**УДК** 004.421.2:519.17 004.657

**Алгоритм поиска всех путей в графе с заданными контекстно-свободными ограничениями на основе матричного умножения**

**Рустам Шухратуллович Азимов, Семён Вячеславович Григорьев**

**Аннотация**

**Предмет исследования.** Рассмотрена задача поиска путей в графе, удовлетворяющих заданным контекстно-свободным ограничениям. Данная задача заключается в поиске всех путей в помеченном ориентированном графе, метки на рёбрах которых образуют слова из языка, порождённого входной контекстно-свободной грамматикой. Существует два наиболее эффективных подхода к решению данной задачи с использованием операций линейной алгебры: с использованием обычного матричного умножения и с использованием произведения Кронекера. Но до сих пор не существует алгоритма, использующего обычное матричное умножение, способного найти все пути, удовлетворяющие заданным контекстно-свободным ограничениям. В работе предложен алгоритм поиска всех путей в графе, удовлетворяющих заданным контекстно-свободным ограничениям, который основан на обычном матричном умножении. **Метод.** В матрицу смежности входного графа для каждой пары вершин добавляется дополнительная информация о найденных путях между этими вершинами в виде множества возможных промежуточных вершин. На первом этапе осуществляется построение множества матриц, хранящих в себе такую информацию о всех путях, удовлетворяющих заданным ограничениям. На втором этапе осуществляется построение искомых путей. **Основные результаты.** Предложенный алгоритм был реализован в виде программы на языке С++, и было проведено сравнение с другими наиболее эффективными алгоритмами поиска путей в графе с заданными контекстно-свободными ограничениями, а именно: с алгоритмом, использующим матричное умножение и позволяющим найти только один такой путь, и с алгоритмом, использующим произведение Кронекера и позволяющим найти все такие пути в графе. Результаты экспериментального исследования показали, что предложенный алгоритм существенно эффективнее (до 1000 раз быстрее) строит искомые пути, однако в некоторых случаях потребляет до 150 раз больший объём памяти, чем алгоритм, основанный на произведении Кронекера. **Практическая значимость.** Предложенный алгоритм может быть применён в задачах статического анализа кода, биоинформатике, сетевом анализе, а также в графовых базах данных, когда требуется найти все возможные зависимости в данных, представленных в виде помеченного графа.

**Ключевые слова**

поиск путей в графе, линейная алгебра, контекстно-свободные грамматики, матричное умножение, графовые базы данных, стандарт GraphBLAS

**Благодарности**

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 19-37-90101.

**Context-free path querying with all-path query semantics**

**Rustam Azimov, Semyon Grigorev**

**Abstract**

**Subject of Research.** The problem of context-free path querying with all-path query semantics is considered. This problem consists in finding all paths of the graph, the labels on the edges of which form words from the language generated by the input context-free grammar. There are two approaches to evaluate context-free path queries using linear algebra operations: matrix multiplication-based and the Kronecker product-based. But until now, there is no algorithm using the matrix multiplication capable of handling context-free path queries with the most complex all-path query semantics, in which the all paths that match the query must be provided. The paper proposes the algorithm for context-free path query evaluation using the matrix multiplication, which is capable of processing queries with the all-path query semantics. **Method.** In the adjacency matrix of the input graph for each pair of vertices, we store additional information about the paths found between these vertices in the form of a set of possible intermediate vertices. At the first stage, a set of matrices is constructed that store such information about all paths that satisfy the input query. At the second stage, all queried paths are restored from the constructed set of matrices. **Main Results.** The proposed algorithm was implemented in C++ and a comparison was made with other most efficient algorithms for evaluating context-free path queries, namely: with the matrix-based algorithm that allows us to find only one such path, and with the Kronecker product-based algorithm that allows us to find all such paths in the graph. The results of the experimental study showed that the proposed algorithm is significantly more efficient in restoring the queried paths, but in some cases it consumes a significantly larger amount of memory than the algorithm based on the Kronecker product. **Practical Relevance.** The proposed algorithm can be applied in problems of static code analysis, bioinformatics, network analysis, as well as in graph databases, when it is required to find all possible dependencies in the data presented in the form of a labeled graph.

**Keywords**

path querying, linear algebra, context-free grammars, matrix multiplication, graph databases, GraphBLAS

**Aknowledgements**

The reported study was funded by RFBR, project number 19-37-90101.

**Введение**

В современном мире становится всё больше данных, которые требуют обработки и анализа. При этом графы являются одной из самых распространённых и удобных структур данных, позволяя компактно представлять большие объемы информации и реализовывать эффективные алгоритмы для её анализа. В процессе анализа графа, например, может исследоваться существование определенных путей, достижимость некоторых вершин и т.д. Графы используются в статическом анализе кода [1,2], биоинформатике [3], в сетевом анализе [4]. Также в настоящее время активно развиваются графовые базы данных (Neo4j, RedisGraph и т.д.), используемые для хранения и реализации запросов к данным, представленным в виде графов.

Одной из важнейших задач анализа графов является поиск путей, обладающих некоторыми заданными свойствами. Если в результате анализа графа нет необходимости предъявлять такие пути, то решается задача достижимости, в которой исследуется лишь вопрос существования определённых путей. Но часто требуется предъявить пути, обладающие заданными свойствами, и тогда решаются задачи поиска путей, например, поиск одного пути или поиск всех путей.

Для описания свойств искомых путей в помеченном графе естественно задавать на них ограничения с помощью формальных грамматик над некоторым алфавитом. С помощью таких грамматик ограничивают множество слов, получаемых конкатенацией меток на рёбрах рассматриваемых путей. В настоящее время активно исследуются ограничения, представленные в виде контекстно-свободных (КС) грамматик. Этот подход позволяет описывать более широкий набор ограничений, чем, например, широко используемые на практике регулярные выражения.

Был предложен ряд алгоритмов поиска путей в графе с заданными контекстно-свободными ограничениями, которые основаны на различных методах синтаксического анализа: на основе (G)LL и (G)LR алгоритмов [5,6,7,8]; на основе CYK алгоритма [4]; с использованием парсер-комбинаторов [9]. Но недавнее исследование Йохема Куиджперса (Jochem Kuijpers) [10] показывает, что существующие решения не применимы для анализа реальных графов из-за значительных времени работы и потребления памяти.

Одним из распространённых способов улучшения производительности алгоритмов анализа графов является их переформулирование в терминах линейной алгебры. В такой формулировке, в основном, используются операции над матрицами и векторами. Для тех алгоритмов, которые позволяют найти такую формулировку, становится возможным применить для представления графов разреженные матрицы (т.е. матрицы, имеющие малое количество ненулевых элементов) и использовать параллельные вычисления, в частности, на основе GPU-технологий. Кроме того, такого рода алгоритмы зачастую просты в реализации, так как позволяют использовать существующие библиотеки линейной алгебры (GraphBLAS::SuiteSparse, cuSPARSE, cuBLAS, m4ri, Scipy и др.). Относительно недавно такая формулировка была найдена для задачи достижимости [11] и задачи поиска одного пути [12] в графе с заданными контекстно-свободными ограничениями. Алгоритмы решения данных задач основаны на матричных операциях и позволяют получить достаточно высокоэффективные реализации с помощью существующих библиотек линейной алгебры. Другой алгоритм поиска путей в графе с заданными контекстно-свободными ограничениями, сформулированный на языке линейной алгебры, основан на произведении Кронекера [13]. Этот алгоритм принципиально отличается от алгоритмов, основанных на обычных матричных операциях, тем, что не требует преобразования в нормальную форму входной КС-грамматики, в то время как прочие алгоритмы требуют преобразовывать входную КС-грамматику, например, в нормальную форму Хомского. Кроме того, алгоритм, основанный на произведении Кронекера, в процессе анализа строит более сложные структуры, тем самым позволяя находить все пути, удовлетворяющие контекстно-свободным ограничениям. Но до сих пор не существует алгоритма, использующего обычное матричное произведение, способного находить все пути, удовлетворяющие контекстно-свободным ограничениям. Преимуществом подхода, использующего обычное матричное произведение, по сравнению с подходом, предложенным в [13] и использующего произведение Кронекера, является построение матриц с более явной информацией об искомых путях в графе, что на практике позволит строить искомые пути значительно быстрее.

Таким образом, в данной работе ставится задача разработки алгоритма поиска всех путей с заданными контекстно-свободными ограничениями, использующего обычное матричное умножение.

**Постановка задачи**

*Терминал (терминальный символ)* — объект, непосредственно присутствующий в словах формального языка, соответствующего формальной грамматике, и имеющий конкретное, неизменяемое значение. В данной статье будем использовать Σ для обозначения конечного алфавита, который состоит из терминальных символов формальной грамматики.

*Нетерминал (нетерминальный символ)* — объект, обозначающий какую-либо сущность языка (например: формула, арифметическое выражение, команда) и не имеющий конкретного символьного значения. В данной статье будем использовать *N* для обозначения конечного множества нетерминальных символов формальной грамматики.

*Контекстно-свободная грамматика в слабой нормальной форме Хомского* — это четвёрка *G* = (*N,* Σ*, P, S*), где *N* — конечное множество нетерминальных символов, Σ — конечное множество терминальных символов, — стартовый нетерминал и *P* — конечное множество правил следующего вида:

* , для и операции конкатенации;
* , для *,* , где — пустая строка.

Мы рассматриваем только КС-грамматики в слабой нормальной форме Хомского, так как для каждой КС-грамматики можно построить эквивалентную ей грамматику в данной форме [14]. Операцию конкатенации в правой части правил иногда будем опускать и писать .

Мы будем писать, чтобы указать, что строка может быть получена из нетерминала *A* некоторой последовательностью применений правил КС-грамматики. А я*зыком*, задаваемым КС-грамматикой *G* = (*N,* Σ*, P, S)*,будем называть

.

Пусть Σ — конечное множество терминальных символов. *Помеченным графом* будем называть ориентированный граф *D* = (*V, E,* Σ), где *V* является множеством вершин, а — множеством ребер с метками из алфавита Σ. Для пути *π* в графе *D* мы будем использовать обозначение *l*(π), чтобы указать на слово, полученное конкатенацией меток на рёбрах данного пути. Кроме того, запись *iπj* будет обозначать, что в рассматриваемом графе существует путь из вершины в вершину .

Для заданного графа *D* = (*V, E*) и КС-грамматики *G* = (*N,* Σ*, P, S*) определяется *контекстно-свободное отношение* , являющееся множеством пар вершин, между которыми существует путь, образующий строку, выводимую из стартового нетерминала *S*. Такое отношение определяется следующим образом:

.

Используя данные определения мы можем сформулировать задачу поиска всех путей в графе с заданными контекстно-свободными ограничениями.

**Задача.** Проблема поиска всех путей в графе, удовлетворяющих контекстно-свободным ограничениям, для заданного помеченного графа D и КС-грамматики G заключается в вычислении контекстно-свободного отношения и вычислении для каждой пары вершин множества всех путей между ними, таких что .

Множество таких путей может быть бесконечным в случае наличия циклов во входном графе. Поэтому обычно для таких множеств используется некоторое конечное представление.

**Матричный алгоритм поиска всех путей в графе с заданными контекстно-свободными ограничениями**

Предлагаемый алгоритм основан на матричном алгоритме [11], решающем задачу достижимости с заданными контекстно-свободными ограничениями. Этот алгоритм сводит данную задачу к вычислению операций над булевыми матрицами и, как следствие, позволяет использовать высокопроизводительные библиотеки линейной алгебры и современное параллельное оборудование для решения данной задачи анализа графов.

Для каждой пары вершин мы будем добавлять в ячейки матриц дополнительную информацию о найденных путях между этими вершинами в виде множества возможных промежуточных вершин. Добавленные промежуточные вершины описывают найденные пути как конкатенации двух меньших путей. Используя эту информацию, мы имеем возможность восстанавливать все пути, которые образуют слова, выводимые из любого нетерминала заданной КС-грамматики.

Мы определим матричное умножение , где 𝑎 и 𝑏 — квадратные матрицы размера , которые имеют множества вершин графа (промежуточных) в качестве элементов, как , где

.

Также мы используем операцию + поэлементного сложения матриц 𝑎 и 𝑏 одинакового размера: 𝑎 + 𝑏 = 𝑐, где .

Используя введённые матричные операции, мы представляем матричный алгоритм поиска всех путей в графе с заданными контекстно-свободными ограничениями. Псевдокод данного алгоритма приведен на рис. 1.

Для заданных КС-грамматики 𝐺 = (𝑁, Σ, 𝑃, 𝑆) и помеченного графа 𝐷 = (𝑉, 𝐸, Σ) результатом работы представленного алгоритма является набор матриц 𝑇 (так называемый индекс), который хранит информацию обо всех путях в графе 𝐷, образующих слово, выводимое из некоторого нетерминала КС-грамматики 𝐺. В строках 4 и 5 представленного алгоритма мы добавляем специальное значение 𝑛 к ячейкам , чтобы указать, что это путь с одним ребром или пустой путь .

Построенный индекс позволяет вычислить контекстно-свободное отношение (включая в него все пары вершин (i,j), для которых ), а также построить любой из найденных путей и тем самым решает задачу поиска всех путей в графе с заданными контекстно-свободными ограничениями. Однако, множество таких путей может быть бесконечным в случае наличия циклов во входном графе. С практической точки зрения, при построении найденных путей необходимо использовать «ленивое» вычисление или каким-то образом ограничить результирующее множество путей. Например, можно потребовать построить некоторое фиксированное количество путей или ограничить сверху длину путей.

**Function**AllPathCFPQ(

D = (V, E,), // Входной помеченный граф

G = (N, , P, S)) // Входная КС-грамматика

1. for all
2. for all
3. while do
4. for all do
6. return

*Рисунок 1. Матричный алгоритм поиска всех путей в графе с заданными контекстно-свободными ограничениями*

*Figure 1. A matrix-based context-free path querying algorithm with all-path query semantics*

Рисунок 2. Алгоритм построения всех путей, удовлетворяющих заданным контекстно-свободным ограничениям

**Function**ExtractAllPaths(i, j, A, **,** G = (N, , P, S))

1. if then
2. return // Таких путей не существует
3. for all do
4. if then // Добавляем путь из одного ребра или пустой путь
5. for all do
6. if then
8. if then
10. else // Добавляем к результату конкатенацию путей из *i* в *k* и путей из *k* в *j*
11. for all do

14. if then


18. return

*Figure 2. Paths extraction algorithm for the context-free path querying*

Кроме алгоритма построения индекса, мы также предлагаем алгоритм построения найденных путей, псевдокод которого приведен на рис. 2. Если для заданных 𝑖, 𝑗, 𝐴, соответствующий элемент матрицы равен , то алгоритм возвращает пустое множество, так как путей искомого вида не существует. Данный алгоритм возвращает множество с пустым путём , только если 𝑖 = 𝑗 и 𝐴 → 𝜀 ∈ 𝑃. В строке 19 мы используем операцию , которая естественным образом обобщает операцию конкатенации путей, строя все возможные конкатенации пар путей из двух множеств. Предполагается, что множества путей вычисляются «лениво», чтобы обеспечить завершаемость алгоритма в случае бесконечного количества путей.

**Эксперименты**

Целью данного экспериментального исследования является изучение применимости предложенного матричного алгоритма и сравнение с другими наиболее эффективными алгоритмами поиска путей в графе с заданными контекстно-свободными ограничениями, использующими операции линейной алгебры. Мы сравним лучшие реализации алгоритмов поиска путей, удовлетворяющих контекстно-свободным ограничениям, которые основаны на операциях линейной алгебры, а именно:

* 𝑀𝑡𝑥𝑆𝑖𝑛𝑔𝑙𝑒 — реализация из [12] матричного алгоритма поиска одного пути в графе с заданными контекстно-свободными ограничениями;
* 𝑇𝑛𝑠 — реализация из [13] алгоритма поиска всех путей в графе с заданными контекстно-свободными ограничениями, основанного на произведении Кронекера;
* 𝑀𝑡𝑥𝐴𝑙𝑙 — реализация предложенного матричного алгоритма поиска всех путей в графе с заданными контекстно-свободными ограничениями, которая для матричных операций использует реализацию стандарта GraphBLAS.

Все реализации используют ЦПУ и матрицы в разреженном формате. Сначала мы измерили время выполнения и требуемую память для создания индекса. Затем мы сравнили практическую применимость построения путей для обеих реализаций 𝑀𝑡𝑥𝐴𝑙𝑙 и 𝑇𝑛s алгоритмов поиска всех путей в графе с заданными контекстно-свободными ограничениями. Исходный код доступен на GitHub[[1]](#footnote-1). Для экспериментов мы использовали ПК с установленной ОС Ubuntu 18.04, процессором Intel Core i7-6700, 3,4 ГГц и оперативной памятью DDR4 64 ГБ.

Таблица 1. Время построения индекса в секундах и использованная память в мегабайтах для грамматики g1

Table 1. Index construction time in seconds and used memory in megabytes for the grammar g1

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Граф | |V| | |E| | КС-грамматика g1 | | | | | |
| MtxAll | | Tns | | MtxSingle | |
| Время | Память | Время | Память | Время | Память |
| pathways | 6 238 | 18 598 | 0,04 | 91 | 0,02 | 123 | 0,01 | 671 |
| go-hierarchy | 45 007 | 980 218 | 22,12 | 38 797 | 0,17 | 265 | 1,41 | 660 |
| enzyme | 48 815 | 109 695 | 0,4 | 307 | 0,04 | 137 | 0,01 | 216 |
| eclass\_514en | 239 111 | 523 727 | 25,02 | 14 416 | 0,24 | 205 | 0,23 | 216 |
| go | 272 770 | 534 311 | 11,8 | 8 290 | 1,58 | 282 | 1,45 | 215 |
| geospecies | 450 609 | 2 311 461 | 4,45 | 2 691 | 0,08 | 218 | 0,06 | 2 250 |
| taxonomy | 5 728 398 | 14 922 125 | error | error | 4,42 | 2 018 | 2,73 | 1 962 |

Таблица 2. Время построения индекса в секундах и использованная память в мегабайтах для грамматики g2

*Table 2. Index construction time in seconds and used memory in megabytes for the grammar g2*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Граф | |V| | |E| | КС-грамматика g2 | | | | | |
| MtxAll | | Tns | | MtxSingle | |
| Время | Память | Время | Память | Время | Память |
| pathways | 6 238 | 18 598 | 0,01 | 49 | 0,01 | 122 | 0,01 | 671 |
| go-hierarchy | 45 007 | 980 218 | 15,66 | 28 447 | 0,24 | 252 | 0,84 | 671 |
| enzyme | 48 815 | 109 695 | 0,02 | 61 | 0,02 | 132 | 0,01 | 217 |
| eclass\_514en | 239 111 | 523 727 | 0,22 | 126 | 0,27 | 193 | 0,16 | 216 |
| go | 272 770 | 534 311 | 1,13 | 990 | 1,27 | 243 | 0,93 | 217 |
| geospecies | 450 609 | 2 311 461 | 0,34 | 156 | 0,01 | 196 | 0,01 | 2 251 |
| taxonomy | 5 728 398 | 14 922 125 | 19,13 | 27 232 | 3,56 | 1 776 | 1,15 | 2 250 |

**Данные.** Мы используем графы и соответствующие КС-грамматики из набора данных, представленного в [12], который содержит графы для реальных данных в формате RDF (*pathways, go-hierarchy, enzyme, eclass\_514en, go, geospecies, taxonomy*) и грамматики 𝑔1, 𝑔2, которые используются для поиска всех вершин в графе, находящихся на одном уровне иерархии [15]. Описанное свойство — один из важных примеров свойств, которые могут быть выражены с помощью КС-грамматик, но не с помощью регулярных.

**Результаты.** Результаты создания индекса для всех трех реализаций представлены в таблицах 1 и 2. Мы видим, что матричный алгоритм поиска одного пути позволяет строить индекс наиболее быстро, особенно при работе с большими графами. Но 𝑀𝑡𝑥𝑆𝑖𝑛𝑔𝑙𝑒 не может построить все найденные при анализе графа пути, так как использует более простой индекс, чтобы восстановить только один путь для каждой пары вершин. Однако реализация 𝑇𝑛𝑠 алгоритма, основанного на произведении Кронекера, использует более сложный, но компактный индекс, что позволяет не только построить все найденные пути, но и потреблять меньше памяти, чем другие матричные алгоритмы.

Реализация 𝑀𝑡𝑥𝐴𝑙𝑙 предлагаемого матричного алгоритма поиска всех путей в графе с заданными контекстно-свободными ограничениями сравнима по времени выполнения с реализацией 𝑇𝑛𝑠 на маленьких графах, но значительно медленнее на некоторых больших графах со сложной структурой. Кроме того, на некоторых графах 𝑀𝑡𝑥𝐴𝑙𝑙 потребляет значительно больше памяти, чем 𝑇𝑛𝑠. Причина такого поведения заключается в том, что предлагаемый матричный алгоритм пытается сохранить информацию обо всех найденных путях в более явном виде. На самом большом графе 𝑡𝑎𝑥𝑜𝑛𝑜𝑚𝑦 и КС-грамматике 𝑔1 произошла ошибка нехватки памяти для реализации 𝑀𝑡𝑥𝐴𝑙𝑙, что отмечено в таблице с помощью «error».

После построения индекса мы сравнили время построения путей для двух реализаций алгоритмов поиска всех путей в графе с заданными контекстно-свободными ограничениями. Результаты построения путей с длиной, не превосходящей 10, для графов 𝑔𝑜 и 𝑒𝑐𝑙𝑎𝑠𝑠\_514𝑒𝑛 представлены на рис. 3 и 4 (используются стандартные диаграммы размаха, указаны медианы и выбросы опускаются). После построения путей для каждой пары вершин, мы группируем время построения по количеству возвращаемых путей. Видно, что время извлечения путей в реализации 𝑀𝑡𝑥𝐴𝑙𝑙 предложенного матричного алгоритма до 1000 раз меньше, чем в реализации 𝑇𝑛𝑠. В предложенном матричном алгоритме, мы строим индекс с более явной информацией о найденных путях, что позволяет значительно быстрее их восстанавливать.

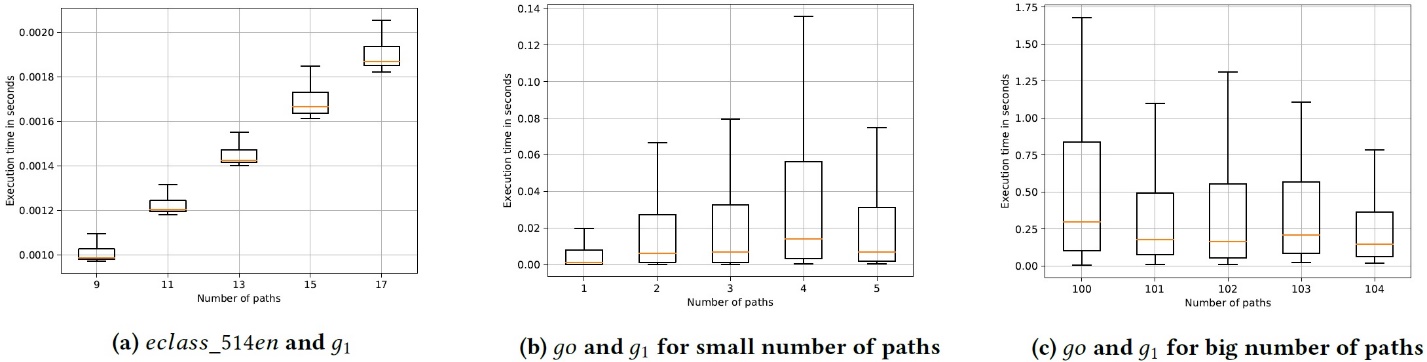
****

Рисунок 3. Время построения путей в секундах для реализации Tns алгоритма, основанного на произведении Кронекера

*Figure 3. Paths construction time in seconds for the implementation Tns of the Kronecker product-based algorithm*

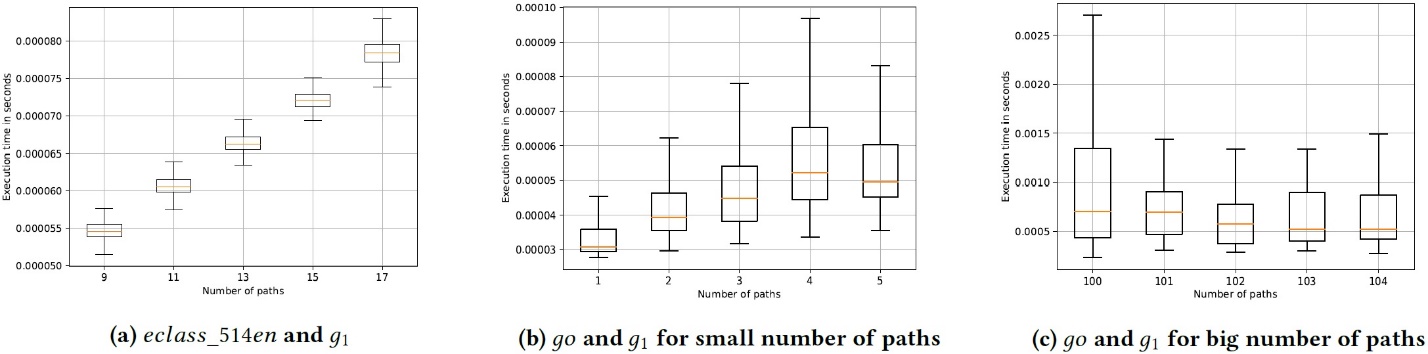
****

Рисунок 4. Время построения путей в секундах для реализации MtxAll предложенного матричного алгоритма

*Figure 4. Paths construction time in seconds for the implementation MtxAll of the proposed matrix-based algorithm*

**Выводы.** По результатам экспериментального исследования мы можем сделать следующие выводы.

* Для задачи поиска одного пути, удовлетворяющего контекстно-свободным ограничениям, матричный алгоритм [12] является наиболее производительным, а алгоритм [13], основанный на произведении Кронекера, потребляет наименьшее количество памяти.
* Для задачи поиска всех путей с заданными контекстно-свободными ограничениями, в случае, когда необходимо часто пересчитывать индекс для изменяющегося графа или КС-грамматики, лучшим выбором является алгоритм [13], основанный на произведении Кронекера, с более быстрым построением индекса и меньшим потреблением памяти. Если же необходимо строить пути много раз для однажды построенного индекса или если изменения в индексе могут быть эффективно подсчитаны динамически, то предлагаемый матричный алгоритм поиска всех путей с заданными контекстно-свободными ограничениями предпочтительнее.

**Заключение**

В данной статье предлагается алгоритм поиска всех путей с заданными контекстно-свободными ограничениями, основанный на матричном алгоритме для задачи достижимости [11].

Предложенный алгоритм был реализован с использованием реализации стандарта GraphBLAS. Кроме того, было проведено сравнение полученной реализации с реализациями наиболее эффективных алгоритмов поиска путей в графе с заданными контекстно-свободными ограничениями, основанных на операциях линейной алгебры. Для задачи поиска одного пути, удовлетворяющего контекстно-свободным ограничениям, матричный алгоритм [12] является наиболее производительным, а алгоритм [13], основанный на произведении Кронекера, потребляет наименьшее количество памяти. В то время как для задачи поиска всех путей, удовлетворяющих контекстно-свободным ограничениям, предложенный алгоритм является наиболее производительным в случае, когда необходимо извлекать пути много раз для однажды построенного индекса или если изменения в индексе могут быть эффективно подсчитаны динамически.

В дальнейшем, предложенному алгоритму необходимы улучшения в потреблении памяти при построении индекса, а для алгоритма, основанного на произведении Кронекера, необходимо ускорение построения путей.

**Литература**

1. Rehof J., Fähndrich M. 2001. Type-Base Flow Analysis: From Polymorphic Subtyping to CFL-Reachability // SIGPLAN V. 36, N 3, 2001, P. 54–66. doi: 10.1145/373243.360208
2. 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 (POPL ’08). ACM, New York, NY, USA, 2008. P. 197–208. doi: 10.1145/1328438.1328464
3. Sevon P., Eronen L. Subgraph Queries by Context-free Grammars // Journal of Integrative Bioinformatics, V. 5, N 2, 2008. P. 157–172. doi: 10.1515/jib-2008-100
4. Zhang X. et al. Context-Free Path Queries on RDF Graphs // In The Semantic Web – ISWC, 2016, Springer International Publishing, Cham, P. 632–648.
5. Medeiros C. et al. Efficient Evaluation of Context-free Path Queries for Graph Databases // In Proceedings of the SAC’18. ACM, New York, NY, USA, 2018. P. 1230–1237. doi: 10.1145/3167132.3167265
6. Santos F. et al. A Bottom-Up Algorithm for Answering Context-Free Path Queries in Graph Databases // In Web Engineering, Springer International Publishing, Cham, 2018. P. 225–233.
7. Grigorev S., Ragozina A. Context-free Path Querying with Structural Representation of Result // In Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR’17), 2017, ACM, New York, NY, USA, Article 10, 7 pages. doi: 10.1145/3166094.3166104
8. Verbitskaia E., Grigorev S., Avdyukhin D. Relaxed Parsing of Regular Approximations of String-Embedded Languages // In Perspectives of System Informatics, Springer International Publishing, Cham, 2016. P. 291–302.
9. Verbitskaia E. et al. Parser Combinators for Context-free Path Querying // In Proceedings of the 9th ACM SIGPLAN International Symposium on Scala (Scala 2018). ACM, New York, NY, USA, 2018. P. 13–23. doi: 10.1145/3241653.3241655
10. Kuijpers J. et al. An Experimental Study of Context-Free Path Query Evaluation Methods // In Proceedings of the SSDBM’19. ACM, New York, NY, USA, 2019. P. 121–132. doi: 10.1145/3335783.3335791
11. Azimov R., Grigorev S. Context-free Path Querying by Matrix Multiplication // In Proceedings of the GRADES-NDA’18. ACM, New York, NY, USA, 2018. Article 5, 10 pages. doi: 10.1145/3210259.3210264
12. Terekhov A. et al. Context-Free Path Querying with Single-Path Semantics by Matrix Multiplication // In Proceedings of the GRADES-NDA’20. ACM, New York, NY, USA, 2020. Article 5, 12 pages. doi: 10.1145/3398682.3399163
13. Orachev E. et al. Context-Free Path Querying by Kronecker Product // In European Conference on Advances in Databases and Information Systems. Springer, 2020. P. 49–59.
14. Chomsky N. On certain formal properties of grammars. Information and control, Vol. 2, N 2, 1959. P. 137–167.
15. Abiteboul S. et al. Foundations of Databases: The Logical Level (1st ed.). Addison-Wesley Longman Publishing Co., Inc., USA, 1995.

1. Исходный код предложенного алгоритма. [Электронный ресурс]. – Режим доступа: <https://github.com/JetBrains-Research/CFPQ_PyAlgo>, свободный. Яз. англ. (дата обращения 01.04.2021). [↑](#footnote-ref-1)