Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование
информационных систем
Системное программирование

Илья Владимирович Эпельбаум

Поиск путей в графовых базах данных через тензорное произведение

Курсовая работа

Научный руководитель:
к. ф.-м. н., доцент С. В. Григорьев

Санкт-Петербург
2020

Оглавление

Введение

Сегодня людям необходимо иметь доступ к большому объему информации. Для этого были придуманы базы данных. Разработано множество моделей построения баз данных. Например, часто используемой моделью является реляционная, где данные представлены в виде отношений. Однако данный тип не всегда представляется удобным в использовании, так как не обладает гибкостью во внесении изменений. В свою очередь, графовая модель данных обладает этим достоинством, что говорит о её конкурентоспособности. Впервые применение такой модели в базе данных предложена Михалисом Яннакакисом [9]. В ней информация представляется в виде вершин (сущности) и дуг (связи между сущностями). Примером графовой СУБД является Neo4j[1].

Также возникает потребность в обработке информации. Например, осуществлять поиск путей в графовой базе данных. В основе графовой базы данных лежит граф, поэтому далее в рамках данной работы естественно отождествлять эти понятия и свести задачу поиска путей в такой базе данных к поиску путей в графе.

Имеется множество подходов к поиску путей в графе с ограничениями. Один из таких способов предоставляет теория формальных языков. А именно, пусть имеется помеченный граф, где метки взяты из некоторого алфавита. При таких допущениях наличие пути сразу же свидетельствует о наличии слова, полученного конкатенацией меток на дугах. Соответственно множество путей является множеством слов, то есть по определению оказывается языком. Теперь, чтобы ввести ограничения на эти пути, предлагается задать грамматику, которая будет описывать необходимый язык, то есть пути с необходимыми ограничениями. Таким образом алгоритм поиска путей должен выдавать только те пути, объединение меток которых является словом языка, порождаемого грамматикой.

Данный подход применяется в следующих областях: статический анализ кода [7], биоинформатика [8], анализ RDF файлов [3].

[1]Официальный сайт Neo4j: https://neo4j.com. Дата посещения: 16.05.2020.

Существует множество алгоритмов, решающих данную задачу. Однако авторы статьи [10], взяв существующие алгоритмы, показали, что каждый из них может быть успешно применен в промышленных продуктах лишь при определенных условиях на входные данные. Но при этом в общем виде требуют больших вычислительных ресурсов, поэтому успешное использование их в настоящее время не представляется возможным. Вместе с этим, в недавнем исследовании, проведенном авторами статьи [4], было продемонстрировано, что линейная алгебра является одним из путей к высокопроизводительным алгоритмам в данной области. Вследствие этого лабораторией языковых инструментов JetBrains ведутся разработки новых более эффективных методов, использующих этот подход. Результатом такой деятельности стал алгоритм, основанный на тензорном произведении. Соответственно, возникает потребность в сравнении этого результата с конкурирующими алгоритмами.

1. Постановка задачи

Целью данной работы является сравнение алгоритма, основанного на тензорном произведении, и алгоритма, основанного на матричном умножении. Для достижения цели были поставлены приведённые ниже задачи.

1. Реализовать алгоритм поиска путей через тензорное произведение.

2. Внедрить реализацию в СУБД RedisGraph.

3. Произвести экспериментальное исследование производительности реализации. Для демонстрации затраченного времени для работы реализованного алгоритма над исходными данными использовать набор данных [2].

[2]Репозиторий набора данных: https://github.com/JetBrains-Research/CFPQ_Data . Дата посещения: 16.05.2020.

2. Обзор

Приведем определения понятий, которые используются в работе.

2.1. Базовые определения

Следующее определение описывает структуру представления грамматики, которую использует алгоритм, основанный на тензоронм произведении.

Определение 2.1. Рекурсивный автомат [1] над конечным алфавитом Σ есть набор $(M, m, \{C_i\}_{i \in M})$, где

- M конечное множество меток,
- $m \in M$ начальная метка,
- $\{C_i\}_{i \in M}$ множество *конечных автоматов*, где $C_i = (\Sigma \cup M, Q_i, q_i^0, F_i, \delta_i)$:
  - $\Sigma \cup M$ множество символов, $\Sigma \cap M = \emptyset$,
  - Q_i конечное множество состояний, где $Q_i \cap Q_j = \emptyset, \forall i \neq j$,
  - q_i^0 начальное состояние C_i,
  - F_i множество финальных состояний C_i, где $F_i \subseteq Q_i$,
  - δ_i функция переходов C_i, где $\delta_i : Q_i \times (\Sigma \cup M) \to Q_i$.

Определение 2.2. Пусть G = $\langle$Σ,N,P,S$\rangle$ констекстно-свободная грамматика, тогда G находится в **ослабленной нормальной форме Хомского** (ОНФХ), если содержит только правила вида:

- $A \to BC$, где A, B, $C \in N$
- $A \to a$, где $A \in N$, $a \in \Sigma$
- $A \to \varepsilon$, где $A \in N$

Определение ОНФХ отличается от нормальной формы Хомского наличием правил вида $A \to \varepsilon$, где A — любой нетерминал, то есть A не обязательно является стартовым, а также допущением использовать стартовый нетерминал в правых частях правил.

2.2. Алгоритм, основанный на матричном умножении

В настоящий момент существует два алгоритма для КС запросов, использующих линейную алгебру. Первый предложен аспирантом кафедры системного программирования, Рустамом Азимовым [2]. Имеет сложность $O(n^2|N|^2|P|(MM(n)))$, где $|P|$ — количество правил грамматики, $|N|$ — мощность множества нетерминалов, MM(n) — сложность умножения квадратных матриц размера n. Основан на обычном произведении матриц, что позволяет достичь хорошей производительности. Однако он требует, чтобы грамматика находилась в ослабленной нормальной форме Хомского, что приводит к её разрастанию и, как следствие, отрицательно сказывается на производительности и потребляемой памяти. В борьбе с этим, был разработан другой алгоритм. Он основан на тензорном произведении.

2.3. Алгоритм, основанный на тензорном произведении

На листинге 1 приведен алгоритм, основанный на тензорном произведении. На вход алгоритм принимает ориентированный помеченный граф $\mathcal{G} = \langle V, E, L \rangle$ и грамматику G.

Сначала, используя подход, описанный в статье [1], строится рекурсивный автомат для G. Далее пусть M_1 — матрица смежности рекурсивного автомата, а M_2 — матрица смежности $\mathcal{G}$. На первом шаге вычисляется тензорное произведение M_1 и M_2, тем самым пересекая два автомата (граф и рекурсивный автомат). После чего результат транзитивно замыкается. Следующим шагом, обходя матрицу транзитивного замыкания, проверяется наличие ребра, и принадлежность начальной и конечной вершин стартовому и конечному состоянию рекурсивного автомата, для этого на листинге 2 приведены вспомогательные функции. При выполнении условия алгоритм добавляет нетерминал/нетерминалы, соответствующие стартовому и конечному состоянию автомата, в ячейку

M_2. Данные шаги алгоритма повторяются пока матрица M_2 изменяется. За каждую итерацию i получаются пути, выводимые из грамматики за i шагов. Результатом работы алгоритма является матрица M_2

Listing 1 Алгоритм

```
1: function contextFreePathQuerying(G, 𝒢)
2:     R ← Рекурсивный автомат для G
3:     M_1 ← Матрица смежности для R
4:     M_2 ← Матрица смежности для 𝒢
5:     for s ∈ 0..dim(M_1) − 1 do
6:         for i ∈ 0..dim(M_2) − 1 do
7:             M_2[i, i] ← M_2[i, i] ∪ getNonterminals(R, s, s)
8:     while Матрица M_2 изменяется do
9:         M_3 ← M_1 ⊗ M_2          ▷ Вычисление тензорного произведения
10:        C_3 ← transitiveClosure(M_3)
11:        n ← dim(M_3)
12:        for i ∈ 0..n − 1 do
13:            for j ∈ 0..n − 1 do
14:                if C_3[i, j] then
15:                    s, f ← getStates(C_3, i, j)
16:                    if getNonterminals(R, s, f) ≠ ∅ then
17:                        x, y ← getCoordinates(C_3, i, j)
18:                        M_2[x, y] ← M_2[x, y] ∪ getNonterminals(R, s, f)
19:    return M_2
```

Listing 2 Вспомогательные функции

```
1: function getStates(M_1, i, j)
2:     r ← dim(M_1)
3:     return ⌊i/r⌋ , ⌊j/r⌋
4: function getCoordinates(M_2, i, j)
5:     n ← dim(M_2)
6:     return i mod n, j mod n
```

Данный алгоритм использует рекурсивный автомат, что полностью избавляет от необходимости преобразовывать грамматику, а также полностью основан на линейно алгебре. Однако в процессе работы, а именно в результате тензорного произведения, возникают матрицы больших размеров. Например, если тензорно перемножить одну матрицу размером $m \times p$ и вторую матрицу $n \times q$, то в результате получается матрица $mn \times pq$, что может отрицательно сказаться на производительности.

2.4. Используемые технологии

Рассмотрим технологии, которые используются для реализации и внедрения алгоритма, основанного на тензорном произведении.

2.4.1. Библиотека SuiteSparse

Для реализации был выбрана библиотека SuiteSparse[3]. SuiteSparse является реализацией API GraphBlas. Обладает широким спектром возможностей [5] для реализаций алгоритмов, которые подобны изложенным выше, так как создавался для произведения операций над графами средствами линейной алгебры. Например, предоставляет возможность определять новые полукольца с операциями, которые также можно определять. В частности имеет встроенное тензорное произведение над булевым полукольцом. Также позволяет хранить матрицы, учитывая свойство разреженности, что необходимо для эфективной реализации алгоритма, основанного на тензорном произведении.

2.4.2. Графовая база данных RedisGraph

Для внедрения реализации была выбрана СУБД RedisGraph[4]. СУБД с открытым исходным кодом, что предоставляет возможность полноценно внедрить реализацию алгоритма. RedisGraph использует стандарт GraphBlas для выполнения операций над графами, что упрощает

[3]Сайт проекта: http://faculty.cse.tamu.edu/davis/suitesparse.html. Дата посещения: 16.05.2020.

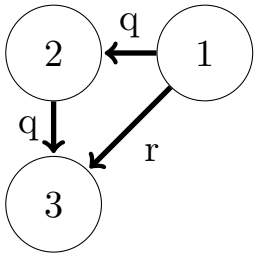
[4]Репозиторий проекта RedisGraph: https://github.com/RedisGraph/RedisGraph. Дата посещения: 16.05.2020.

интеграцию алгоритма. Одним из главных факторов выбора является быстродействие работы СУБД, а как показано в статье [6], RedisGraph является конкурентноспособной СУБД по данному направлению. Изложенные аргументы подтверждают правильность выбора платформы для выполнения поставленной цели.

3. Реализация и интеграция

3.1. Подход к реализации

Для реализации поставленной задачи был применен следующий подход. Вместо обычной матрицы смежности графа для каждой метки заводится булева матрица смежности графа, то есть в матрице на месте $(i; j)$ стоит 1, если в графе из вершины i в вершину j есть дуга с соответствующей меткой. Продемонстрируем на примере.



$$\Rightarrow \begin{bmatrix} 0 & q & r \\ 0 & 0 & q \\ 0 & 0 & 0 \end{bmatrix} \Rightarrow r : \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, q : \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

В таком случае тензорное произведение вычисляется в булевом полукольце. А именно, вычисляется тензорное произведение булевых матриц смежности, которые соответствуют одним меткам, и результаты суммируются.

3.2. Архитектура

В ходе работы над реализацией и интеграции алгоритма в RedisGraph были реализованы модули, указанные на рис. 1 синим цветом. Для их разработки использовался язык программирования C, так как исходный код RedisGraph написан на данном языке.

Модуль "Автомат" представляет структуру данных, использующую матрицы из библиотеки SuiteSparse, для хранения рекурсивного автомата. Матрица смежности представлена способом, описанным в разделе 3.1. Данная структура содержит только один метод для загрузки автомата из файла.

Модуль "Алгоритм" является реализацией алгоритма, основанного на тензорном произведении. Реализация принимает на вход граф из RedisGraph и экземпляр структуры данных рекурсивного автомата.

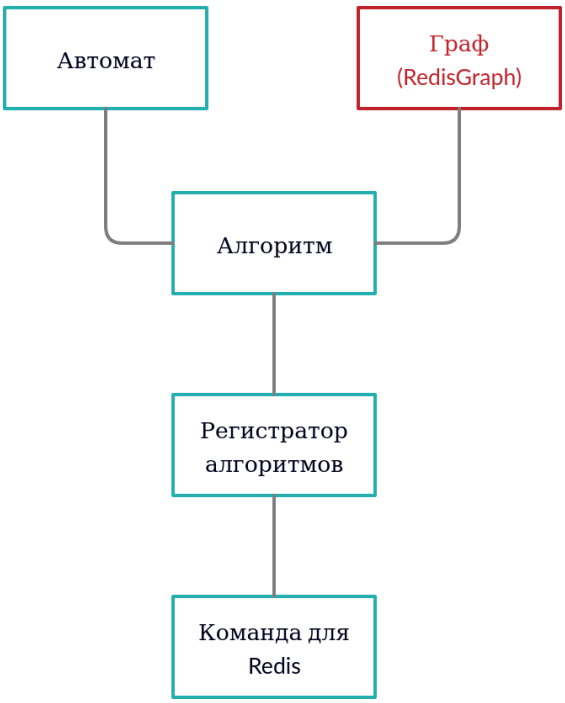


Рис. 1: Модули

Структура хранения графа в RedisGraph соответствует подходу, описанному в разделе 3.1. Поэтому для вычисления тензорного произведения матриц над булевым полукольцом используется встроенная функция библиотеки SuiteSparse.

Модуль ”Регистратор алгоритмов” представляет структуру данных для хранения реализаций алгоритмов, использующих граф и рекурсивный автомат. Добавление алгоритма для последующего запуска в redis-server является обязательным.

Модуль ”Команда для Redis” является реализацией команды для redis-server. Команда принимает 3 обязательных аргумента: название алгоритма, содержащееся в регистраторе алгоритмов, название графа из загруженных в RedisGraph и путь до автомата. При запуске команды происходит создание структуры данных рекурсивного автомата и его загрузка, после чего происходит запуск алгоритма.

4. Эксперименты

Для постановки экспериментов над реализацией был использован ПК с операционной системой Ubuntu 18.04 и конфигурацией: Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz CPU, DDR4 32 Gb RAM. Сравнения производились с реализацией алгоритма, основанного на матричном умножении, использующая тот же набор технологий и ПК для проведения экспериментов.

В качестве набора данных [5] были взяты графы следующих классов:

- Worstcase — теоретически наихудшая ситуация.
- FullGraph — наихудшая ситуация для разреженных матриц.
- RDF — реальные данные.

В ходе проведения экспериментов бралась выборка из 10 результатов над каждым классом. В итоге были получены следующие результаты.

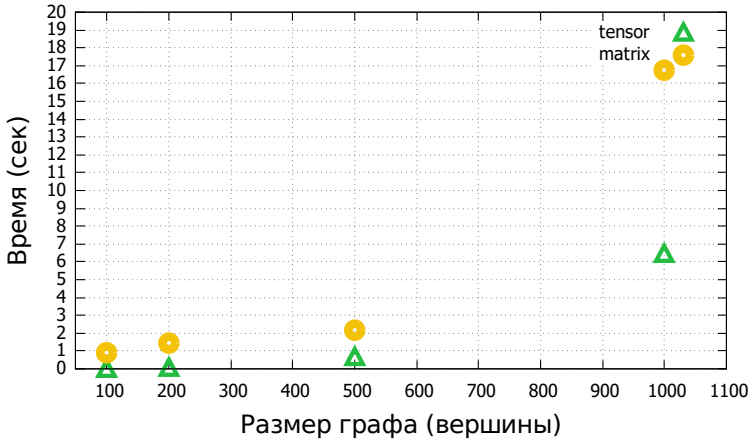


Рис. 2: FullGraph

На рис. 2 и рис. 3 представлены графики результатов замеров на классах FullGraph и Worstcase соответственно. Показатели алгоритма, основанного на тензорном произведении, демонстрируют превосходство

[5] Репозиторий набора данных: https://github.com/JetBrains-Research/CFPQ_Data . Дата посещения: 16.05.2020.

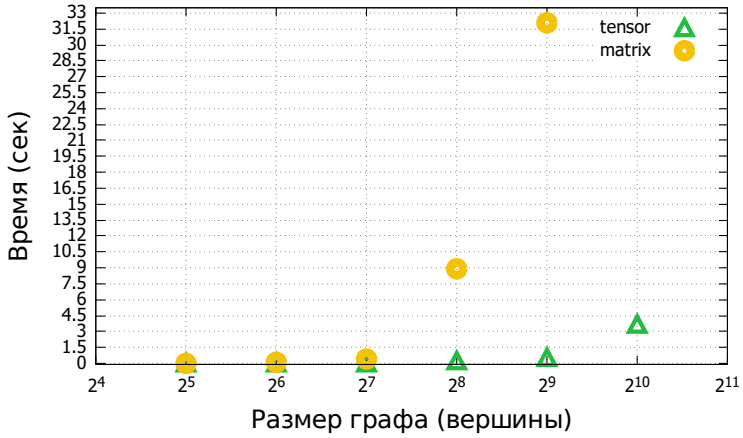


Рис. 3: Worstcase

над результатами алгоритма, предложенного Рустамом Азимовым, более чем в 50 раз и в 2 раза соответственно. Показатель для доверительного интервала — $\pm\Delta = \pm 0,06$.

Граф	V	E	Tensor	Matrix
pathways	6238	37196	0.008	0.009
go	272770	1068622	1.744	0.604
atom-primitive	291	685	0.011	0.016
eclass_514en	239111	1047454	0.329	0.067
foaf	256	815	0.001	0.002
funding	778	1480	0.005	0.006
go-hierarchy	45007	1960436	0.179	0.091
bio-mesure	341	711	0.010	0.016
pizza	671	2604	0.026	0.030
enzyme	48815	219390	0.157	0.018

Таблица 1: RDF

В таблице 1 собраны результаты на реальных данных. Положительные показатели наблюдаются только на графах, отмеченных зеленым цветом, а именно: pathways, atom-primitive, foaf, pizza, funding, bio-mesure. Показатель для доверительного интервала — $\pm\Delta = \pm 0,06$.

Таким образом из приведенных результатов экспериментов можно сделать следующие выводы.

- Положительные результаты для RDF наблюдаются только на маленьких графах. Однако стоит заметить, что сравнения производились с реализацией, которая улучшалась в течение нескольких лет, а реализация алгоритма, представленная в работе, является единственной.

- Показатели на синтетических данных говорят о некотором потенциале нового алгоритма.

Приведенные выводы формируют некоторую картину производительности алгоритма в сравнении с алгоритмом, основанным на матричном умножении. Вследствие её можно сделать вывод о необходимости улучшения реализации для получения лучших результатов.

Заключение

В рамках курсовой работы были выполнены следующие задачи:

1. Реализован[6] алгоритм, основанный на тензорном произведении
2. Реализация интегрирована в СУБД RedisGraph
3. Произведены экспериментальные исследования производительности реализации на наборах данных:
   - WorstCase
   - FullGraph
   - RDF

Также результаты работы изложены в статье "Context-Free Path Querying by Kronecker Product". Статья принята на конференцию ADBIS 2020. В качестве дальнейших исследований предлагается следующее.

- Произвести улучшение реализации алгоритма и сравнить новые результаты с существующими. Улучшения, например, могут состоять в использовании других методов библиотеки SuiteSparse для обновления графа и транзитивного замыкания.
- Реализовать восстановление путей и провести эксперименты для сравнения показателей с результатами алгоритма, основанного на матричном умножении, учитывая тот факт, что рассмотренный в работе алгоритм обладает возможностью восстанавливать все пути.

Работа проводилась в лаборатории языковых инструментов JetBrains-Research. Автор выражает благодарность сотрудникам лаборатории за всестороннюю помощь, а также за предоставленное оборудование для проведения экспериментов.

[6]Реализация: https://github.com/IlyaEp/RedisGraph/tree/IlyaEp_tensor. Дата посещения: 16.05.2020.

Список литературы

[1] Analysis of Recursive State Machines / Rajeev Alur, Michael Benedikt, Kousha Etessami et al. // ACM Trans. Program. Lang. Syst. — 2005. — Vol. 27, no. 4. — P. 786–818. — URL: https://doi.org/10.1145/1075382.1075387.

[2] Azimov Rustam, Grigorev Semyon. Graph Parsing by Matrix Multiplication // CoRR. — 2017. — Vol. abs/1707.01007. — 1707.01007.

[3] Context-Free Path Queries on RDF Graphs / Xiaowang Zhang, Zhiyong Feng, Xin Wang, Guozheng Rao // CoRR. — 2015. — Vol. abs/1506.00743. — 1506.00743.

[4] Evaluation of the Context-Free Path Querying Algorithm Based on Matrix Multiplication / Nikita Mishin, Iaroslav Sokolov, Egor Spirin et al. // Proceedings of the 2Nd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA). — GRADES-NDA'19. — New York, NY, USA : ACM, 2019. — P. 12:1–12:5. — URL: http://doi.acm.org/10.1145/3327964.3328503.

[5] Mathematical Foundations of the GraphBLAS / Jeremy Kepner, Peter Aaltonen, David A. Bader et al. // CoRR. — 2016. — Vol. abs/1606.05790. — 1606.05790.

[6] RedisGraph GraphBLAS Enabled Graph Database / Pieter Cailliau, Tim Davis, Vijay Gadepally et al. // CoRR. — 2019. — Vol. abs/1905.01294. — 1905.01294.

[7] Reps Thomas. Program Analysis via Graph Reachability // Proceedings of the 1997 International Symposium on Logic Programming. — ILPS '97. — Cambridge, MA, USA : MIT Press, 1997. — P. 5–19. — URL: http://dl.acm.org/citation.cfm?id=271338.271343.

[8] Sevon Petteri, Eronen Lauri. Subgraph queries by context-free grammars // Journal of Integrative Bioinformatics : JIB. — 2008. — Vol. 5, no. 100, 16 s.

[9] Yannakakis Mihalis. Graph-theoretic Methods in Database Theory // Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. — PODS '90. — New York, NY, USA : ACM, 1990. — P. 230–242. — URL: http://doi.acm.org/10.1145/298514.298576.

[10] An experimental study of context-free path query evaluation methods / Jochem Kuijpers, George Fletcher, Nikolay Yakovets, Tobias Lindaaker // Proceedings of the 31st International Conference on Scientific and Statistical Database Management, SSDBM 2019 / Ed. by Tanu Malik, Carlos Maltzahn, Ivo Jimenez. — United States : Association for Computing Machinery, Inc, 2019. — 7. — P. 121–132.