Graph Parsing by Matrix Multiplication

Rustam Azimov
Saint Petersburg State University
7/9 Universitetskaya nab.
St. Petersburg, 199034 Russia
rustam.azimov19021995@gmail.com

Semyon Grigorev
Saint Petersburg State University
7/9 Universitetskaya nab.
St. Petersburg, 199034 Russia
Semen.Grigorev@jetbrains.com

ABSTRACT

Graph data model is widely used in many areas, for example, bioinformatics, graph databases, RDF. One of the most common graph queries are navigational queries. The result of query evaluation are implicit relations between nodes of the graph, i.e. paths in the graph. A natural way to specify these relations is by specifying paths using formal grammars over edge labels. The answer to the context-free path queries in this approach is usually a set of triples (A, m, n) such that there is a path from node m to node n whose labeling is derived from a non-terminal A of the given context-free grammar. This type of queries is evaluated using the relational query semantics. There is a number of algorithms for query evaluation which use such semantics but they have computational problems with big data. One of the most common technique for efficient big data processing is GPGPU, but these algorithms do not allow to use this technique effectively. In this paper we propose a graph parsing algorithm for query evaluation which use relational query semantics and context-free grammars, and is based on matrix operations which allows to speed up computations by means of GPGPU.

1. INTRODUCTION

Graph data model is widely used in many areas, for example, bioinformatics [2], graph databases [9], RDF [17]. In these areas, it is often required to process large graphs. Most common among graph queries are navigational queries. The result of query evaluation is implicit relations between nodes of the graph, i.e. paths in the graph. A natural way to specify these relations is by specifying paths using formal grammars (regular expressions, context-free grammars) over edge labels. Context-free grammars are actively used in graphs queries because of the limited expressive power of regular expressions.

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

Existing algorithms for query evaluation using this semantics have computational problems with big data. One of the most common technique for efficient big data processing is *GPGPU* (General-Purpose computing on Graphics Processing Units), but these algorithms do not allow to use this technique effectively. The algorithms for context-free recognizing had a similar problem until Valiant [14] proposed a parsing algorithm that computes a recognition table by computing a matrix transitive closure. Thus, the active use of matrix operations (such as matrix multiplication) in the process of computing a transitive closure makes it possible to effectively apply GPGPU computing techniques [3].

Therefore the question is whether it is possible to create an algorithm for query evaluation using the relational query semantics which allows to speed up computations with GPGPU by using the matrix operations.

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

- We show how the query evaluation using the relational query semantics and context-free grammars can be reduced to the calculation of the matrix transitive closure.
- We provide a formal proof of correctness of the proposed reduction.
- We introduce an algorithm for query evaluation which use the relational query semantics and context-free grammars, and is based on matrix operations which allows to speed up computations by means of GPGPU.

2. PRELIMINARIES

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

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

According to Hellings [6], we deviate from the usual definition of a context-free grammar in *Chomsky Normal Form* [4] by not including a special start non-terminal, which will be specified in the 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, then it is sufficient to only consider grammars of the following type. A *context-free 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:

- $A \to BC$, 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 limit the applicability of further algorithms because checking that an empty string belongs to the context-free language in Chomsky normal form is trivial and only the empty paths $m\pi m$ correspond to an empty string ε.

We use the conventional notation $A \xrightarrow{*} w$ 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 *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 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 on arbitrary subsets of N and union of sets as an addition, we can define a *matrix multiplication*, $a \cdot b = c$, where a and b are matrices of the suitable size that have subsets of N as elements, as $c_{i,j} = \bigcup_{k=1}^{n} a_{i,k} \cdot b_{k,j}$.

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

Similar to the case of the context-free grammars, we deviate from the usual definition of a conjunctive grammar in the *binary normal form* [11] 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 [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 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 $A \xrightarrow{*} w$ 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 $L(G_S) = \{w \in \Sigma^* \mid S \xrightarrow{*} w\}$.

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

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

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

3. RELATED WORKS

Our work is inspired by Valiant [14], who proposed an algorithm for general context-free recognition in less than cubic time. This algorithm computes the same parsing table as the Cocke-Kasami-Younger algorithm [8, 16] but does this by offloading the most intensive computations into calls to a Boolean matrix multiplication procedure. This approach not only provides an asymptotically more efficient algorithm but it also allows to effectively apply GPGPU computing techniques. Valiant used the following definition of a transitive closure.

Definition 1. *The transitive closure of a square matrix a is a matrix $a^+ = a^{(1)} \cup a^{(2)} \cup \cdots$ where:*

- $a^{(i)} = \bigcup_{j=1}^{i-1} a^{(j)} \cdot a^{(i-j)}$, $i \geq 2$;
- $a^{(1)} = a$.

Valiant also showed that the matrix multiplication operation used in this approach is computationally no more difficult than a Boolean matrix multiplication. Denote the number of elementary operations executed by the algorithm of multiplying $n \times n$ Boolean matrices as $BMM(n)$. Valiant showed that the matrix multiplication operation used in this approach is essentially the same as $|N|^2$ Boolean matrix multiplications, where $|N|$ is the number of non-terminals of the given context-free grammar in Chomsky normal form.

Hellings [6] presented an algorithm for query evaluation using the relational query semantics and context-free grammars. According to Hellings, for a given graph $D = (V, E)$ and a grammar $G = (N, \Sigma, P)$ the query evaluation using the relational query semantics reduces to a calculation of the relations R_A. Thus, in this paper, we focus on the calculation of these relations.

Yannakakis [15] analyzed the reducibility of various graph parsing problems to the calculation of the transitive closure. He formulated a problem of generalization Valiant's technique to the query evaluation using the relational query semantics and context-free grammars. Also, he assumed that this technique can not be generalized for arbitrary graphs, though it does for acyclic graphs.

Thus, the possibility of reducing the query evaluation using the relational query semantics and context-free grammars to the calculation of the transitive closure is an open problem. In this paper, we do not generalize Valiant's approach. We use a different definition of the transitive closure. The possibility of using Valiant's transitive closure in graph parsing is an open problem.

4. GRAPH PARSING BY THE CALCULATION OF TRANSITIVE CLOSURE

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

4.1 Reducing graph parsing to transitive closure

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

Let $G = (N, \Sigma, P)$ be a 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 transitive closure $b^+ = b^{(1)} \cup b^{(2)} \cup \cdots$ where $b^{(i)} = b^{(i-1)} \cup (b^{(i-1)} \cdot b^{(i-1)})$, $i \geq 2$ and $b^{(1)} = b$. For the transitive closure b^+, the following statements holds.

LEMMA 1. *Let $D = (V, E)$ be a graph, let $G = (N, \Sigma, P)$ be a grammar. Then for any i, j and for any non-terminal $A \in N$, $A \in b^{(k)}_{i,j}$ iff $(i,j) \in R_A$ and $i\pi j$, such that there is a derivation tree according to the string $l(\pi)$ and a context-free grammar $G_A = (N, \Sigma, P, A)$ of the height $h \leq k$.*

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

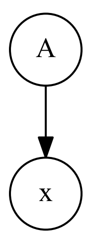

Figure 1: The derivation tree for the string $x = l(\pi)$ of the height $h = 1$.

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 $p \geq 2$. Since $b^{(p)} = b^{(p-1)} \cup (b^{(p-1)} \cdot b^{(p-1)})$ then for any i, j and for any non-terminal $A \in N$, $A \in b^{(p)}_{i,j}$ iff $A \in b^{(p-1)}_{i,j}$ or $A \in (b^{(p-1)} \cdot b^{(p-1)})_{i,j}$.

Let $A \in b^{(p-1)}_{i,j}$. By the inductive hypothesis, $A \in b^{(p-1)}_{i,j}$ iff $(i,j) \in R_A$ and there exists $i\pi j$, such that there is a derivation tree according to the string $l(\pi)$ and a context-free grammar $G_A = (N, \Sigma, P, A)$ of the height $h \leq (p-1)$. Since the height h of this tree is also less than or equal to p, then the statement of the lemma holds for $k = p$.

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

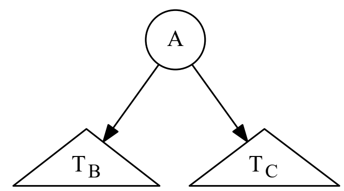

Figure 2: The derivation tree for the string $w = l(\pi)$ of the height $h = 1 + max(h_1, h_2)$, where T_B and T_C are the derivation trees for strings w_1 and w_2 respectively.

Since the height $h = 1 + max(h_1, h_2) \leq p$, then the statement of the lemma holds for $k = p$ and this completes the proof of the lemma. □

THEOREM 1. *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 $A \in N$, $A \in b^{+}_{i,j}$ iff $(i,j) \in R_A$.*

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

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

4.2 The algorithm

In this section we introduce an algorithm for calculating the transitive closure b^+ which was discussed in Section 4.1.

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

Algorithm 1 Context-free recognizer for graphs

function CONTEXTFREEGRAPHPARSING(D, G)
  $n \leftarrow$ a number of nodes in D
  $E \leftarrow$ the directed edge-relation from D
  $P \leftarrow$ a set of production rules in G
  $T \leftarrow$ a matrix $n \times n$ in which each element is $\emptyset$
  for all $(i, x, j) \in E$ **do** ▷ Matrix initialization
    $T_{i,j} \leftarrow T_{i,j} \cup \{A \mid (A \to x) \in P\}$
  while matrix T is changing **do**
    $T \leftarrow T \cup (T \cdot T)$ ▷ Transitive closure calculation
  return T

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

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

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

PROOF. It is sufficient to show, that the operation in 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. $\square$

According to Valiant, required the matrix multiplication operation 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 line **9** of the Algorithm 1 can be calculated in $O(|N|^2 BMU(n))$. Since line **9** of the Algorithm 1 is executed no more than $|V|^2|N|$ times, then the following theorem holds.

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

4.3 The example

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

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

- A set of non-terminals $N = \{S\}$.
- A set of terminals $\Sigma = \{subClassOf, subClassOf^{-1}, type, type^{-1}\}$.
- A set of production rules P is presented in Figure 3.

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

Figure 3: Production rules for the example query grammar.

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

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

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

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

- A set of terminals $\Sigma' = \{subClassOf, subClassOf^{-1}, type, type^{-1}\}$.
- A set of production rules P' is presented in Figure 4.

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

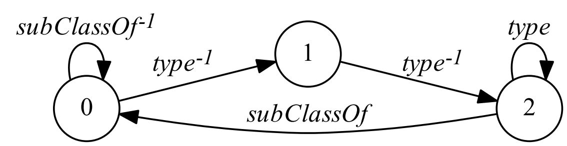

Figure 5: An input graph for the example query.

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

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

Figure 6: Initial matrix for the example query.

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

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

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

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.

$$T_0 \cdot T_0 = \begin{pmatrix} \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \{S\} \\ \emptyset & \emptyset & \emptyset \end{pmatrix}$$

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

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

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

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

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

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

Figure 8: Remaining states of the matrix T.

These relations for each non-terminal of the grammar G' are presented in Figure 9.

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

Figure 9: Resulting context-free relations for the example query.

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

5. GRAPH PARSING USING CONJUNCTIVE GRAMMARS

Since the query evaluation using the relational query semantics and conjunctive grammars is undecidable problem [6] then we propose an algorithm that calculates the over-approximation of all conjunctive relations R_A.

5.1 Reducing graph parsing 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.

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 2. *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 $A \in b^{(k)}_{i,j}$.*

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 $A \in b^{(1)}_{i,j}$ 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 $A \in b^{(p-1)}_{i,j}$. Since $b^{(p)} = b^{(p-1)} \cup (b^{(p-1)} \circ b^{(p-1)})$ then $A \in b^{(p)}_{i,j}$ 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 $B_x \in b^{(p-1)}_{i,t_x}$ and $C_x \in b^{(p-1)}_{t_x,j}$, 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 $d_{i,j} = \bigcup_{t=1}^{n} b^{(p-1)}_{i,t} \times b^{(p-1)}_{t,j}$. 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 $(b^{(p-1)} \circ b^{(p-1)})_{i,j} = \{A \mid \exists (A \to B_1C_1 \,\&\ldots\&\, B_mC_m) \in P$ such that $(B_k, C_k) \in d_{i,j}\}$ then $A \in b^{(p)}_{i,j}$ and the statement of the lemma holds for $k = p$. This completes the proof of the lemma. $\square$

Theorem 4. *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 $A \in b^{conj}_{i,j}$.*

Proof. By the lemma 2, if $(i,j) \in R_A$ then $A \in b^{(k)}_{i,j}$ 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 $A \in b^{(k)}_{i,j}$ for some $k \geq 1$ then $A \in b^{conj}_{i,j}$. Therefore, if $(i, j) \in R_A$ then $A \in b^{conj}_{i,j}$. 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.

5.2 The algorithm

In this section we introduce an algorithm for calculating the conjunctive transitive closure b^{conj} which was discussed in Section 5.1.

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 5. *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. EVALUATION

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

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

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

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

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

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

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

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

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

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

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

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

Table 1: Evaluation results for Query 1

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

Table 2: Evaluation results for Query 2

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

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

Figure 10: Production rules for the query 1 grammar.

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

The result of query 1 evaluation is presented in Table 1, where #triples is a number of triples (o, p, s) in a RDF file, and #results is a number of pairs (n, m) in the context-free relation R_S. We can determine whether $(i, j) \in R_S$ by asking whether $S \in b^+_{i,j}$, where b^+ is the transitive closure calculated by the proposed algorithm. Since a dense matrix representation significantly degrades performance with increasing of the graph size, then we omit *dGPU* performance on graphs g_1, g_2 and g_3. All implementations in Table 1 have the same #results and demonstrate up to 1000 times better performance as compared to the algorithm presented in [17] for Q_1. Our implementation *sGPU* demonstrates better performance than *GLL* with increasing of the graph size. We also can conclude that acceleration from the *GPU* increases with the size of the graph.

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

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

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

Figure 11: Production rules for the query 2 grammar.

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

The result of the query 2 evaluation is presented in Table 2. Since a dense matrix representation significantly degrades performance with increasing of the graph size, then we omit *dGPU* performance on graphs g_1, g_2 and g_3. All implementations in Table 2 have the same #results. On almost all graphs *sGPU* demonstrates better performance than *GLL* implementation and we also can conclude that acceleration from the *GPU* increases with the size of the graph.

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

7. CONCLUSION AND FUTURE WORK

In this paper, we presented the algorithm for reducing graph query evaluation using the relational query semantics to the calculation of matrix transitive closure. Also, we provide a formal proof of the correctness of the proposed reduction. In addition, we introduce an algorithm for computing this transitive closure, which allows to effectively apply GPGPU computing techniques. Finally, we show the practical applicability of the proposed algorithm by running different implementations of our algorithm on real-world data.

We identify several open problems for further research. In this paper we have considered only one semantics of graph querying but there are other important semantics, such as *single-path* and *all-path* semantics [7], which require to present paths, not only check reachability. Graph parsing implemented with algorithm [5] can answer the queries in these semantics by parsing a forest construction. It is possible to construct a parsing forest for a linear input parsing by the matrix multiplication [12]. Whether it is possible to generalize this approach for a graph input is an open question.

In our algorithm, we calculate the matrix transitive closure naively, but there are algorithms for the transitive closure calculation, which are asymptotically more efficient. Therefore, the question is whether it is possible to apply these algorithms for the matrix transitive closure calculation to the problem of graph parsing. One way to answer this question is to generalize the Valiant's technique to arbitrary graphs. Yannakakis [15] formulated this problem and assumed that Valiant's technique does not seem to generalize to arbitrary graphs.

Also, there are Boolean grammars [10], which have more expressive power than context-free grammars. Graph parsing with boolean grammars is undecidable problem [6] but our algorithm can be trivially generalized to work on boolean grammars because parsing with boolean grammars can be expressed by matrix multiplication [12]. It is not clear, what will be a result of our algorithm applied to Boolean grammars. Our hypothesis is that it will produce the upper approximation of a solution.

Matrix multiplication in the main loop of the proposed algorithm may be performed on different GPGPU independently. It can help to utilize the power of multi-GPU systems and increase the performance of graph parsing.

Acknowledgments

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

8. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. Foundations of databases, 1995.

[2] J. W. Anderson, Á. Novák, Z. Sükösd, M. Golden, P. Arunapuram, I. Edvardsson, and J. Hein. Quantifying variances in comparative rna secondary structure prediction. *BMC bioinformatics*, 14(1):149, 2013.

[3] S. Che, B. M. Beckmann, and S. K. Reinhardt. Programming gpgpu graph applications with linear algebra building blocks. *International Journal of Parallel Programming*, pages 1–23, 2016.

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

[5] S. Grigorev and A. Ragozina. Context-free path querying with structural representation of result. *arXiv preprint arXiv:1612.08872*, 2016.

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

[7] J. Hellings. Querying for paths in graphs using context-free path queries. *arXiv preprint arXiv:1502.02242*, 2015.

[8] T. Kasami. An efficient recognition and syntaxanalysis algorithm for context-free languages. Technical report, DTIC Document, 1965.

[9] A. Mendelzon and P. Wood. Finding regular simple paths in graph databases. *SIAM J. Computing*, 24(6):1235–1258, 1995.

[10] A. Okhotin. Boolean grammars. *Information and Computation*, 194(1):19–48, 2004.

[11] A. Okhotin. Conjunctive and boolean grammars: the true general case of the context-free grammars. *Computer Science Review*, 9:27–59, 2013.

[12] A. Okhotin. Parsing by matrix multiplication generalized to boolean grammars. *Theoretical Computer Science*, 516:101–120, 2014.

[13] D. Syme, A. Granicz, and A. Cisternino. *Expert F# 3.0*. Springer, 2012.

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

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

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

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