Context-Free Path Querying via Real Matrix Equations

Yuliya Susanina
Saint Petersburg State University
St. Petersburg, Russia
st049970@student.spbu.ru

Semyon Grigorev
Saint Petersburg State University
St. Petersburg, Russia
s.v.grigoriev@spbu.ru

ABSTRACT

Context-free path querying is reduced to the problem of solving a system of matrix equations over $\mathbb{R}$.

CCS CONCEPTS

• **Information systems → Query languages**; • **Theory of computation → Formal languages and automata theory**; • **Design and analysis of algorithms → Approximation algorithms analysis**.

KEYWORDS

context-free path querying, graph databases, context-free grammar, nonlinear matrix equations, newton method

ACM Reference Format:
Yuliya Susanina and Semyon Grigorev. 2018. Context-Free Path Querying via Real Matrix Equations. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Context-free path querying (CFPQ) is becoming more popular in many different areas, for example, bioinformatics [1], graph databases [5] or static code analysis [8], etc. A query is presented as a context-free grammar and all possible paths (strings) are a set with a directed labeled graph. This type of query increases the expressive power of commonly used regular expressions and therefore forms a promising research area. The computation of such queries is a graph parsing with respect to the given grammar. The result of the CFPQ evaluation is a set of triples (A, m, n), such that a path from the node m to the node n exists in the given graph and a string obtained by concatenating the labels of the edges along this path can be derived from nonterminal A of the given context-free grammar.

For most CFPQ application areas a large amount of data is common, so an especially pronounced problem is search and development of high-performance algorithms. The matrix-based algorithms, for example, [3], are the most potential for practical tasks. The possibility of applying various computing techniques to speed up matrix calculations, such as GPGPU or sparse matrix representation allows to improve the performance on the real-world data [6]. However, these algorithms, just like the others, suffer from computational problem issues and can be low-speed in some cases.

There are also several reasons to consider the applicability of numerical linear algebra and computational mathematics for CFPQ. Firstly, the benefits of matrix-based CFPQ algorithms (e.g. the utilization of parallel techniques) will remain. Well-known linear algebra operations, such as matrix inversion or decomposition, can be used in the creation of better algorithms. For instance, there are several successful results of applying linear algebra methods to logic programming [2, 7]. Secondly, approximate computational methods can accelerate CFPQs processing. And most importantly, the popularity of artificial intelligence techniques pushed the development and improvement of many efficient libraries for numerical computing.

In this article, we modify the matrix-based algorithm mentioned above and reduce GFPQ evaluation to solving the systems of Boolean matrix equations. And then we propose a new approach for CFPQs processing, based on solving the systems of equations over $\mathbb{R}$. We also assess the feasibility of using both accurate and approximate methods of computational mathematics. The evaluation of our approach on a set of conventional benchmarks shows its practical applicability.

2 BACKGROUND

2.1 Preliminaries

Context-free grammar (CFG) is a quadruple $G = (N, \Sigma, R)$, where N is a set of nonterminal symbols, Σ is a set of terminal symbols and R is a set of productions of the followings form: $A \Rightarrow \alpha, \alpha \in (N \cup \Sigma)^*$. $\mathcal{L}(G_S)$ denotes a language specified by CFG G with respect to $S \in N$: $\mathcal{L}(G_S) = \{\omega \mid S \Rightarrow_G^* \omega\}$.

Directed graph is a triple $D = (V, E, \sigma)$, where V is a set of vertices, $\sigma \subseteq \Sigma$ is a set of labels, and a set of edges $E \subseteq V \times \sigma \times V$. Path p in graph D is a list of incident edges: $p = e_0, \ldots, e_{n-1}$, where $v_i \in V$, $e_i = (v_i, l_i, v_{i+1}) \in E$, $l_i \in \sigma$, $|p| = n, n \geq 1$. $P = \{p \mid p - \text{path in } D\}$. For $p \in P$, $trace(p)$ is the unique word, obtained by concatenating the labels of the edges along the path p.

For a graph D and a CFG G, we define *context-free relations* $R_A \subseteq V \times V$ for each $A \in N$: $R_A = \{p \mid p \in P,\ trace(p) \in \mathcal{L}(G_A)\}$.

2.2 Matrix-Based CFPQ Algorithm

Matrix-based algorithm, proposed by Rustam Azimov [3], processes CFPQs by using relational query semantics [4]. It constructs a parsing table T of size $|V| \times |V|$ for an input graph $D = (V, E)$ and grammar $G = (N, \Sigma, R)$ in Chomsky normal form. Each element of T contains the set of nonterminals such that $A \in T_{i,j} \iff \exists p \in R_A$.

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-9999-9/18/06...$15.00
https://doi.org/10.1145/1122445.1122456

For each vertices i and j we initialize $T_{i,j} = \{A \mid (i, a, j) \in E, A \to a \in R\}$. Then, the computations of parsing table T happens through the calculation of matrix transitive closure: $M^* = M^{(1)} \cup M^{(2)} \ldots$, where $M^{(1)} = M, M^{(k)} = M^{(k-1)} \cup (M^{(k-1)} \times M^{(k-1)})$ for $k > 1$. Also we can represent parsing table T as a set of Boolean matrices of size $|V| \times |V|$ for each $A \in N$. So, we can replace the computation of transitive closure $T = T \cup (T \times T)$ to several Boolean matrix multiplications $T_A = T_A + T_B T_C$ for each $A \to BC \in R$.

This algorithm can be effectively applied to real-world data with implementation based on parallel techniques and high-performance libraries [6], but it takes time $O(|N|^3|V|^2(BMM(|V|) + BMU(|V|)))$.

3 EQUATION-BASED APPROACH

In this section we transform matrix-based algorithm. We replace the calculation of matrices' products by the computation of several matrix equations.

3.1 From Iteration to Equations

The main difference of our approach is that we do not transform the input grammar to Chomsky normal form, as the matrix-based algorithm requires. However, the elimination of ε-rules is still needed.

For each $X \in (N \cup \Sigma)$ we create a Boolean matrix T_X. All matrices corresponding to nonterminals are filled in accordance with an input graph: $((T_x)_{ij} = 1 \iff (v_i, x, v_j) \in E)$ for $x \in \Sigma$, and cannot be changed.

Let's consider a simple grammar $D_1 : S \to aSb \mid ab$ and its representation in Chomsky normal form: $D_1 : S \to AS_1 \mid AB$; $S_1 \to SB$; $A \to a$; $B \to b$. In the original algorithm on each step of the while loop Boolean matrices is changing in 3 calculations (rules $A \to a$ and $B \to b$ are not used after matrices initialization). It can be replaced into a simple iterative process:

$$\left.\begin{array}{l} T_{S_1} = T_{S_1} + T_S T_b \\ T_S = T_S + T_a T_{S_1} \\ T_S = T_S + T_a T_b \end{array}\right\} \Rightarrow \begin{array}{l} T_S^0 = 0 \\ T_S^{k+1} = T_a T_S^k T_b + T_a T_b \end{array}$$

$\{T_S^k\}$ is a monotonically increasing series of Boolean matrices. It converges and the limit T_S^* is the least solution of Boolean matrix equation:

$$T_S = T_A T_S T_B + T_A T_B$$

Unfortunately, this modification does not entail any meaningful improvements. But we can consider another equation over $\mathbb{R}$:

$$\mathcal{T}_S = \epsilon(T_A \mathcal{T}_S T_B + T_A T_B)$$

And the corresponding matrix series $\{\mathcal{T}_S^k\}$, which converges to $\mathcal{T}_S^*$ when $\mathcal{T}_S^k \leq \mathbf{1}$ as a a monotonically increasing series of matrices with an upper bound:

$$\mathcal{T}_S^0 = 0$$
$$\mathcal{T}_S^{k+1} = \epsilon(T_A \mathcal{T}_S^k T_B + T_A T_B).$$

It can be proved that $(\mathcal{T}_S^{k+1})_{ij} > 0 \iff (T_S^{k+1})_{ij} = 1$ and $ceil(\mathcal{T}_S^*) = T_S^*$, where $ceil$ returns the smallest integer not less than x.

So, each rule of the following form $X \to V \ldots W \mid \cdots \mid Y \ldots Z$, where $X \in N, V, W, \ldots, Y, Z \in (N \cup \Sigma)$ can be replaced by equation: $\mathcal{T}_X = \epsilon_X(T_V \cdot \ldots \cdot T_W + \cdots + T_Y \cdot \ldots \cdot T_Z)$, where ϵ_X is chosen such that $\mathcal{T}_X^k \leq \mathbf{1}$ for each k.

3.2 Linear Equations

If our CFG is linear, then the most difficult case is the one presented as an example in the previous subsection. We can also solve it as a Sylvester equation in $O(|V|^3)$, but only for one type of brackets. Otherwise it can be reduced to solving a linear system $Ax = b$, where A is a matrix of size $(|V|^2) \times (|V|^2)$ and time required to compute its solution is $O(|V|^6)$ or $O(|V|^{4\omega+2})$ with more efficient matrix multiplication algorithms. The use of sparse matrix representation can be very efficient for solving the equations of this type.

3.3 Nonlinear Equations

We can rewrite our equation of the form $X = \Psi(X)$ to the equivalent $F(X) = X - \Psi(X) = 0$ and use Newton's method for nonlinear functions root finding:

$$F(X) = 0, X_0$$
$$X_{i+1} = X_i - (F'(X_i))^{-1}F(X_i) \quad \iff \quad \begin{cases} F'(X_i)H_i = -F(X_i) \\ X_{i+1} = X_i + H_i \end{cases}$$

Here X_0 is an initial guess, in our case $X_0 = \mathbf{0}$, as our solution is a matrix consists of small positive numbers. The convergence of this method can be quadratic which allows finding the solution significantly faster. Even as it is necessary to solve an equation for H_i on each iteration step, the majority of high-performance implementations do not compute the Jacobian inverse and use its approximate value.

The main difficulty in using Newton's method is choosing an appropriate ϵ, to ensure the least positive solution for nonlinear equations ϵ must be smaller than $\frac{1}{|V|}$.

3.4 Systems of Equations

Until now, we consider only cases with one equation, but mostly we will deal with the systems of matrix equations. We construct the dependency graph D_G for nonterminals of the given grammar G and split the set of the equations into the disjoint subsets accordingly to the set of strongly connected components in D_G, which can be found in $O(|V| + |E|)$. So we can solve our system in stages without any troubles.

4 EVALUATION

The equation-based approach for CFPQ was implemented. We evaluated **Query 2** from [3]. The equation constructed for this query were resolved by two different ways using Python package *scipy*: **sSLV** — solving as a sparse linear system using *spsolve* and **dNWT** — funding roots of a function using *optimize.newton_krylov*.

We compare the results with the first matrix-based algorithm implementations described in [3] (Table 1). Our approach demonstrates that it can be applied on real-world data as well as the matrix-based algorithm. Moreover, we can improve the performance by the utilization of parallel techniques for matrix operations.

5 CONCLUSION AND FUTURE WORK

We proposed a new approach for CFPQs processing, based on solving the systems of equations over $\mathbb{R}$. The evaluation of our approach on a set of conventional benchmarks showed its practical applicability on real-world data.

Table 1: Evaluation results for Query 2 (in ms)

Ontology	dGPU	sCPU	dNWT	sSLV	sGPU
skos	10	2	5	7	1
generations	9	2	0	5	0
travel	31	7	51	5	10
univ-bench	55	15	40	8	9
atom	36	9	40	27	2
bio-meas	276	91	284	35	24
foaf	53	14	26	16	3
people-pets	144	38	73	49	6
funding	1246	344	502	184	27
wine	722	179	791	171	6
pizza	943	256	334	161	23

The directions for future research are high-performance implementation using GPGPU or other parallel techniques. Also, we plan to examine the special cases of the reduction of solving the systems of matrix equations to CFPQ.

ACKNOWLEDGMENTS

The research was supported by the Russian Science Foundation grant 18-11-00100 and a grant from JetBrains Research.

REFERENCES

[1] 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.

[2] Yaniv Aspis. 2018. *A Linear Algebraic Approach to Logic Programming.* Ph.D. Dissertation. Imperial College London.

[3] Rustam Azimov and Semyon Grigorev. 2018. 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).* ACM, 5.

[4] Jelle Hellings. 2015. Querying for paths in graphs using context-free path queries. *arXiv preprint arXiv:1502.02242* (2015).

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

[6] Nikita Mishin, Iaroslav Sokolov, Egor Spirin, Vladimir Kutuev, Egor Nemchinov, Sergey Gorbatyuk, and Semyon Grigorev. 2019. Evaluation of the Context-Free Path Querying Algorithm Based on Matrix Multiplication. In *Proceedings of the 2nd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA).* ACM, 12.

[7] Taisuke Sato. 2017. A linear algebraic approach to Datalog evaluation. *Theory and Practice of Logic Programming* 17, 3 (2017), 244–265.

[8] 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.