Context-Free Path Querying by Matrix Multiplication

ABSTRACT

Context-free path querying is a technique, which recently gains popularity in many areas, for example, graph databases, bioinformatics, static analysis, etc. In some of these areas, it is often required to query large graphs, and existing algorithms demonstrate a poor performance in this case. The generalization of matrix-based Valiant's context-free language recognition algorithm for graph case is widely considered as a recipe for efficient context-free path querying; however, no progress has been made in this direction so far.

We propose the first generalization of matrix-based Valiant's algorithm for context-free path querying. Our generalization does not deliver a truly sub-cubic worst-case complexity algorithm, whose existence still remains a hard open problem in the area. On the other hand, the utilization of matrix operations (such as matrix multiplication) in the process of context-free path query evaluation makes it possible to efficiently apply a wide class of optimizations and computing techniques, such as *GPGPU* (General-Purpose computing on Graphics Processing Units), parallel processing, sparse matrix representation, distributed-memory computation, etc. Indeed, the evaluation on a set of conventional benchmarks shows, that our algorithm outperforms the existing ones.

CCS CONCEPTS

• Information systems → Query languages for non-relational engines; **• Theory of computation → Grammars and context-free languages**;

KEYWORDS

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

1 INTRODUCTION

Graph data models are widely used in many areas, for example, graph databases [14], bioinformatics [3], static analysis [13, 28], etc. In these areas, it is often required to process queries for large graphs. The most common type of graph queries is 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 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* [10]. There is a number of algorithms for context-free path query evaluation using this semantics [8, 10, 21, 30].

The existing algorithms for context-free path query evaluation w.r.t. relational semantics demonstrate a poor performance when applied to large graphs. The algorithms for context-free language recognition had a similar problem until Valiant [24] 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.

We propose the first matrix-based algorithm for context-free path query evaluation using relational query semantics. Valiant's algorithm computes the transitive closure of an upper triangular matrix by increasing the length of paths considered. We use another definition of matrix transitive closure, which does not depend explicitly on the path length since the cyclic graphs contain paths of infinite length. Additionally, we compute the transitive closure of an arbitrary matrix, since the context-free path query evaluation requires to process arbitrary graphs. While we do not achieve the same worst-case time complexity for graph input as Valiant's algorithm for linear case, the use of matrix operations (such as matrix multiplication) in our algorithm makes it possible to efficiently apply such computing techniques as *GPGPU* (General-Purpose computing on Graphics Processing Units) and parallel computation [4]. 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 [5, 22, 27].

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

- we show, how the context-free path querying w.r.t. relational query semantics can be reduced to the calculation of matrix transitive closure;
- we introduce a matrix-based algorithm for context-free path querying w.r.t. relational query semantics which is based on matrix operations that makes it possible to speed up computations, using GPGPU;
- we prove the correctness of our algorithm;

GRADES-NDA'18, June 10, 2018, Houston, Texas, USA

ACM ISBN 123-4567-24-567/08/06...$15.00
https://doi.org/10.475/123_4

- we show the practical applicability of our algorithm by presenting the results of its evaluation on a set of conventional benchmarks.

This paper is structured as follows: the section 2 provides a small motivating example; the section 3 defines some notions, used later on; in the section 4 the overview of related works is presented; the section 5 discusses our matrix-based algorithm for context-free path querying and provides a step-by-step demonstration for a small example; we evaluate the performance of our algorithm in the section 6, and provide some concluding remarks in the section 7.

2 THE MOTIVATING EXAMPLE

In this section, we formulate the problem of context-free path query evaluation, using a small graph and the classical *same-generation query* [1], which cannot be expressed using regular expressions.

Let us have a graph database or any other object, which can be represented as a graph. The same-generation query can be used for discovering a vertex similarity, for example, gene similarity [21]. For graph databases, the same-generation query is aimed at the finding all the nodes at the same hierarchy level. The language, formed by the paths between such nodes, is not regular and corresponds to the language of matching parentheses. Hence, the query is formulated as a context-free grammar.

For example, let us have a small double-cyclic graph (see Figure 1). One of the cycles has three edges, labeled with a, and the other has two edges, labeled with b. Both cycles are connected via a shared node 0.

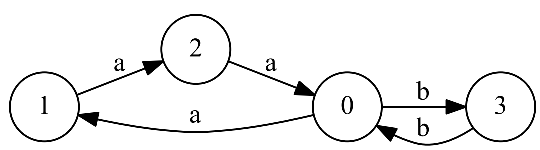

Figure 1: An example graph.

For this graph, we have a same-generation query, formulated as a context-free grammar, which generates a context-free language $L = \{a^n b^n ; n \geq 1\}$.

The result of context-free path query evaluation for this example is a set of node pairs (m, n), such that there is a path from the node m to the node n, whose labeling forms a word from the language L. For example, the node pair $(0, 0)$ must be in this set, since there is a path from the node 0 to the node 0, whose labeling forms a string $w = aaaaaabbbbbb = a^6 b^6 \in L$.

3 PRELIMINARIES

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

Following Hellings [10], we deviate from the usual definition of a context-free grammar in *Chomsky Normal Form* [6] by not including a special starting non-terminal, which will be specified in the path queries for the graph. Since every context-free grammar can be transformed into an equivalent one in Chomsky Normal Form and checking, that empty string belongs to 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 empty string. This does not restrict the applicability of our algorithm since only the empty paths $m\pi m$ correspond to 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 production rule applications 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 $(\cdot)$ for 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 subset multiplication, and union 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 [24], we define the *transitive closure* of a square matrix a as $a^+ = a_+^{(1)} \cup a_+^{(2)} \cup \cdots$, where $a_+^{(1)} = a$ and

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

Valiant proposed an algorithm for a context-free recognition for a linear input, which in graph terms corresponds to a directed chain of nodes. The algorithm enumerates the positions in the input string s from 0 to the length of s, constructs an upper-triangular matrix, and computes its transtive closure. In the context-free path querying input graphs can be arbitrary, which removes the upper-triangularity limitation. For this reason, we introduce another definition of transitive closure for arbitrary square matrix a as $a^{cf} = a^{(1)} \cup a^{(2)} \cup \cdots$, where $a^{(1)} = a$ and

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

These two transitive closure definitions are in fact equivalent (a formal proof can be found in Appendix A). Furthermore, in this paper we use the transitive closure a^{cf} instead of a^+, and the algorithm for computing a^{cf} also computes Valiant's transitive closure a^+.

4 RELATED WORKS

Traditionally, query languages for graph databases use regular expressions to describe paths to find [2, 7, 14, 15, 19], but there are some other useful queries, which cannot be expressed by regular expressions. For example, there are classical *same-generation queries* [1], which can be used for finding all the nodes at the same level in some hierarchy, and are useful for discovering vertex similarity. The context-free path querying algorithms can be used to evaluate such types of queries since this queries can be represented by context-free grammars.

There are a number of solutions [10, 21, 30] for context-free path query evaluation w.r.t. relational query semantics, which make use of such parsing algorithms as CYK [12, 26] or Earley [9].

Hellings [10] presented an algorithm for context-free path query evaluation using 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. relational query semantics reduces to a calculation of a set of context-free relations R_A. Thus, in this paper, we focus on the calculation of these context-free relations. Also, the algorithm in [10] was implemented by [30] in the context of RDF processing.

Other examples of path query semantics are *single-path* and *all-path query semantics* [11]. The all-path query semantics requires a finding of all possible paths from a node m to a node n whose labelings are derived from a non-terminal A. The single-path query semantics requires presenting only one such path. Hellings [11] presented some algorithms for context-free path query evaluation using single-path and all-path query semantics. If a context-free path query w.r.t. all-path query semantics is evaluated for cyclic graphs, then the query result can be an infinite set of paths. For this reason, in [11] annotated grammars were proposed as a way to represent the results.

In [8], an algorithm for a context-free path query evaluation w.r.t. all-path query semantics is proposed. This algorithm is based on the generalized top-down parsing algorithm (GLL) [20]. For the result representation, this solution uses derivation trees, which is more native for grammar-based analysis. The algorithms [8, 11] for context-free path query evaluation w.r.t. all-path query semantics can also be used for query evaluation using relational and single-path semantics.

Our work is inspired by Valiant [24], who proposed an algorithm for general context-free recognition in less than cubic time. This algorithm computes the same parsing table as CYK algorithm but does this by offloading the most intensive computations into calls to the Boolean matrix multiplication procedure. This approach not only provides an asymptotically more efficient algorithm but 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 has shown, that the matrix multiplication operation ($\times$) is essentially the same as $|N|^2$ Boolean matrix multiplications, where $|N|$ is the number of non-terminals in the given context-free grammar in Chomsky normal form.

Yannakakis [25] analyzed the reducibility of various path querying problems to the calculation of transitive closure. He formulated a problem of Valiant's technique generalization for the context-free path query evaluation w.r.t. relational query semantics. Also, he conjectured, that this technique cannot be generalized for arbitrary graphs, though it does for acyclic graphs.

Thus, reducing a context-free path query evaluation w.r.t. relational query semantics to a calculation of matrix transitive closure was an open problem until now.

5 CONTEXT-FREE PATH QUERYING BY TRANSITIVE CLOSURE CALCULATION

In this section, we show, how the context-free path query evaluation using 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.

5.1 Reducing Context-Free Path Querying to the Calculation of Transitive Closure

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

Let $G = (N, \Sigma, P)$ be a grammar and $D = (V, E)$ be a graph. We enumerate the nodes of the graph D from 0 to $(|V| - 1)$. We initialize the elements of 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 5.1. *Let $D = (V, E)$ be a graph, let $G = (N, \Sigma, P)$ be a grammar. Then for any i, j and for any non-terminal $A \in N$, $A \in a^{(k)}_{i,j}$ 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)

Base case: Show that the lemma holds for $k = 1$. For any i, j and for any non-terminal $A \in N$, $A \in a^{(1)}_{i,j}$ 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 on Figure 2, for the string x and a context-free grammar $G_A = (N, \Sigma, P, A)$. Thus, it has been shown that the lemma holds for $k = 1$.

Inductive step: Assume that 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 $A \in N$,

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

since

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

Let $A \in a^{(p-1)}_{i,j}$. By the inductive hypothesis, $A \in a^{(p-1)}_{i,j}$ iff $(i, j) \in R_A$ and there exists $i\pi j$, such that there is a derivation tree of the

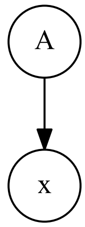

Figure 2: The derivation tree of the height $h = 1$ for the string $x = l(\pi)$.

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, $B \in a^{(p-1)}_{i,r}$ and $C \in a^{(p-1)}_{r,j}$, such that $(A \to BC) \in P$. Hence, by the inductive hypothesis, there are $i\pi_1 r$ and $r\pi_2 j$, such that $(i, r) \in R_B$ and $(r, j) \in R_C$, and there are the derivation trees T_B and T_C of heights $h_1 \leq (p-1)$ 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 on Figure 3, for the string $w = l(\pi)$ and a context-free grammar G_A.

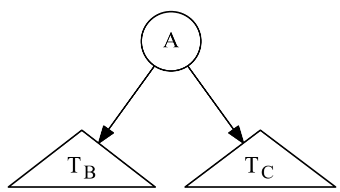

Figure 3: The derivation tree of the height $h = 1 + max(h_1, h_2)$ for the string $w = l(\pi)$, where T_B and T_C are the derivation trees for strings w_1 and w_2 respectively.

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 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 $A \in N$, $A \in a^{cf}_{i,j}$ iff $(i, j) \in R_A$.*

Proof. Since the matrix $a^{cf} = a^{(1)} \cup a^{(2)} \cup \cdots$, for any i, j and for any non-terminal $A \in N$, $A \in a^{cf}_{i,j}$ iff there is $k \geq 1$, such that $A \in a^{(k)}_{i,j}$. By the lemma 5.1, $A \in a^{(k)}_{i,j}$ iff $(i, j) \in R_A$ and there is $i\pi j$, such that there is a derivation tree of the height $h \leq k$ 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 $A \in a^{cf}_{i,j}$. Thus, we show how the context-free relations R_A can be calculated by computing the transitive closure a^{cf} of the matrix a.

5.2 The algorithm

In this section, we introduce an algorithm for calculating the transitive closure a^{cf} which was discussed in section 5.1.

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, 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 2. *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. □

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 3. *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 3 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 4.

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

Figure 4: 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 1.

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 $h = 2 * u * v > 2 * v * v = |V|^2/4 = O(|V|^2)$.

5.3 An Example

In this section, we provide a step-by-step demonstration of the proposed algorithm. For this, we consider the example with the worst-case time complexity.

The **example query** is based on the context-free grammar $G = (N, \Sigma, P)$ of the worst-case example query which was discussed in section 5.2. The set of production rules for this grammar is shown on Figure 4.

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 $N' = \{S, S_1, A, B\}$;
- the set of terminals $\Sigma' = \{a, b\}$;
- the set of production rules P' is presented on Figure 5.

We run the query on a graph, presented on Figure 1. 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 on Figure 6.

$$\begin{array}{llll} 0: & S & \rightarrow & A\, B \\ 1: & S & \rightarrow & A\, S_1 \\ 2: & S_1 & \rightarrow & S\, B \\ 3: & A & \rightarrow & a \\ 4: & B & \rightarrow & b \end{array}$$

Figure 5: Production rules for the example query grammar in normal form.

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

Figure 6: The initial matrix for the example query.

Let T_i be the matrix T, obtained after executing the loop in the lines **8-9** of the Algorithm 1 i times. The calculation of the matrix T_1 is shown on Figure 7.

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

$$T_1 = T_0 \cup (T_0 \times T_0) = \begin{pmatrix} \varnothing & \{A\} & \varnothing & \{B\} \\ \varnothing & \varnothing & \{A\} & \varnothing \\ \{A\} & \varnothing & \varnothing & \{\boldsymbol{S}\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

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

When the algorithm at some iteration finds new paths in the graph D, then it adds corresponding nonterminals to the matrix T. For example, after the first loop iteration, non-terminal S is added to the matrix T. This non-terminal is added to the element with a row index $i = 2$ and a column index $j = 3$. This means, that there is $i\pi j$ (a path π from the node 2 to the node 3), such that $S \xrightarrow{*} l(\pi)$. For example, such a path consists of two edges with labels a and b, and thus $S \xrightarrow{*} a\, b$.

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, $k = 13$ since $T_{13} = T_{12}$. The remaining iterations of computing the transitive closure are presented on Figure 8 (new matrix elements on each iteration are shown in bold).

Thus, the result of the Algorithm 1 for the example query is the matrix $T_{13} = T_{12}$. 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 on Figure 9.

In the context-free relation R_S, we have all node pairs corresponding to paths, whose labeling is in the language $L(G_S) = \{a^n b^n; n \geq 1\}$. This conclusion is based on the fact, that the grammar G'_S is equivalent to the grammar G_S and $L(G_S) = L(G'_S)$.

$$T_2 = \begin{pmatrix} \varnothing & \{A\} & \varnothing & \{B\} \\ \varnothing & \varnothing & \{A\} & \varnothing \\ \{A, \boldsymbol{S_1}\} & \varnothing & \varnothing & \{S\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix} \quad T_3 = \begin{pmatrix} \varnothing & \{A\} & \varnothing & \{B\} \\ \{\boldsymbol{S}\} & \varnothing & \{A\} & \varnothing \\ \{A, S_1\} & \varnothing & \varnothing & \{S\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix} \quad T_4 = \begin{pmatrix} \varnothing & \{A\} & \varnothing & \{B\} \\ \{S\} & \varnothing & \{A\} & \{\boldsymbol{S_1}\} \\ \{A, S_1\} & \varnothing & \varnothing & \{S\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

$$T_5 = \begin{pmatrix} \varnothing & \{A\} & \varnothing & \{B, \boldsymbol{S}\} \\ \{S\} & \varnothing & \{A\} & \{S_1\} \\ \{A, S_1\} & \varnothing & \varnothing & \{S\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix} \quad T_6 = \begin{pmatrix} \{\boldsymbol{S_1}\} & \{A\} & \varnothing & \{B, S\} \\ \{S\} & \varnothing & \{A\} & \{S_1\} \\ \{A, S_1\} & \varnothing & \varnothing & \{S\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix} \quad T_7 = \begin{pmatrix} \{S_1\} & \{A\} & \varnothing & \{B, S\} \\ \{S\} & \varnothing & \{A\} & \{S_1\} \\ \{A, S_1, \boldsymbol{S}\} & \varnothing & \varnothing & \{S\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

$$T_8 = \begin{pmatrix} \{S_1\} & \{A\} & \varnothing & \{B, S\} \\ \{S\} & \varnothing & \{A\} & \{S_1\} \\ \{A, S_1, S\} & \varnothing & \varnothing & \{S, \boldsymbol{S_1}\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix} \quad T_9 = \begin{pmatrix} \{S_1\} & \{A\} & \varnothing & \{B, S\} \\ \{S\} & \varnothing & \{A\} & \{S_1, \boldsymbol{S}\} \\ \{A, S_1, S\} & \varnothing & \varnothing & \{S, S_1\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix} \quad T_{10} = \begin{pmatrix} \{S_1\} & \{A\} & \varnothing & \{B, S\} \\ \{S, \boldsymbol{S_1}\} & \varnothing & \{A\} & \{S_1, S\} \\ \{A, S_1, S\} & \varnothing & \varnothing & \{S, S_1\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

$$T_{11} = \begin{pmatrix} \{S_1, \boldsymbol{S}\} & \{A\} & \varnothing & \{B, S\} \\ \{S, S_1\} & \varnothing & \{A\} & \{S_1, S\} \\ \{A, S_1, S\} & \varnothing & \varnothing & \{S, S_1\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix} \quad T_{12} = \begin{pmatrix} \{S_1, S\} & \{A\} & \varnothing & \{B, S, \boldsymbol{S_1}\} \\ \{S, S_1\} & \varnothing & \{A\} & \{S_1, S\} \\ \{A, S_1, S\} & \varnothing & \varnothing & \{S, S_1\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix} \quad T_{13} = \begin{pmatrix} \{S_1, S\} & \{A\} & \varnothing & \{B, S, S_1\} \\ \{S, S_1\} & \varnothing & \{A\} & \{S_1, S\} \\ \{A, S_1, S\} & \varnothing & \varnothing & \{S, S_1\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

Figure 8: Remaining states of the matrix T.

$$\begin{aligned} R_S &= \{(0,0),(0,3),(1,0),(1,3),(2,0),(2,3)\}, \\ R_{S_1} &= \{(0,0),(0,3),(1,0),(1,3),(2,0),(2,3)\}, \\ R_A &= \{(0,1),(1,2),(2,0)\}, \\ R_B &= \{(0,3),(3,0)\}. \end{aligned}$$

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

6 EVALUATION

To show the practical applicability of the proposed algorithm for context-free path querying w.r.t. relational query semantics, we implemented this algorithm with a number of optimizations and applied these implementations to the navigation query problem for some popular ontologies, taken from [30]. We also compared the performance of our implementations with existing analogues from [8, 30]. These analogues use more complex algorithms, while our algorithm uses only the simple matrix operations.

Since our algorithm works with graphs, each RDF file from a dataset was converted to an edge-labeled directed graph as follows. For each triple (o, p, s) from an RDF file, we added an edge (o, p, s) to the graph. In addition, we added an edge (s, p^{-1}, o) to the graph, if p corresponded to the terminals *subClassOf* and *type* of the query grammars. We also constructed synthetic graphs g_1, g_2 and g_3, simply repeating 8 times the existing graphs for *funding*, *wine* and *pizza*, respectively.

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 [8] as *GLL*. Our algorithm can be easily implemented using the existing libraries for matrix operations calculation on a CPU or on a GPU. We did not find suitable libraries for bit matrix multiplication to implement our Boolean matrix multiplication. Therefore, we used standard libraries for matrix operations. The algorithm, presented in this paper, is implemented in F# programming language [23] and is available on GitHub. We denote our implementations of the proposed algorithm as follows:

- dGPU (dense GPU) — an implementation with a row-major matrix representation and a GPU for matrix operation calculation. For the calculations of matrix operations on a GPU, we used a wrapper for the CUBLAS library from the managedCuda[1] library.
- sCPU (sparse CPU) — an implementation using the CSR format for sparse matrix representation and a CPU for matrix operation calculation. For sparse matrix representation in CSR format we used the Math.Net Numerics[2] package.
- sGPU (sparse GPU) — an implementation using the CSR format for sparse matrix representation and a GPU for matrix operation calculation. For calculations of the matrix operations on a GPU, where matrices represented in a CSR format, we used 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 as the graph size grows.

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

We evaluated two classical *same-generation queries* [1] which, for example, can be used in bioinformatics.

Query 1 is based on the grammar G^1_S for retrieving the 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 on Figure 10.

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

Figure 10: Production rules for the query 1 grammar.

The grammar G^1 is transformed into an equivalent grammar in normal form, which is necessary for our algorithm. 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 V is a number of vertices in the constructed graph, E is a number of edges in the graph, 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 proposed algorithm. All implementations in Table 1 have the same #results and demonstrate up to 1000 times better performance in comparison with the algorithm, presented in [30]. Our implementation *sGPU* demonstrates a better performance than *GLL*, on almost all graphs. *GLL* is faster than *sGPU* only on two small graphs due to a data transfer between CPU and GPU. Also, for this query, the acceleration from the *GPU* increases with the graph size growth.

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

- 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 on Figure 11.

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

Figure 11: 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* demonstrates a better performance than *GLL* implementation, and we also can conclude, that acceleration from the *GPU* increases with the graph size growth.

As a result, we conclude, that our algorithm can be applied to some real-world problems and allows us to speed up the computations, using GPGPU. Also, our algorithm can be easily implemented using standard libraries for matrix operations calculation. We can also speculate, that the use of bit matrix multiplication algorithms to implement our Boolean matrix multiplication can significantly improve the performance of our algorithm.

7 CONCLUSION AND FUTURE WORK

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

The active use of matrix operations (such as matrix multiplication) in the proposed algorithm makes it possible to efficiently apply a wide class of matrix optimizations and computing techniques (GPGPU, parallel processing, sparse matrix representation, distributed-memory computation, etc.)

We can identify several open problems for future research. In this paper, we have considered only one semantics of context-free path querying, but there are other important semantics, such as single-path and all-path query semantics [11]. Context-free path querying, implemented with the algorithm [8], can process the queries in all-path query semantics by constructing a parse forest. It is possible to construct a parse forest for a linear input by matrix multiplication [18]. 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 better. Therefore, the question is if it is possible to apply these algorithms for matrix transitive closure calculation for context-free path querying.

Also, there are conjunctive [17] and Boolean grammars [16], which have more expressive power, than context-free grammars. Path querying with conjunctive and Boolean grammars is known to be undecidable [10], but our algorithm can be trivially generalized to work for these grammars because parsing with conjunctive and Boolean grammars can be expressed by matrix multiplication [18]. It is not clear, how the results of our algorithm can be interpreted in this case. Our conjecture is that it produces an upper approximation of the solution. Also, path querying problem w.r.t. conjunctive grammars can be applied to static code analysis [29].

Table 1: Evaluation results for Query 1

Ontology	V	E	#results	GLL(ms)	dGPU(ms)	sCPU(ms)	sGPU(ms)
skos	144	323	810	10	56	14	12
generations	129	351	2164	19	62	20	13
travel	131	397	2499	24	69	22	30
univ-bench	179	413	2540	25	81	25	15
atom-primitive	291	685	15454	255	190	92	22
biomedical-measure-primitive	341	711	15156	261	266	113	20
foaf	256	815	4118	39	154	48	9
people-pets	337	834	9472	89	392	142	32
funding	778	1480	17634	212	1410	447	36
wine	733	2450	66572	819	2047	797	54
pizza	671	2604	56195	697	1104	430	24
g_1	6224	11840	141072	1926	—	26957	82
g_2	5864	19600	532576	6246	—	46809	185
g_3	5368	20832	449560	7014	—	24967	127

Table 2: Evaluation results for Query 2

Ontology	V	E	#results	GLL(ms)	dGPU(ms)	sCPU(ms)	sGPU(ms)
skos	144	323	1	1	10	2	1
generations	129	351	0	1	9	2	0
travel	131	397	63	1	31	7	10
univ-bench	179	413	81	11	55	15	9
atom-primitive	291	685	122	66	36	9	2
biomedical-measure-primitive	341	711	2871	45	276	91	24
foaf	256	815	10	2	53	14	3
people-pets	337	834	37	3	144	38	6
funding	778	1480	1158	23	1246	344	27
wine	733	2450	133	8	722	179	6
pizza	671	2604	1262	29	943	258	23
g_1	6224	11840	9264	167	—	21115	38
g_2	5864	19600	1064	46	—	10874	21
g_3	5368	20832	10096	393	—	15736	40

A EQUIVALENCE OF TRANSITIVE CLOSURE DEFINITIONS

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

Lemma A.1. *Let $G = (N, \Sigma, P, S)$ be a context-free grammar in Chomsky Normal Form, let a be a square matrix. Then $a^{(k)} \geq a_+^{(k)}$ for any $k \geq 1$.*

Proof. (Proof by Induction)

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

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

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

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

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

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

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

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

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

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

and from this it follows that

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

By the definition,

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

and this completes the proof of the lemma. □

Lemma A.2. *Let $G = (N, \Sigma, P, S)$ be a context-free grammar in Chomsky Normal Form, let a be a square matrix. Then for any $k \geq 1$ there is $j \geq 1$, such that $(\bigcup_{i=1}^{j} a_+^{(i)}) \geq a^{(k)}$.*

Proof. (Proof by Induction)

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

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

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

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

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

By the definition,

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

and from this it follows that

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

The following holds

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

since

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

and

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

Therefore there is $2j$, such that

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

and this completes the proof of the lemma. □

Theorem 4. *Let $G = (N, \Sigma, P, S)$ be a context-free grammar in Chomsky Normal Form, let a be a square matrix. Then $a^+ = a^{cf}$.*

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

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

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

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

Hence

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

for any $k \geq 1$. Therefore

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

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

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

and this completes the proof of the theorem. □

REFERENCES

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

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

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

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

[5] Jaeyoung Choi, David W Walker, and Jack J Dongarra. 1994. PUMMA: Parallel universal matrix multiplication algorithms on distributed memory concurrent computers. *Concurrency and Computation: Practice and Experience* 6, 7 (1994), 543–570.

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

[7] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Yinghui Wu. 2011. Adding regular expressions to graph reachability and pattern queries. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on.* IEEE, 39–50.

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

[9] Dick Grune and Ceriel J. H. Jacobs. 2006. *Parsing Techniques (Monographs in Computer Science).* Springer-Verlag New York, Inc., Secaucus, NJ, USA.

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

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

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

[13] John Kodumal and Alex Aiken. 2004. The set constraint/CFL reachability connection in practice. *ACM Sigplan Notices* 39, 6 (2004), 207–218.

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

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

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

[17] Alexander Okhotin. 2013. Conjunctive and Boolean grammars: the true general case of the context-free grammars. *Computer Science Review* 9 (2013), 27–59.

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

[19] Juan L Reutter, Miguel Romero, and Moshe Y Vardi. 2017. Regular queries on graph databases. *Theory of Computing Systems* 61, 1 (2017), 31–83.

[20] Elizabeth Scott and Adrian Johnstone. 2010. GLL parsing. *Electronic Notes in Theoretical Computer Science* 253, 7 (2010), 177–189.

[21] Petteri Sevon and Lauri Eronen. 2008. Subgraph queries by context-free grammars. *Journal of Integrative Bioinformatics* 5, 2 (2008), 100.

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

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

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

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

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

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

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

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

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