Multiple Context-Free Path Querying by Matrix Multiplication*

Ilya Epelbaum[1], Rustam Azimov[1,2], and
Semyon Grigorev[1,2][0000−0002−7966−0698]

[1] St.Petersburg State University, 7/9 Universitetskaya nab., St. Petersburg, Russia, 199034
iliyepelbaun@gmail.com, rustam.azimov19021995@gmail.com, s.v.grigoriev@spbu.ru

[2] JetBrains Research, Primorskiy prospekt 68-70, Building 1, St. Petersburg, Russia, 197374

Abstract. Many graph analysis problems can be formulated as formal language-constrained path querying problems where the formal languages are used as constraints for navigational path queries. Recently, the context-free language (CFL) reachability formulation has become very popular and can be used in many areas, for example, querying graph databases, RDF analysis, static code analysis, biological data analysis, graph segmentation. However, the generative capacity of context-free grammars (CFGs) is too weak to generate some complex queries, for example, from natural languages, and the various extensions of CFGs have been proposed. Multiple context-free grammar (MCFG) is one of such extensions of CFGs. Despite the fact that, to the best of our knowledge, there is no algorithm for MCFL-reachability, this problem is known to be decidable. The creation of the MCFL-reachability algorithms allows one to use more complex graph queries that may find application in various areas, for example, in static code analysis. In this paper, we provide the first MCFL-reachability algorithm that also is based on Boolean matrix multiplication. Thus, the proposed algorithm can be implemented by using high-performance libraries and modern parallel hardware.

Keywords: Multiple context-free path querying · Graph database · RDF · Multiple context-free grammars · Boolean matrix multiplication · GraphBLAS API.

1 Introduction

Many graph analysis problems can be formulated as formal language-constrained path querying [3] problems where the formal languages are used as constraints for navigational path queries. More precisely, a path in an edge-labeled graph is viewed as a word (or a string) constructed by the concatenation of edge labels, and the formal languages are used to constrain the paths of interest. When answering a query to the graph some information about paths labeled by words

* The research was supported by the Russian Science Foundation, grant №18-11-00100

from the given formal language should be found. Recently, the context-free language (CFL) reachability formulation [15] become very popular and can be used in many areas, for example, querying graph databeses [20], RDF analysis [25], static code analysis [26, 14], biological data analysis [18], graph segmentation [12].

However, the generative capacity of context-free grammars (CFGs) is too weak to generate natural languages, and the various extensions of CFGs have been proposed to define the syntax of natural languages. *Multiple context-free grammar* (MCFG) is one of such extensions of CFGs. A nonterminal of an MCFG derives tuples of words while the nonterminals of a CFG can only derive words. Using the MCFGs, it is possible to formulate more complex path constraints as, for example, structures involving discontinuous constituents such as "respectively" sentences or inverted sentences in a simple manner.

Therefore, the creation of the MCFL-reachability algorithms allows one to use more complex graph queries. Such algorithms may find application in various areas, for example, in static code analysis. In practice, the Dyck language [23, 11] is the most widely used language in CFL-reachability problem. This language essentially generates the well-matched parentheses. Particularly, many program analyses use the Dyck language to exactly model the *matched-parenthesis* property for *context-sensitivity* or *data-dependence* analysis [16]. Namely, context-sensitivity describes the well-balanced procedure calls and returns using open and close parentheses, respectively. Similarly, the data-dependence represents another well-balanced property among language constructors, for example, field accesses (i.e., reads and writes [4, 22]), pointer indirections (i.e., references and dereferences [27]), etc. However, the precise analysis that captures two or more well-balanced properties is undecidable [16]. For example, the context-sensitive and data-dependence analysis describes an interleaved matched-parenthesis language which is not even context-free. The traditional approach is to approximate the solution using the CFL-reachabilty algorithms. An interleaved matched-parenthesis language can be viewed as the intersection of two CFLs. However, the CFLs are not closed under intersection [8]. Therefore, the precision of either context-sensitivity or data-dependence must be sacrificed by approximating the corresponding Dyck language using a regular one [9, 19].

However, for more precise analysis other classes of formal languages can be used. For example, the *linear conjunctive languages* (LCL) can be applied for context-sensitive data-dependence analysis and demonstrate significant precision and scalability advantages of this approach [24]. Thus, the class of *multiple context-free languages* (MCFLs) may also contain the formal languages that can be used to increase the precision of the solution for some program analysis problems. One of the candidates for such a language is the O_n language that can model the matched number of opening and closing parentheses for context-sensitive data-dependence analysis. These languages are approximations of interleaved matched-parenthesis languages and are known not to be context-free. For example, $O_2 = \{w \in \{a, \bar{a}, b, \bar{b}\}^* \mid |w|_a = |w|_{\bar{a}} \wedge |w|_b = |w|_{\bar{b}}\}$.

Despite the fact that, to the best of our knowledge, there is no algorithm for MCFL-reachability, this problem is known to be decidable because the MCFL

are closed under intersection with regular languages (i.e. with graphs) and the reachability information can be computed by checking the resulting language for emptiness, which is a decidable problem [17].

In practice, the good receipt to achieve high-performance solutions for graph analysis problems is to offload the most critical computations into linear algebra (LA) operations, for example matrix operations [10]. Then such algorithm can be effectively implemented using the high-performance LA libraries with wide class of optimizations like parallel computations [1, 2]. There are LA-based efficient MCFL recognition algorithms that use the Boolean matrix multiplications [13, 5] and can form the basis of new MCFL-reachability algorithms. However, the MCFL parsing algorithm in [5] can be applied only for some subclass of multiple context-free grammars called *unbalanced*.

To sum up, we make the following contributions in this paper.

1. We provide the first MCFL-reachability algorithm by extending the MCFL parsing algorithm from [13]. Our algorithm is LA-based, hence it allows one to use high-performance libraries and utilize modern parallel hardware.
2. We implement the proposed algorithm using the pygraphblas[3] implementation of the GraphBLAS API [10] and evaluate it on some real RDFs and synthetic graphs using some classical MCFLs.

2 Preliminaries

In this section, we introduce common definitions in graph theory and formal language theory which are used in this paper. Also, we provide a brief description of the MCFL-reachability problem.

2.1 Basic Definitions of Graph Theory

In this paper, we use a labeled directed graph as a data model and define it as follows.

Definition 1. Labeled directed graph *is a tuple* $D = (V, E, \Sigma)$*, where*

- V *is a finite set of vertices. For simplicity, we assume that the vertices are natural numbers ranging from* 1 *to* $|V|$*,*
- Σ *is a set of edge labels,*
- $E \subseteq V \times \Sigma \times V$ *is a set of labeled edges.*

An example of the labeled directed graph D_1 is presented in Figure 1 where the set of vertices $V = \{1, 2, 3, 4, 5\}$, the set of edge labels $\Sigma = \{a, b, c, d\}$, and the set of labeled edges

$$E = \{(1, a, 2), (2, d, 1), (2, b, 3), (3, c, 2), (3, c, 4), (4, b, 3), (4, d, 5), (5, a, 4)\}.$$

[3] The pygraphblas GitHub repository: https://github.com/Graphegon/pygraphblas. Access date: 06.04.2022.

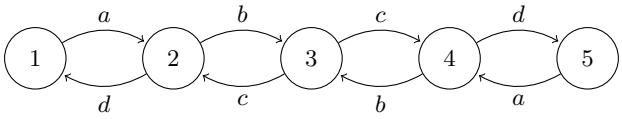

Fig. 1: The input graph D_1

Definition 2. *A* path π *in the graph* $D = (V, E, \Sigma)$ *is a finite sequence of labeled edges* $(e_1, ..., e_n)$*, where* $\forall i : 1 \leq i \leq n \;\; e_i = (v_{i-1}, l_i, v_i) \in E$*.*

Definition 3. *The* word $l(\pi) \in \Sigma^*$ *for the graph* $D = (V, E, \Sigma)$ *is the unique word* $l_1...l_n$*, obtained by concatenating the labels of the edges along the path*

$$\pi = (e_1 = (v_0, l_1, v_1), \ldots, e_n = (v_{n-1}, l_n, v_n)).$$

Definition 4. *Let* $D = (V, E, \Sigma)$ *be an labeled directed graph. The* adjacency matrix M_D *of the graph* D *is a matrix of size* $|V| \times |V|$*, such that*

$$M_D[i, j] = \{l \mid (i, l, j) \in E\}$$

It is usual to decompose the adjacency matrix into a set of Boolean matrices for the implementation reasons. Thus, we introduce the following definitions.

Definition 5. *Let* M_D *be the adjacency matrix of the graph* $D = (V, E, \Sigma)$*. An* Boolean adjacency matrix for the label $l \in \Sigma$ *of the graph* D *is a matrix* M_D^l *of size* $|V| \times |V|$*, such that*

$$M_D^l[i, j] = \begin{cases} True & , l \in M_D[i, j] \\ False & , otherwise \end{cases}$$

Definition 6. *A* Boolean decomposition of adjacency matrix M_D *of the graph* D *is a set of Boolean matrices* $\mathcal{M} = \{M_D^l \mid l \in \Sigma\}$*.*

2.2 Basic Definitions of Formal Languages

We use multiple context-free languages (MCFLs) as path constraints, thus in this subsection, we define multiple context-free languages and grammars.

Definition 7. *A* multiple context-free grammar G *is a tuple* (N, Σ, P, S, d)*, where*

- N *is a finite set of nonterminals*
- $S \in N$ *is the start nonterminal*
- *For each* $A \in N$ *a natural number* $d(A)$*, called the dimension of* A*, is defined. In particular, we assume* $d(S) = 1$*. Sometimes* A *is written as* $(A^1, \ldots, A^{d(A)})$*, where* $\forall i : 1 \leq i \leq d(A) \;\; A^i$ *are called component symbols of* A*. Let* $N_c = \{A^i | A \in N, 1 \leq i \leq d(A)\}$ *be the set of component symbols.*

- Σ *is a finite set of terminals,* $N \cap \Sigma = \varnothing$
- P *is a finite set of production rules.* $p \in P$ *has a form of* $p : (A^1, \ldots, A^{d(A)}) \to (\gamma_1, \ldots, \gamma_{d(A)})$*, where* $A \in N$ *and* $\gamma_i \in (N_c \cup \Sigma)^*$*. Each rule* p *satisfies the following condition.*
  - ***Right-linearity****:* $\forall A^i \in N_c \quad A^i$ *appears in the right-hand side (rhs) of* p *at most once.*

Definition 8. *Let* G *be a multiple context-free grammar (MCFG). The dimension of* G *is* $m = \max\limits_{A \in N}(d(A))$.

For example, $m = 1$ for the context-free grammars.

We use the conventional notation $A \overset{*}{\underset{G}{\Longrightarrow}} (w_1, \ldots, w_{d(A)})$, where $\forall i : w_i \in \Sigma^*$, to denote, that a tuple of words can be derived from a nonterminal A by some sequence of production rule applications from P in grammar G if $\forall i : w_i$ can be derived from $i - th$ component of a nonterminal A.

Definition 9. *A* multiple context-free language *(MCFL) is a language generated by a multiple context-free grammar* $G = (N, \Sigma, P, S, d)$*:*

$$L(G) = \{w \in \Sigma^* \mid S \overset{*}{\underset{G}{\Longrightarrow}} w\}.$$

Also, we introduce the following normal form for the MCFGs that allow us to solve the MCFL-reachability problem using the LA operations.

Definition 10. *A multiple context-free grammar* $G = (N, \Sigma, P, S, d)$ *is in* normal form *if every production rule* $p : A \to (\gamma_1, \ldots, \gamma_{d(A)}) \in P$ *satisfies one of two forms:*

- $\forall i : |\gamma_i| = 1$ *and* $\gamma_i \in \Sigma \cup \{\varepsilon\}$*. In this case, we call the rule* p *terminating,*
- $\forall i : \gamma_i \in N_c^*$*, i.e. no terminal symbol appears in the rhs of* p*. For simplicity of notation, we denote* $p : A \to f(B_1, \ldots, B_n)$*, where* $B_k \in N$*. It is necessary that* $n = 2$*. To sum up,* $p : A \to f(B_1, B_2)$*, where* $B_1, B_2 \in N$*. In this case, we call the rule* p *nonterminating. Any rule in this form must satisfy the following conditions.*
  - ***Non-erasing condition****:*$\forall i \in \{1, 2\}$ $B_i^j \quad 1 \leq j \leq d(B_i)$ *appears in* γ_k *for some* k*,*
  - *no pair* B^j, B^k *of component symbols of the same nonterminal appear adjacently in the rhs in one component, i.e. component symbols of* B_1, B_2 *appear alternately in one component,*
  - $\exists i : 1 \leq i \leq d(A) \quad |\gamma_i| \geq 2$.

According to [13], the following theorem holds.

Theorem 1 *Let* G *be an MCFG. An MCFG* G' *can be constructed from* G *such that* $L(G') = L(G)$ *and* G' *satisfies described normal form.*

For example, for $m = 1$ the normal form defined above is the same as the weak Chomsky normal form for context-free grammars described in [1].

Let us consider the MCFG $G_1 = (N_1, \Sigma_1, P_1, S, d_1)$, where $N_1 = \{S, A, B\}$ ($d_1(S) = 1$, $d_1(A) = d_1(B) = 2$), $\Sigma_1 = \{a, b, c, d\}$, $P_1 = \{$

$$\begin{aligned} S^1 &\to A^1 B^1 A^2 B^2 \\ (A^1, A^2) &\to (a, c) \\ (B^1, B^2) &\to (b, d) \\ (B^1, B^2) &\to (bB^1, dB^2) \\ (A^1, A^2) &\to (aA^1, cA^2) \end{aligned}$$

$\}$.

This grammar generates one of the classical MCFLs: $L(G_1) = \{a^n b^m c^n d^m \mid n, m \in \mathbb{N}\}$ that is known not to be context-free.

A grammar G_1' in normal form can be constructed from G, where $L(G_1') = L(G_1)$ and $N_1' = \{S, S_1, S_2, A, B, C, D\}$, $P_1' = \{$

$$\begin{aligned} A^1 &\to a \\ B^1 &\to b \\ C^1 &\to c \\ D^1 &\to d \\ (S_1^1, S_1^2) &\to (a, c) \\ (S_2^1, S_2^2) &\to (b, d) \\ (S_1^1, S_1^2) &\to (S_3^1, S_3^2 C^1) \\ (S_3^1, S_3^2) &\to (A^1 S_1^1, S_1^2) \\ (S_2^1, S_2^2) &\to (S_4^1, S_4^2 D^1) \\ (S_4^1, S_4^2) &\to (B^1 S_2^1, S_2^2) \\ S^1 &\to S_1^1 S_2^1 S_1^2 S_2^2 \end{aligned}$$

$\}$.

2.3 MCFL-reachability problem

Definition 11. *Let $D = (V, E, \Sigma)$ be a labeled graph, L be an MCFL. Then a* multiple context-free relation *with language L on the labeled graph D is the relation $R_{D,L} \subseteq V \times V$:*

$$\begin{aligned} R_{D,L} = \{&(v_0, v_n) \in V \times V \mid \\ &\exists \pi = ((v_0, l_1, v_1), \ldots, (v_{n-1}, l_n, v_n)) \in \pi(D) : \\ &l(\pi) \in L\}. \end{aligned}$$

Finally, we can define MCFL-reachability problem.

Definition 12. MCFL-reachability *is the problem of finding multiple context-free relation $R_{D,L}$ for a given directed labeled graph D and a multiple context-free language L.*

In other words, the result of MCFL-reachability evaluation is a set of vertex pairs such that there is a path between them that forms a word from the given MCFL.

Next, we build an algorithm to solve MCFL-reachability problem.

3 Matrix-based MCFL-reachability algorithm

In this section, we introduce the matrix-based algorithm for the MCFL-reachability problem.

3.1 Algorithm

Definition 13. *Let $G = (N, \Sigma, P, S, d)$ be an MCFG. Then $\forall p : A \to f(B, C) \in P$ we define:*

- *$end\_B(p) = \{2(i-1) |$ iff B^i is the leftmost in the component of rhs of $p\} \cup \{2(i-1)+1 |$ iff B^i is the rightmost in the component of rhs of $p\}$, each of the sets is considered ordered by the components of the A and inside the component from left to right;*
- *$end\_C(p)$ defined similarly with $end\_B(p)$ but with an offset of $2d(B)$, that is, $end\_C(p)$ will be of the form $2(i-1) + 2d(B)$ and $2(i-1) + 1 + 2d(B)$;*
- *end_A is ordered set of pairs where the first element of the pair corresponds to the leftmost nonterminal of some component in the rule and the second element — to the rightmost, i.e. the elements of pairs are elements of the sets $end\_B(p)$ and $end\_C(p)$ defined above;*
- *$(2(i-1)) \in alter\_B(p)$ or $(2(i-1)+1) \in alter\_B(p)$ iff $(2(i-1)) \notin end\_B(p)$ or $(2(i-1)+1) \notin end\_B(p)$, $\forall 1 \leq i \leq d(B)$;*
- *$alter\_C(p)$ defined similarly with $alter\_B(p)$ but with an offset of $2d(B)$.*

Note: $|alter\_B(p)| = |alter\_C(p)|$ for grammars in the described normal form.

Let $G = (N, \Sigma, P, S, d)$ be an MCFG, $D = (V, E, \Sigma)$ — labeled directed graph, where $|V| = n$. The proposed algorithm is presented in Listing 1. The $MCFL\_rechability$ procedure takes as input a graph and a multiple context-free grammar in *normal form*.

At the first stage, the algorithm processes rules where there are only terminals on their rhs. Thus, the algorithm restores the paths that can be obtained in one application of the rule. The *update* procedure is presented in Listing 2. It is used to update all necessary matrices for rules with nonterminal B in rhs with a new value according to the new paths found. In the update procedure, only the index of the value is recalculated, taking into account the sets $end\_B(p)$, $end\_C(p)$, $alter\_B(p)$ and $alter\_C(p)$ and the value $True$ is added according to the calculated index.

In line 6 of Listing 1 the $d(A)$ paths of length 1 or 0 corresponding to the components of the rule $A \to (a_1, \ldots, a_{d(A)})$ are found. That is, each (l_i, r_i) is a pair of vertices between which there is a path derived from the i-component of the rule. We note two facts about this index. First of all, there are $d(A)$ pairs in this index, that is, the number of elements in it is even. Second, such index can be encoded as a $(n+1)-ary$ number (n — number of vertices in the graph D). The second fact allows us to use the $FromIndex$ algorithm ($ToIndex$ inverse to it) to convert such a number to the $(n+1)-ary$ numeral system. The parity of the number of elements allows us to divide the index in half and write the first part in the row number of the matrix and the second part in the column number. Thus, we write the fact of the restored paths for the nonterminal into a square Boolean matrix by dividing the index in half and translating each part into the desired numeral system (let these numbers be i and j), and then put $True$ value in the cell (i, j).

Listing 1 MCFL-reachability algorithm

```
1: G = (N, Σ, P, S, d) — MCFG, D = (V, E, Σ) — directed edge-labeled graph
2: A_p, B_p, B_p_new, C_p, C_p_new ← Boolean matrices of proper size
3: procedure MCFL_reachability(G, D)
4:     for p ∈ P : p = A → (a_1, ..., a_d(A)) do
5:         for (l_1, a_1, r_1), ..., (l_d(A), a_d(A), r_d(A)) ∈ E do
6:             index ← (l_1, r_1, l_2, ..., r_d(A))
7:             update(A, index)                ▷ add information about terminating rules
8:     for p ∈ P : p = A → f(B, C) do
9:         B_p_new, C_p_new ← B_p, C_p         ▷ all values are new for the first iteration
10:    while matrices B_p, C_p are changing do
11:        for p ∈ P : p = A → f(B, C) do      ▷ consider nonterminating rules
12:            A_p_new ← B_p_new × C_p + B_p × C_p_new      ▷ use only new values
13:            B_p_new, C_p_new ← empty Boolean matrices of proper size
14:            for (i, j) : A_p_new[i, j] = True ∧ A_p[i, j] = False do
15:                A_p[i, j] ← True
16:                index ← transform_index(ToIndex(i), ToIndex(j), p)
17:                update(A, index)            ▷ add new information for nonterminal A
18:    Res ← Boolean matrix of proper size
19:    for p ∈ P : p = S → f(B, C) do          ▷ collect all information for the start
       nonterminal S
20:        for (i, j) : S_p[i, j] = True do
21:            index ← transform_index(ToIndex(i), ToIndex(j), p)   ▷ size of index
       is equal to 2 since d(S) = 1
22:            Res[index[0], index[1]] ← True
       return Res
```

Listing 2 The procedure for updating matrix values

```
1: procedure update(B, index)      ▷ update matrices for all rules with B in the rhs
2:     for p ∈ P : p = A → f(B, C) do
3:         i_B, j_B ← empty lists
4:         for end ∈ end_B(p) do
5:             i_B.append(index[end])
6:         for alter ∈ alter_B(p) do
7:             j_B.append(index[alter])
8:         B_p[FromIndex(i_B), FromIndex(j_B)] ← True
9:     for p : A → f(C, B) do
10:        i_B, j_B ← empty lists
11:        for alter ∈ alter_B(p) do
12:            i_B.append(index[alter − 2d(C)])
13:        for end ∈ end_B(p) do
14:            j_B.append(index[end − 2d(C)])
15:        B_p[FromIndex(i_B), FromIndex(j_B)] ← True
```

Further, the algorithm uses five matrices for each nonterminal rule. Namely, for a rule p of the form $A \to f(B, C)$, the algorithm supports the matrix B_p in which the result is stored, taking into account the sets in Definition 13 for the nonterminal B, as well as the matrix B_p_new, which stores the result, taking into account the sets, which was obtained only at the previous step. Similarly for the nonterminal C. Also, the information about found paths corresponding to this rule is stored in matrix A_p, taking into account the set $end\_A(p)$. And after processing the terminating rules, it is necessary to add new results to the supported matrices. This is exactly what the algorithm in lines 8-9 does.

Next, the algorithm proceeds to the consideration of nonterminating rules. Namely, in line 12, the algorithm calculates new paths in the graph using four matrices for nonterminals from the rhs of the rule. Further, the algorithm update the matrices B_p_new and C_p_new to store only new values that was added at this iteration. Also, algorithm writes only new values to the matrix A_p for the nonterminal A and propagate new results among all matrices for the nonterminal A in the rhs of other rules.

The algorithm works while at least one value has appeared on the current iteration using the loop in lines 10-17. As the last step, the algorithm collects values from all rules where there is the starting nonterminal on the left-hand side (lhs) and puts these values into the matrix Res for the MCFL-reachabilty result. The indices must be recalculated using the $transform\_index$ procedure presented in Listing 3.

Listing 3 The procedure for index transformation

```
1: procedure transform_index(i, j, p)  ▷ transform the indices i and j taking into
   account the set end_A(p)
2:     A ← the nonterminal in the lhs of p
3:     index ← empty list
4:     for (end_l, end_r) ∈ end_A(p) do
5:         if end_l < 2d(B) then
6:             pos ← position end_l in end_B(p)
7:             index.append(i[pos])
8:         else
9:             pos ← position end_l in end_C(p)
10:            index.append(j[pos])
11:        if end_r < 2d(B) then
12:            pos ← position end_r in end_B(p)
13:            index.append(i[pos])
14:        else
15:            pos ← position end_r in end_C(p)
16:            index.append(j[pos])
       return index
```

3.2 Example

Let us consider some steps of the algorithm using an example. As input we take the graph D_1 and multiple context-free grammar G_1' from Section 2.

At the first step, the algorithm process the terminating rules of G_1 (lines 4-7 in listing 1). Also, the algorithm updates the matrices with the *_new* suffix for the nonterminals in the rhs of each rule. Consider, for example, what the B_p*_new* and C_p*_new* for the rule $S^1 \to S_1^1 S_2^1 S_1^2 S_2^2$ looks like after these updates. For this rule, the algorithm uses sets from the Definition 13 that look as follows.

- $end\_B = \{0\}$
- $alter\_B = \{1, 2, 3\}$
- $end\_C = \{7\}$
- $alter\_C = \{4, 5, 6\}$
- $end\_A = \{(0, 7)\}$

$$B_p = B_p\_new = \begin{array}{c} \\ (1) \\ (5) \end{array} \begin{array}{c} \begin{array}{cccc} (2,3,2) & (2,3,4) & (4,3,2) & (4,3,4) \end{array} \\ \begin{pmatrix} True & True & . & . \\ . & . & True & True \end{pmatrix} \end{array}$$

$$C_p = C_p\_new = C_p = \begin{array}{c} \\ (2,3,2) \\ (4,3,2) \\ (2,3,4) \\ (4,3,3) \end{array} \begin{array}{c} \begin{array}{cc} (1) & (5) \end{array} \\ \begin{pmatrix} True & . \\ . & True \\ True & . \\ . & True \end{pmatrix} \end{array}$$

At the second step, the algorithm processes nonterminating rules (lines 10-17 in listing 1). Consider the rule with nonterminal S in the lhs. First, the algorithm multiplies the matrices B_p, B_p_new, C_p and C_p_new in the specified order (line 12 in listing 1). As a result, the matrix A_p_new is computed of the following form. Note that we omit rows and columns with all zero values.

$$A_p\_new = \begin{matrix} \\ (1) \\ (5) \end{matrix} \begin{matrix} \begin{matrix} (1) & (5) \end{matrix} \\ \begin{pmatrix} True & True \\ True & True \end{pmatrix} \end{matrix}$$

Next, the algorithm will try to propagate the information and consider other rules. However, no other paths will be obtained further. At the last stage, the algorithm builds a matrix for the starting nonterminal, which in this case will coincide with the previous matrix.

$$Res = A_p = \begin{matrix} \\ (1) \\ (5) \end{matrix} \begin{matrix} \begin{matrix} (1) & (5) \end{matrix} \\ \begin{pmatrix} True & True \\ True & True \end{pmatrix} \end{matrix}$$

Thus, the algorithm received information that there are paths that satisfy the grammar G_1 from vertex 1 to 5, from 5 to 1, from 1 to 1, and from 5 to 5. Therefore, the multiple context-free relation $R_{D_1,L(G_1)} = \{(1,5),(5,1),(1,1),(5,5)\}$.

3.3 Correctness and complexity

Similarly to the proof of the correctness of the matrix-based CFL-reachability algorithm from [2], it can be shown that the following theorem holds by the induction on the iteration number and on the height of derivation trees.

Theorem 2 *Let $G = (N, \Sigma, P, S, d)$ be an MCFG in* normal form*, $D = (V, E, \Sigma)$ be an labeled directed graph and Res matrix obtained as a result of the algorithm (listing 1). Then $(v_0, v_n) \in R_{D,L(G)}$ iff $Res[v_0, v_n] = True$.*

The most critical part of the algorithm is the evaluation of new values for $B_p \times C_p$ in line 13. Next, we will estimate the time needed for computing $B_p \times C_p$. Assume the sizes of Boolean matrices B_p and C_p are $(n+1)^q \times (n+1)^r$ and $(n+1)^r \times (n+1)^s$, respectively. Then for every nonterminating rule $p : A \to f(B, C)$ we define the *degree* of this rule $e(p) = d(A) + d(B) + d(C)$. Also, we define the multiplication unit of p as follows.

Definition 14. *Let $G = (N, \Sigma, P, S, d)$ be an MCFG in* normal form*. For every nonterminating rule $p : A \to f(B, C)$, define the number $i(p)$, called the multiplication unit of p, as $d(A) + d(B) + d(C) - 2 \cdot max\{d(A), d(B), d(C)\}$.*

According to [13], the $B_p \times C_p$ operations can be computed in $O(n^{e(p)-0.624i(p)})$. Now we must estimate the number of iterations of the algorithm in listing 1. In [13] the algorithm solves the MCFL recognition problem, and only n iterations

are needed. Thus, the MCFL recognition algorithm in [13] computes the result in time $O(n^{e(p)-0.624i(p)+1})$. In our MCFL-reachability algorithm, we may need to make more iterations. The matrices B_p (or C_p) can only change if a new tuple of paths $(\pi_1, \ldots, \pi_{d(B)})$ was found, such that $B \overset{*}{\underset{G}{\Longrightarrow}} (l(\pi_1), \ldots, l(\pi_{d(B)}))$. Such tuples of paths are described in the algorithm using the indices $(l_1, r_1, l_2, \ldots, r_{d(B)})$ where $\forall 1 \leq i \leq d(B)$, π_i — is a path from vertex l_i to vertex r_i. In the worst case, exactly one new index will appear on each iteration. Thus, the number of iterations does not exceed the number of distinct indices. Let m be the dimension of the MCFG G. Then the number of distinct indices is $O(n^{2m})$. Therefore, the following theorem holds.

Theorem 3 *Let $G = (N, \Sigma, P, S, d)$ be an MCFG in* normal form *with the dimension m, $D = (V, E, \Sigma)$ be an labeled directed graph. Let p' be a rule such that $e(p') - 0.624i(p') = max\{e(p) - 0.624i(p) \mid p \in P\}$, and let $e' = e(p')$, $i' = i(p')$. The algorithm in listing 1 computes the result in $O(|V|^{e'-0.624i'+2m})$.*

This is the first naive time complexity bound for MCFL-reachabilty problem. For example, for the grammar G'_1 from Section 2, the $e' = 5$, the $i' = 1$, and the $m = 2$. Thus the time complexity bound for this grammar is $O(|V|^{8,376})$. However, $O(n^{2m})$ iterations can be reached only on special artificial graphs, and usually, on real-world graphs the number of iterations is small. Also, there are known techniques for improving this complexity bound by recursive multiplication of submatrices [21]. Finally, the complexity bound can be improved by taking into account the sparsity of matrices and the sparse matrix operation.

4 Evaluation

We provide a prototype implementation of the proposed MCFL-reachability algorithm using the pygraphblas implementation of the GraphBLAS API. The source code is available on GitHub[4]. For evaluation, we use a PC with Ubuntu 18.04 installed. It has Intel core i7-6700 CPU, 3.4GHz, and DDR4 64Gb RAM.

4.1 Dataset Description

We evaluate our implementation on some real-world RDFs and synthetic graphs using following classical MCFLs [6]. The query Q_1 corresponds to the MCFL $L_1 = \{a^n b^m c^n d^m \mid n, m \in \mathbb{N}\}$ that was discussed in Section 2, and the query Q_2 corresponds to the MCFL $L_2 = \{b^n a^m b^n \mid n, m \in \mathbb{N}\}$. These languages are known to be not context-free [17].We use some RDFs from the CFPQ_Data dataset[5] provided in [20]. Also, we generate some synthetic graphs that describe

[4] Sources of the prototype implementation of the proposed MCFL-reachability algorithm: https://github.com/JetBrains-Research/CFPQ_PyAlgo. Access date: 06.04.2022.

[5] CFPQ_Data dataset GitHub repository: https://github.com/JetBrains-Research/CFPQ_Data. Access date: 06.04.2022.

network structures using the LFR graph generator from the NetworkX [7] Python package.

4.2 Evaluation Results

The results of the MCFL-reachability evaluation for queries Q_1 and Q_2 are presented in Table 1. We can see, that while the execution time for small graphs is decent, our prototype implementation is underperforming for graphs with thousands of vertices. To show the reason for such behavior, we also present the number of non-zero elements in matrices B_p and C_p added by the MCFL-reachability algorithm in Table 2. The NNZ_1 is a sum of non-zero elements in these matrices for the query Q_1, and the NNZ_2 — for the query Q_2. This information describes the big amount of consumed memory and the complexity of the used matrix operations. Our implementation is underperforming for used graphs and queries because there is a big number of combinations of paths that are relevant to the query, and was founded by our algorithm. The used graphs are composed entirely of edges that are relevant to the query, and they form such a large number of combinations. For more practical use of our algorithm, the huge graphs can contain only a small part of the edges that are relevant to the query. This will lead to a decent number of the non-zero matrix elements, a small number of used memory, and a small running time. Also, for big graphs our algorithm constructs matrices of huge sizes that can exceed the numeric range of integer data types. However, this problem can be solved since there are formats like COO (coordinate list) for storing the sparse matrices that stores only non-zero values and does not depend on the matrix size.

Table 1: MCFL-reachability execution time in seconds for queries Q_1 and Q_2

	Graph	#V	#E	Q_1	Q_2		Graph	#V	#E	Q_1	Q_2
RDF	skos	144	323	0.07	0.08		pathways	6,238	37,196	533.73	148.62
	pizza	671	2,604	3.75	2.06	LFR	LFR_{100}	100	210	0.12	0.06
	wine	733	2,450	4.55	4.41		LFR_{500}	500	970	2.55	1.47
	funding	778	1,480	1.68	1.50		LFR_{1000}	1,000	2,100	13.50	6.10
	core	1,323	8,684	10.77	9.93		LFR_{10000}	10,000	21,005	1,261.97	656.28

We conclude that we should improve our implementation to achieve better performance, and find the graphs and queries for more practical use, while the algorithm idea is viable.

5 Conclusion

In this paper, we propose the first MCFL-reachability algorithm by extending the MCFL parsing algorithm from [13]. Thus, we show how the MCFL-reachability problem can be solved using the LA operations. We implement the proposed

Table 2: The number of non-zero elements in matrices after the MCFL-reachability evaluation

	Graph	#E	NNZ_1	NNZ_2		Graph	#E	NNZ_1	NNZ_2
RDF	skos	323	5,043	5,312		pathways	37,196	19,452,226	9,734,396
	pizza	2,604	201,554	134,993	LFR	LFR_{100}	210	5,687	2,851
	wine	2,450	252,323	241,485		LFR_{500}	970	118,449	63,786
	funding	1,480	101,304	94,401		LFR_{1000}	2,100	553,002	276,299
	core	8,684	531,900	503,943		LFR_{10000}	21,005	55,172,204	27,349,209

algorithm using the GraphBLAS API. We should improve our prototype implementation using various high-performance libraries for the LA operations, distributed computations, and modern parallel hardware like GPU. Also, we should find the real graphs and queries for more practical use of our algorithm, for example from the area of static code analysis. Further research is also needed to improve the time complexity bound $O(|V|^{e'-0.624i'+2m})$ by using the recursive multiplication of submatrices and the sparse matrix operations.

References

1. Azimov, R., Epelbaum, I., Grigorev, S.: Context-free path querying with all-path semantics by matrix multiplication. In: Proceedings of the 4th ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA). pp. 1–7 (2021)
2. Azimov, R., Grigorev, S.: Context-free path querying by matrix multiplication. In: Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA). pp. 1–10 (2018)
3. Barrett, C., Jacob, R., Marathe, M.: Formal-language-constrained path problems. SIAM Journal on Computing **30**(3), 809–837 (2000). https://doi.org/10.1137/S0097539798337716, https://doi.org/10.1137/S0097539798337716
4. Bastani, O., Anand, S., Aiken, A.: Specification inference using context-free language reachability. In: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 553–566 (2015)
5. Cohen, S.B., Gildea, D.: Parsing linear context-free rewriting systems with fast matrix multiplication. Computational Linguistics **42**(3), 421–455 (2016)
6. Götzmann, D.N.: Multiple context-free grammars https://www.ps.uni-saarland.de/courses/seminar-ws06/papers/04_daniel_goetzmann.pdf
7. Hagberg, A., Swart, P., S Chult, D.: Exploring network structure, dynamics, and function using networkx. Tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States) (2008)
8. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation. Acm Sigact News **32**(1), 60–65 (2001)
9. Huang, W., Dong, Y., Milanova, A., Dolby, J.: Scalable and precise taint analysis for android. In: Proceedings of the 2015 International Symposium on Software Testing and Analysis. pp. 106–117 (2015)

10. Kepner, J., Aaltonen, P., Bader, D., Buluç, A., Franchetti, F., Gilbert, J., Hutchison, D., Kumar, M., Lumsdaine, A., Meyerhenke, H., et al.: Mathematical foundations of the graphblas. In: 2016 IEEE High Performance Extreme Computing Conference (HPEC). pp. 1–9. IEEE (2016)
11. Kodumal, J., Aiken, A.: The set constraint/cfl reachability connection in practice. ACM Sigplan Notices **39**(6), 207–218 (2004)
12. Miao, H., Deshpande, A.: Understanding data science lifecycle provenance via graph segmentation and summarization. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE). pp. 1710–1713 (2019)
13. Nakanishi, R., Takada, K., Seki, H.: An efficient recognition algorithm for multiple context-free languages. In: In Proceedings of the Fifth Meeting on Mathematics of Language, MOL5. Citeseer (1997)
14. Rehof, J., Fähndrich, M.: Type-base flow analysis: From polymorphic subtyping to cfl-reachability. SIGPLAN Not. **36**(3), 54–66 (Jan 2001). https://doi.org/10.1145/373243.360208, https://doi.org/10.1145/373243.360208
15. Reps, T.: Program analysis via graph reachability. Information and software technology **40**(11-12), 701–726 (1998)
16. Reps, T.: Undecidability of context-sensitive data-dependence analysis. ACM Transactions on Programming Languages and Systems (TOPLAS) **22**(1), 162–186 (2000)
17. Seki, H., Matsumura, T., Fujii, M., Kasami, T.: On multiple context-free grammars. Theoretical Computer Science **88**(2), 191–229 (1991)
18. Sevon, P., Eronen, L.: Subgraph queries by context-free grammars. Journal of Integrative Bioinformatics **5**(2), 157 – 172 (2008). https://doi.org/https://doi.org/10.1515/jib-2008-100, https://www.degruyter.com/view/journals/jib/5/2/article-p157.xml
19. Sridharan, M., Bodík, R.: Refinement-based context-sensitive points-to analysis for java. ACM SIGPLAN Notices **41**(6), 387–400 (2006)
20. Terekhov, A., Khoroshev, A., Azimov, R., Grigorev, S.: Context-free path querying with single-path semantics by matrix multiplication. In: Proceedings of the 3rd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA). GRADES-NDA'20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3398682.3399163, https://doi.org/10.1145/3398682.3399163
21. Valiant, L.G.: General context-free recognition in less than cubic time. Journal of computer and system sciences **10**(2), 308–315 (1975)
22. Yan, D., Xu, G., Rountev, A.: Demand-driven context-sensitive alias analysis for java. In: Proceedings of the 2011 International Symposium on Software Testing and Analysis. pp. 155–165 (2011)
23. Zhang, Q., Lyu, M.R., Yuan, H., Su, Z.: Fast algorithms for dyck-cfl-reachability with applications to alias analysis. In: Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 435–446 (2013)
24. Zhang, Q., Su, Z.: Context-sensitive data-dependence analysis via linear conjunctive language reachability. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages. pp. 344–358 (2017)
25. Zhang, X., Feng, Z., Wang, X., Rao, G., Wu, W.: Context-free path queries on rdf graphs. In: Groth, P., Simperl, E., Gray, A., Sabou, M., Krötzsch, M., Lecue, F., Flöck, F., Gil, Y. (eds.) The Semantic Web – ISWC 2016. pp. 632–648. Springer International Publishing, Cham (2016)

26. Zheng, X., Rugina, R.: Demand-driven alias analysis for c. In: Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 197–208. POPL '08, ACM, New York, NY, USA (2008). https://doi.org/10.1145/1328438.1328464, http://doi.acm.org/10.1145/1328438.1328464
27. Zheng, X., Rugina, R.: Demand-driven alias analysis for c. In: Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. pp. 197–208 (2008)