of this implementations allows us to determine the efficiency of the GPU acceleration of the Algorithm 2.

We evaluate two queries which correspond to two classical conjunctive grammars.

Query 1 is based on the grammar G^3_S, which generates the language $\{a^n b^n c^n | n > 0\}$, where:

- The grammar $G^3 = (N^3, \Sigma^3, P^3)$.

- The set of non-terminals $N^3 = \{S, A, B, C, D\}$.

- The set of terminals $\Sigma^3 = \{a, b, c\}$.

- The set of production rules P^3 is presented in Figure 21.

$$
\begin{array}{lrcl}
0: & S & \rightarrow & AB \ \& \ DC \\
1: & A & \rightarrow & AA \mid a \\
2: & B & \rightarrow & bBc \mid bc \\
3: & C & \rightarrow & CC \mid c \\
4: & D & \rightarrow & aDb \mid ab
\end{array}
$$

Figure 21: Production rules for the query 1 conjunctive grammar.

The grammar G^3 is transformed into an equivalent grammar in normal form, which is necessary for the Algorithm 2. Let R'_S be an over-approximation of the conjunctive relation for a start non-terminal in the transformed grammar, which is computed by the Algorithm 2.

Table 3: Evaluation results for conjunctive Query 1 (time in ms)

|V|	|E|	#results	onCPU(in ms)	onGPU(in ms)
100	25	0	2	7
100	75	0	10	20
100	200	79	101	213
1000	250	1	265	25
1000	750	13	2781	102
1000	2000	731	12050	347
10000	2500	4	26595	41
10000	7500	136	241087	213
10000	20000	4388	1305177	1316

Table 4: Evaluation results for conjunctive Query 2 (time in ms)

|V|	|E|	#results	onCPU(in ms)	onGPU(in ms)
100	25	9	14	67
100	75	29	114	129
100	100	47	254	483
1000	250	82	2566	127
1000	750	279	21394	530
1000	1000	438	64725	1951
10000	2500	829	268843	257
10000	7500	2796	3380046	1675
10000	10000	27668	—	3017

The result of query 1 evaluation is presented in Table 3, where $|V|$ is a number of nodes in the graph, $|E|$ is a number of edges, and #results is a number of pairs (n, m) in the approximation R'_Sc of the conjunctive relation R_S. The implementation which uses a CPU demonstrates a better performance only on some small graphs. We can conclude that acceleration from the GPU increases with the graph size growth.

Query 2 is based on the grammar G^4_S, which generates the language $\{wcw|w \in \{a, b\}^*\}$, where:

- The grammar $G^4 = (N^4, \Sigma^4, P^4)$.
- The set of non-terminals $N^4 = \{S, A, B, C, D, E\}$.
- The set of terminals $\Sigma^4 = \{a, b, c\}$.

- The set of production rules P^4 is presented in Figure 22.

$$
\begin{array}{lrl}
0: & S & \rightarrow C \ \& \ D \\
1: & C & \rightarrow aCa \mid aCb \mid bCa \mid bCb \mid c \\
2: & D & \rightarrow aA \ \& \ aD \mid bB \ \& \ bD \mid cE \\
3: & A & \rightarrow aAa \mid aAb \mid bAa \mid bAb \mid cEa \\
4: & B & \rightarrow aBa \mid aBb \mid bBa \mid bBb \mid cEb \\
5: & E & \rightarrow aE \mid bE \mid \varepsilon
\end{array}
$$

Figure 22: Production rules for the query 2 conjunctive grammar.

The grammar G^4 is transformed into an equivalent grammar in normal form. Let R'_S be an over-approximation of the conjunctive relation for a start non-terminal in the transformed grammar, which is computed by the Algorithm 2.

The result of the query 2 evaluation is presented in Table 4. On almost all graphs *onGPU* implementation demonstrates a better performance than *onCPU* implementation and we also can conclude that acceleration from the GPU increases with the graph size growth.

As a result, we conclude that the Algorithm 2 can be applied to some real-world problems and it allows us to speed up computations by means of GPGPU.

8 Conclusion

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

We can identify several open problems for further research. In this work, we have considered only two semantics of context-free path querying but there are other important semantics, such as all-path query semantics [14] which requires presenting all paths for all triples (A, m, n). Context-free path querying implemented with the algorithm [11] can answer the queries in the all-path query semantics by constructing a parse forest. It is possible to construct a parse forest for a linear input by matrix multiplication [23]. 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 [21], which have more expressive power than context-free and conjunctive grammars. Boolean path querying problems are undecidable [13] but our algorithm for path querying with conjunctive grammars can be trivially generalized to work on Boolean grammars because parsing with Boolean grammars can be expressed by matrix multiplication [23]. It is not clear what a result of our algorithm applied to this grammars would look like. Our hypothesis is that it would produce

some upper approximation of a solution.

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

There is an algorithm [16] for transitive closure calculation on directed graphs which generalized to handle graph sizes inherently larger than the DRAM memory available on the GPU. Therefore, the question is whether it is possible to apply this approach to the matrix transitive closure calculation in the path querying problem.

References

[1] Abiteboul Serge, Hull Richard, Vianu Victor. Foundations of databases: the logical level. — Addison-Wesley Longman Publishing Co., Inc., 1995.

[2] Abiteboul Serge, Vianu Victor. Regular path queries with constraints // Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems / ACM. — 1997. — P. 122–133.

[3] Adding regular expressions to graph reachability and pattern queries / Wenfei Fan, Jianzhong Li, Shuai Ma et al. // Data Engineering (ICDE), 2011 IEEE 27th International Conference on / IEEE. — 2011. — P. 39–50.

[4] Barrett Chris, Jacob Riko, Marathe Madhav. Formal-language-constrained path problems // SIAM Journal on Computing. — 2000. — Vol. 30, no. 3. — P. 809–837.

[5] Bastani Osbert, Anand Saswat, Aiken Alex. Specification inference using context-free language reachability // ACM SIGPLAN Notices / ACM. — Vol. 50. — 2015. — P. 553–566.

[6] Che Shuai, Beckmann Bradford M, Reinhardt Steven K. Programming GPGPU Graph Applications with Linear Algebra Building Blocks // International Journal of Parallel Programming. — 2016. — P. 1–23.

[7] Choi Jaeyoung, Walker David W, Dongarra Jack J. PUMMA: Parallel universal matrix multiplication algorithms on distributed memory concurrent computers // Concurrency and Computation: Practice and Experience. — 1994. — Vol. 6, no. 7. — P. 543–570.

[8] Chomsky Noam. On certain formal properties of grammars // Information and control. — 1959. — Vol. 2, no. 2. — P. 137–167.

[9] Context-free path queries on RDF graphs / X. Zhang, Z. Feng, X. Wang et al. // International Semantic Web Conference / Springer. — 2016. — P. 632–648.

[10] Fast algorithms for Dyck-CFL-reachability with applications to alias analysis / Qirun Zhang, Michael R Lyu, Hao Yuan, Zhendong Su // ACM SIGPLAN Notices / ACM. — Vol. 48. — 2013. — P. 435–446.

[11] Grigorev Semyon, Ragozina Anastasiya. Context-Free Path Querying with Structural Representation of Result // arXiv preprint arXiv:1612.08872. — 2016.

[12] Grune Dick, Jacobs Ceriel J. H. Parsing Techniques (Monographs in Computer Science). — Secaucus, NJ, USA : Springer-Verlag New York, Inc., 2006. — ISBN: 038720248X.

[13] Hellings J. Conjunctive context-free path queries. — 2014.

[14] Hellings Jelle. Querying for Paths in Graphs using Context-Free Path Queries // arXiv preprint arXiv:1502.02242. — 2015.

[15] AN EFFICIENT RECOGNITION AND SYNTAXANALYSIS ALGORITHM FOR CONTEXT-FREE LANGUAGES. : Rep. / DTIC Document ; Executor: Tadao Kasami : 1965.

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

[17] Kodumal John, Aiken Alex. The set constraint/CFL reachability connection in practice // ACM Sigplan Notices. — 2004. — Vol. 39, no. 6. — P. 207–218.

[18] Mendelzon A., Wood P. Finding Regular Simple Paths in Graph Databases // SIAM J. Computing. — 1995. — Vol. 24, no. 6. — P. 1235–1258.

[19] Nolé Maurizio, Sartiani Carlo. Regular path queries on massive graphs // Proceedings of the 28th International Conference on Scientific and Statistical Database Management / ACM. — 2016. — P. 13.

[20] Okhotin Alexander. Conjunctive grammars // Journal of Automata, Languages and Combinatorics. — 2001. — Vol. 6, no. 4. — P. 519–535.

[21] Okhotin Alexander. Boolean grammars // Information and Computation. — 2004. — Vol. 194, no. 1. — P. 19–48.

[22] Okhotin Alexander. Conjunctive and Boolean grammars: the true general case of the context-free grammars // Computer Science Review. — 2013. — Vol. 9. — P. 27–59.

[23] Okhotin Alexander. Parsing by matrix multiplication generalized to Boolean grammars // Theoretical Computer Science. — 2014. — Vol. 516. — P. 101–120.

[24] Quantifying variances in comparative RNA secondary structure prediction / James WJ Anderson, Ádám Novák, Zsuzsanna Sükösd et al. // BMC bioinformatics. — 2013. — Vol. 14, no. 1. — P. 149.

[25] Reps Thomas. Program analysis via graph reachability // Information and software technology. — 1998. — Vol. 40, no. 11. — P. 701–726.

[26] Reutter Juan L, Romero Miguel, Vardi Moshe Y. Regular queries on graph databases // Theory of Computing Systems. — 2017. — Vol. 61, no. 1. — P. 31–83.

[27] Scott Elizabeth, Johnstone Adrian. GLL parsing // Electronic Notes in Theoretical Computer Science. — 2010. — Vol. 253, no. 7. — P. 177–189.

[28] Sevon Petteri, Eronen Lauri. Subgraph queries by context-free grammars // Journal of Integrative Bioinformatics. — 2008. — Vol. 5, no. 2. — P. 100.

[29] Song Fengguang, Tomov Stanimire, Dongarra Jack. Enabling and scaling matrix computations on heterogeneous multi-core and multi-GPU systems // Proceedings of the 26th ACM international conference on Supercomputing / ACM. — 2012. — P. 365–376.

[30] Syme Don, Granicz Adam, Cisternino Antonio. Expert F# 3.0. — Springer, 2012.

[31] Valiant Leslie G. General context-free recognition in less than cubic time // Journal of computer and system sciences. — 1975. — Vol. 10, no. 2. — P. 308–315.

[32] Xu Guoqing, Rountev Atanas, Sridharan Manu. Scaling CFL-reachability-based points-to analysis using context-sensitive must-not-alias analysis // ECOOP / Springer. — Vol. 9. — 2009. — P. 98–122.

[33] Yannakakis Mihalis. Graph-theoretic methods in database theory // Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems / ACM. — 1990. — P. 230–242.

[34] Younger Daniel H. Recognition and parsing of context-free languages in time n3 // Information and control. — 1967. — Vol. 10, no. 2. — P. 189–208.

[35] Zhang Peng, Gao Yuxiang. Matrix multiplication on high-density multi-GPU architectures: theoretical and experimental investigations // International Conference on High Performance Computing / Springer. — 2015. — P. 17–30.

[36] Zhang Qirun, Su Zhendong. Context-sensitive data-dependence analysis via linear conjunctive language reachability // Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages / ACM. — 2017. — P. 344–358.