

# Path Querying with Conjunctive Grammars by Matrix Multiplication

R. Azimov<sup>a,b,\*</sup> and S. Grigorev<sup>a,b,\*\*,\*\*\*</sup>

<sup>a</sup>St. Petersburg State University, St. Petersburg, 199034 Russia

<sup>b</sup>JetBrains Research, St. Petersburg, 197374 Russia

\* e-mail: st013567@student.spbu.ru

\*\* e-mail: s.v.grigoriev@spbu.ru

\*\*\* e-mail: semyon.grigorev@jetbrains.com

Received February 13, 2019; revised February 13, 2019; accepted February 15, 2019

**Abstract**—Problems in many areas can be reduced to one of the formal-languages-constrained path problems. Conjunctive grammars are more expressive than context-free grammars and are used, for example, in static analysis to describe an interleaved matched-parentheses language, which is not context-free. Path querying with conjunctive grammars is known to be undecidable. There is an algorithm for path querying with linear conjunctive grammars which provides an over-approximation of the result, but there is no algorithm for arbitrary conjunctive grammars. We propose the first algorithm for path querying with arbitrary conjunctive grammars. The proposed algorithm is matrix-based and allows us to efficiently apply GPGPU (General-Purpose computing on Graphics Processing Units) computing techniques and other optimizations for matrix operations.

**Keywords:** conjunctive grammars, path querying, graphs, transitive closure, GPGPU, static analysis, matrix multiplication

**DOI:** 10.1134/S0361768819070041

## 1. INTRODUCTION

Problems in many areas can be reduced to one of the formal-languages-constrained path problems [1]. For example, various problems of static code analysis [2, 3] can be formulated in terms of the context-free language reachability [4] or in terms of the linear conjunctive language reachability [5].

The result of a context-free path query 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 nonterminal  $A$  of the given context-free grammar. This type of query is evaluated using the *relational query semantics* [6]. There is a number of algorithms for context-free path query evaluation using this semantics [6–10].

In [7], the matrix-based algorithm for context-free path query evaluation w.r.t. relational query semantics is proposed. 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 allows us to efficiently apply *GPGPU* (General-Purpose computing on Graphics Processing Units) computing techniques and other optimizations for matrix operations.

In static code analysis the expressiveness of context-free grammars is not enough and conjunctive grammars [11] are better suited to do the task. For example, conjunctive grammars are used in static analysis to describe an interleaved matched-parentheses language [5], which is not context-free. Path querying with conjunctive grammars is undecidable [6]. There is an algorithm [5] for path querying with linear conjunctive grammars [11] which provides an over-approximation of the result. However, there is no algorithm for path querying with arbitrary conjunctive grammars.

The purpose of this work is to develop the algorithm for path querying with arbitrary conjunctive grammars using the matrix-based approach from [7]. It will also be shown that the proposed algorithm allows us to efficiently apply *GPGPU* computing techniques.

This paper is structured as follows. Section 2 defines some notions, used later on. In Section 3 the overview of related works is presented. Section 4 discusses our matrix-based algorithm for path querying with arbitrary conjunctive grammar and provides a step-by-step demonstration for a small example. We evaluate the performance of our algorithm in Section 5, and provide some concluding remarks in Section 6.

## 2. PRELIMINARIES

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

Let  $\Sigma$  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  $\pi$  in a graph  $D$ , we denote the unique word obtained by concatenating the labels of the edges along the path  $\pi$  as  $l(\pi)$ . Also, we write  $n\pi m$  to indicate that a path  $\pi$  starts at the node  $n \in V$  and ends at the node  $m \in V$ .

We deviate from the usual definition of a conjunctive grammar in the *binary normal form* [11] by not including a special start nonterminal, which will be specified in the queries to the graph. Since every conjunctive grammar can be transformed into the equivalent one in the binary normal form [11] 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 nonterminals,  $\Sigma$  is a finite set of terminals, and  $P$  is a finite set of productions of the following forms:

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

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

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

$$\dots A \dots \rightarrow \dots (B_1C_1 \& \dots \& B_mC_m) \dots$$

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

$$\dots (w \& \dots \& w) \dots \rightarrow \dots w \dots$$

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

This definition of conjunctive grammars uses rewriting rules. Also, there are other definitions for the semantics of these grammars. For example, there is a definition using language equations with union, intersection and concatenation [12].

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  $R_A = \{(n, m) \mid \exists n\pi m (l(\pi) \in L(G_A))\}$ . Note that this is similar to the definition of the context-free relations which is used for the context-free path querying [6].

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  $c_{i,j} = \{A \mid \exists (A \rightarrow B_1C_1 \& \dots \& B_mC_m) \in P \text{ such that } (B_k, C_k) \in d_{i,j}\}$ , where  $d_{i,j} = \bigcup_{k=1}^n a_{i,k} \times b_{k,j}$ .

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

## 3. RELATED WORKS

The regular language constrained path querying is widely used for graph analysis [13–16].

There are a number of solutions [6, 7, 9, 10] for context-free path query evaluation w.r.t. the relational query semantics, which employ such parsing algorithms as CYK [17, 18] or Earley [19]. 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 nonterminal  $A$  for all triples  $(A, m, n)$  evaluated using the relational query semantics. While the single-path query semantics requires presenting only one such a path for all the node-pairs  $(m, n)$ . Hellings [20] 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 [20], annotated grammars are proposed as a possible solution.

In [8], 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 [21]. This solution uses derivation trees for the result representation which is more native for grammar-based analysis. The algorithms in [8, 20] 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.

Hellings [6] 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 conjunctive relations which are similar to the context-free relations.

There is an algorithm [5] for path querying with linear conjunctive grammars and relational query semantics. These grammars have no more than one nonterminal in each conjunct of the rule. The possibility of creating an algorithm for path query evaluation w.r.t. arbitrary conjunctive grammars is an open

$$\begin{aligned}
S &\rightarrow AB \ \& \ DC \\
A &\rightarrow a \\
B &\rightarrow BC \\
B &\rightarrow b \\
C &\rightarrow c \\
D &\rightarrow AD \\
D &\rightarrow b
\end{aligned}$$

Fig. 1. An example of the input conjunctive grammar.

problem since the linear conjunctive grammars are known to be strictly less powerful than the arbitrary conjunctive grammars [11].

#### 4. PATH QUERYING ALGORITHM FOR CONJUNCTIVE GRAMMARS

In this section, we show how the path querying for 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 [6].

##### 4.1. 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  $b_{i,j} = \{A_k \mid (i, x, j) \in E, (A_k \rightarrow x) \in P\}$ . Finally, we compute the conjunctive transitive closure  $b^{conj} = b^{(1)} \cup b^{(2)} \cup \dots$  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 hold.

**Lemma 1.** *Let  $D = (V, E)$  be a graph, let  $G = (N, \Sigma, P)$  be a conjunctive grammar. Then for any  $i, j$  and for any nonterminal  $A \in N$ , 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.* (By Induction)

**Basis:** Show that the statement of the lemma holds for  $k = 1$ . For any  $i, j$  and for any nonterminal  $A \in N$ , 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  $(A \rightarrow x) \in P$  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  $A \rightarrow B_1 C_1 \ \& \ \dots \ \& \ B_m C_m$  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 nonterminals  $B_1, C_1, \dots, B_m, C_m$  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  $x = 1 \dots m$  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  $x = 1 \dots m$ . Since  $b^{(p)} = b^{(p-1)} \cup (b^{(p-1)} \circ b^{(p-1)})$  and  $(b^{(p-1)} \circ b^{(p-1)})_{i,j} = \{A \mid \exists (A \rightarrow B_1 C_1 \ \& \ \dots \ \& \ B_m C_m) \in P \text{ such that } (B_k, C_k) \in d_{i,j}\}$  then  $A \in b_{i,j}^{(p)}$  and the statement of the lemma holds for  $k = p$ .

**Theorem 1.** *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 nonterminal  $A \in N$ , if  $(i, j) \in R_A$  then  $A \in b_{i,j}^{conj}$ .*

*Proof.* By the Lemma 1, 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  $b^{conj} = b^{(1)} \cup b^{(2)} \cup \dots$ , then for any  $i, j$  and for any nonterminal  $A \in N$ , 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  $b$ .

##### 4.2. The Algorithm

In this section we introduce the algorithm for calculating the conjunctive transitive closure  $b^{conj}$  which was discussed in Section 1.

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

---

**Algorithm 1** Conjunctive recognizer for graphs
 

---

```

1: function conjunctive Graph Parsing( $D, G$ )
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  $\emptyset$ 
6: for all  $(i, x, j) \in E$  do ▷ Matrix initialization
7:    $T_{i,j} \leftarrow T_{i,j} \cup \{A \mid (A \rightarrow x) \in P\}$ 
8: while matrix  $T$  is changing do
9:    $T \leftarrow T \cup (T \circ T)$  ▷ Transitive closure calculation
10: return  $T$ 

```

---

Similar to the case of the context-free grammars [7], we can show that the Algorithm 1 terminates in a finite number of steps. The total number of nonterminals in the matrix  $T$  does not exceed  $|V|^2|N|$  since each element of the matrix  $T$  contains no more than  $|N|$  nonterminals. Therefore, the following theorem holds.

**Theorem 2.** *Let  $D = (V, E)$  be a graph and let  $G = (N, \Sigma, P)$  be a conjunctive grammar. Algorithm 1 terminates in a finite number of steps.*

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

Let the size of the input conjunctive grammar be a constant. Similar to the case of the context-free grammars [7], it can be shown that operations in the line 9 of the Algorithm 1 can be calculated in a constant number of multiplications, unions (element-wise maximums), and intersections (element-wise minimums) of two  $n \times n$  Boolean matrices. Denote the number of elementary operations executed by the algorithm of computing these three operations on two  $n \times n$  Boolean matrices as  $BMM(n)$ ,  $BMU(n)$ ,  $BMI(n)$ , respectively. Since the line 9 of the Algorithm 1 is exe-

cuted no more than  $|V|^2|N|$  times, the following theorem holds.

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

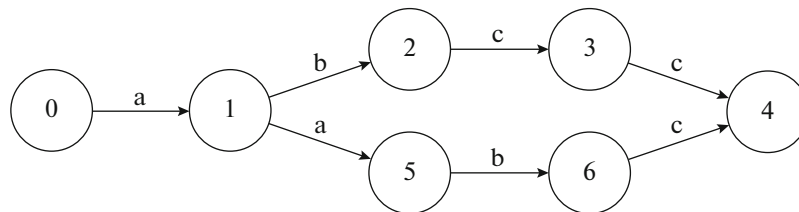
#### 4.3. Example

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 nonterminals  $N = \{S, A, B, C, D\}$ .
- The set of terminals  $\Sigma = \{a, b, c\}$ .
- The set of production rules  $P$  is shown in Fig. 1.

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 run the query on a graph presented in Fig. 2.

We provide a step-by-step demonstration of the work with the given graph  $D$  and grammar  $G$  of the



**Fig. 2.** An example of the input graph.

$$\begin{pmatrix} \emptyset & \{A\} & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \{B, D\} & \emptyset & \emptyset & \{A\} & \emptyset \\ \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{B, D\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \end{pmatrix}$$

Fig. 3. The initial matrix.

Algorithm 1. After the matrix initialization in lines 6, 7 of the Algorithm 1, we have a matrix  $T_0$  (Fig. 3):

Let  $T_i$  be the matrix  $T$  obtained after executing the loop in lines 8, 9 of the Algorithm 1  $i$  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 shown in Fig. 4.

The matrix  $T_1 = T_1 = T_0 \cup (T_0 \circ T_0)$  is shown in Fig. 5.

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 the nonterminal from the left side of this rule to the set  $T_{ij}$ .

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 Fig. 6.

Thus, the result of the Algorithm 1 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  $R_A$ . Approximations for each nonterminal of the grammar  $G$  is shown in Fig. 7.

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  $S \rightarrow AB \ \& \ DC$ , 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  $abc$ .

Thus, the Algorithm 1 does not calculate the exact solution, but an over-approximation.

### 5. EVALUATION

To show that the proposed algorithm allows us to effectively apply GPGPU computing techniques, we implement the Algorithm 1 on a CPU and on a GPU. Also, we apply these implementations to some classical conjunctive grammars [12] and synthetic graphs.

Algorithm 1 is implemented in F# programming language [22] and is available on GitHub<sup>1</sup>. We denote our implementations of the Algorithm 1 as follows:

- 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<sup>2</sup> package.
- onGPU – an implementation using the CSR format for sparse matrix representation and a GPU for

<sup>1</sup> GitHub repository of the YaccConstructor project: <https://github.com/YaccConstructor/YaccConstructor>.

<sup>2</sup> The Math.Net Numerics WebSite: <https://numerics.mathdotnet.com/>.

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

Fig. 4. The matrix  $d$ .

$$\begin{pmatrix} \emptyset & \{A\} & \{D\} & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \{B, D\} & \{B\} & \emptyset & \{A\} & \{D\} \\ \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{B\} & \emptyset & \{B, D\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \end{pmatrix}$$

Fig. 5. The matrix  $T$  after first iteration.

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<sup>3</sup> library.

Comparison of the performance of this implementations allows us to determine the efficiency of the GPU acceleration of the Algorithm 1.

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

- OS: Microsoft Windows, 64 bit;
- CPU: 3.60GHz, 4 Cores;
- GPU: NVIDIA GeForce GTX 1070 (1920 Cuda cores, 1556 MHz, 8192 MB).

We consider two queries which correspond to classical conjunctive grammars, and evaluate these queries on random graphs with labels from the set  $\{a, b, c\}$ .

<sup>3</sup> GitHub repository of the managedCuda library: <https://kuzmi.github.io/managedCuda/>.

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

$$T_3 = \begin{pmatrix} \emptyset & \{A\} & \{D\} & \{S\} & \{S\} & \emptyset & \{D\} \\ \emptyset & \emptyset & \{B, D\} & \{B\} & \{S, B\} & \{A\} & \{D\} \\ \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{B\} & \emptyset & \{B, D\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \end{pmatrix} \quad T_4 = \begin{pmatrix} \emptyset & \{A\} & \{D\} & \{S\} & \{S\} & \emptyset & \{D\} \\ \emptyset & \emptyset & \{B, D\} & \{B\} & \{S, B\} & \{A\} & \{D\} \\ \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{B\} & \emptyset & \{B, D\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \end{pmatrix}$$

Fig. 6. Remaining states of the matrix  $T$ .

**Query 1** is based on the normalized grammar  $G^1$ , such that  $L(G_S^1) = \{a^n b^n c^n \mid n > 0\}$ , where:

- The grammar  $G^1 = (N^1, \Sigma^1, P^1)$ .
- The set of nonterminals  $N^1 = \{S, A, B, C, D\}$ .
- The set of terminals  $\Sigma^1 = \{a, b, c\}$ .
- The set of production rules  $P^1$  is shown in Fig. 8.

Let  $R_S^1$  be an over-approximation of the conjunctive relation for a start nonterminal in the transformed grammar, which is computed by the Algorithm 1.

The result of query 1 evaluation is presented in Table 1, where  $|V|$  is a number of nodes in the graph,  $|E|$  is a number of edges, and no. results is a number of pairs  $(n, m)$  in the approximation  $R_S^1$  of the conjunctive relation  $R_S$ . The implementation which uses a CPU demonstrates 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 normalized grammar  $G^2$ , such that  $L(G_S^2) = \{w c w \mid w \in \{a, b\}^*\}$ , where:

- The grammar  $G^2 = (N^2, \Sigma^2, P^2)$ .
- The set of nonterminals  $N^2 = \{S, A, B, C, D, E\}$ .
- The set of terminals  $\Sigma^2 = \{a, b, c\}$ .
- The set of production rules  $P^2$  is shown in Fig. 9.

The grammar  $G^2$  is transformed into an equivalent grammar in normal form. The result of the query 2 evaluation is presented in Table 2. On almost all graphs *onGPU* implementation demonstrates a better performance than *onCPU* implementation and we

**Table 1.** Evaluation results for query 1. (time in ms)

$ V $	$ E $	#results	onCPU(ms)	onGPU(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

also can conclude that acceleration from the GPU increases with the graph size growth.

We conclude that the Algorithm 1 allows us to speed up computations by means of GPGPU. We can also use other optimizations for matrix operations to increase the performance of the proposed algorithm.

## 6. CONCLUSION AND FUTURE WORK

In this work, we have shown how the over-approximation of the result of path query evaluation w.r.t. the conjunctive grammars and relational query semantics can be found by the calculation of matrix transitive closure. In addition, we introduced an algorithm for computing this transitive closure for an arbitrary conjunctive grammar. Also, we provided a proof of the correctness of the proposed algorithms. Finally, we have shown that the proposed algorithm allows us to efficiently apply GPGPU computing techniques by running different implementations of this algorithm on classical queries.

We can identify several open problems. In this work, we have considered only one semantics of path querying but there are other important semantics, such as single-path and all-path query semantics [20].

**Table 2.** Evaluation results for query 2

$ V $	$ E $	#results	onCPU(ms)	onGPU(ms)
100	25	9	14	67
100	75	29	114	129
100	200	47	254	483
1000	250	82	2566	127
1000	750	279	21394	530
1000	2000	438	64725	1951
10000	2500	829	268843	257
10000	7500	2796	3380046	1675
10000	20000	27668	—	3017

$$R'_S = \{(0,3), (0,4), (1,4)\} \quad (1)$$

$$R'_A = \{(0,1), (1,5)\} \quad (2)$$

$$R'_B = \{(1,2), (1,3), (1,4), (5,4), (5,6)\} \quad (3)$$

$$R'_C = \{(2,3), (3,4), (6,4)\} \quad (4)$$

$$R'_D = \{(0,2), (0,6), (1,2), (1,6), (5,6)\} \quad (5)$$

**Fig. 7.** An approximation of the conjunctive relations.

$$S \rightarrow AB \& DC$$

$$A \rightarrow AA \mid a$$

$$B \rightarrow bBc \mid bc$$

$$C \rightarrow CC \mid c$$

$$D \rightarrow aDb \mid ab$$

**Fig. 8.** Conjunctive grammar for the query 1.

$$S \rightarrow C \& D$$

$$C \rightarrow aCa \mid aCb \mid bCa \mid bCb \mid c$$

$$D \rightarrow aA \& aD \mid bB \& bD \mid cE$$

$$A \rightarrow aAa \mid aAb \mid bAa \mid bAb \mid cEa$$

$$B \rightarrow aBa \mid aBb \mid bBa \mid bBb \mid cEb$$

$$E \rightarrow aE \mid bE \mid \epsilon$$

**Fig. 9.** Conjunctive grammar for the query 2.

Whether it is possible to generalize our approach for these semantics 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 conjunctive path querying.

There is a number of GPU implementations for graph algorithms which are used in static analysis [23–25]. Also, our algorithm allows us to efficiently apply GPGPU computing techniques and can be used in static analysis, for example, using conjunctive grammar to describe an interleaved matched-parentheses language [5]. The question is whether there are other conjunctive grammars which can be used in static analysis.

## FUNDING

This work was supported by the Russian Science Foundation, grant no. 18-11-00100.

## REFERENCES

1. Barrett, Ch., Jacob, R., and Marathe, M., Formal-language-constrained path problems, *SIAM J. Comput.*, 2000, vol. 30, no. 3, pp. 09–837.
2. Bastani, O., Anand, S., and Aiken, A., Specification inference using context-free language reachability, *ACM SIGPLAN Notices*, ACM, 2015, vol. 50, pp. 553–566.
3. Xu, G., Rountev, A., and Sridharan, M., Scaling c-reachability-based points-to analysis using context-sensitive must-not-alias analysis, *ECOOP*, Springer, 2009, vol. 9, pp. 98–122.
4. Reps, T., Program analysis via graph reachability, *Inf. Software Technol.*, 1998, vol. 40, no. 11, pp. 701–726.
5. Zhang, Q. and Su, Z., Context-sensitive data-dependence analysis via linear conjunctive language reachability, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, ACM, 2017, pp. 344–358.
6. Hellings, J., *Conjunctive Context-Free Path Queries*, 2014.
7. Azimov, R. and Grigorev, S., Context-free path querying by matrix multiplication, *Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences and Systems (GRADES) and Network Data Analytics (NDA)*, ACM, 2018, p. 5.
8. Grigorev, S. and Ragozina, A., Context-free path querying with structural representation of result, *Proceedings of the 13th Central and Eastern European Software Engineering Conference in Russia*, ACM, 2017, p. 10.
9. Sevon, P. and Eronen, L., Subgraph queries by context-free grammars, *J. Integrative Bioinformatics*, 2008, vol. 5, no. 2, p. 100.
10. Zhang, X., Feng, Z., Wang, X., Rao, G., and Wu, W., Context-free path queries on rdf graphs, *International Semantic Web Conference*, Springer, 2016, pp. 632–648.
11. Okhotin, A., Conjunctive and boolean grammars: the true general case of the context-free grammars, *Comput. Sci. Rev.*, 2013, vol. 9, pp. 27–59.
12. Okhotin, A., Conjunctive grammars, *J. Automata, Languages, Combinatorics*, 2001, vol. 6, no. 4, pp. 519–535.
13. Abiteboul, S. and Vianu, V., Regular path queries with constraints, *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, ACM, 1997, pp. 122–133.
14. Fan, W., Li, J., Ma, Sh., Tang, N., and Wu, Y., Adding regular expressions to graph reachability and pattern queries, *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, IEEE, 2011, pp. 39–50.
15. Nolé, M. and Sartiani, C., Regular path queries on massive graphs, *Proceedings of the 28th International Conference on Scientific and Statistical Database Management*, ACM, 2016, p. 13.
16. Reutter, J.L., Romero, M., and Vardi, M.Y., Regular queries on graph databases, *Theory Comput. Systems*, 2017, vol. 61, no. 1, pp. 31–83.
17. Kasami, T., An efficient recognition and syntax analysis algorithm for context-free languages, Technical report, DTIC Document, 1965.
18. Younger, D.H., Recognition and parsing of context-free languages in time  $n^3$ , *Information and Control*, 1967, vol. 10, no. 2, pp. 189–208.
19. Grune, D. and Jacobs, C.J.H., *Parsing Techniques (Monographs in Computer Science)*, NJ, USA, Secaucus: Springer-Verlag-New York, Inc., 2006.
20. Hellings, J., Querying for paths in graphs using context-free path queries, arXiv preprint arXiv:1502.02242, 2015.
21. Scott, E. and Johnstone, A., Gll parsing, *Electronic Notes in Theoretical Computer Science*, 2010, vol. 253, no. 7, pp. 177–189.
22. Syme, D., Granicz, A., and Cisternino, A., *Expert F# 3.0*, Springer, 2012.
23. Mendez-Lojo, M., Burtscher, M., and Pingali, K., A gpu implementation of inclusion-based points-to analysis, *ACM SIGPLAN Notices*, 2012, vol. 47, no. 8, pp. 107–116.
24. Yu Su, Ding Ye, and Jingling Xue, Accelerating inclusion-based pointer analysis on heterogeneous cpu-gpu systems, *High Performance Computing (HiPC), 2013 20th International Conference on*, IEEE, 2013, pp. 149–158.
25. Yu Su, Ding Ye, Jingling Xue, and Xiang-Ke Liao, An efficient gpu implementation of inclusion-based pointer analysis, *IEEE Transactions on Parallel and Distributed Systems*, 2016, vol. 27, no. 2, pp. 353–366.