Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 19Б.10-мм

Порсев Денис Витальевич

Разработка алгоритма для задачи достижимости с регулярными ограничениями

Отчёт по учебной практике
в форме «Решение»

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

Санкт-Петербург
2022

Оглавление

1. Введение

При работе с графом зачастую требуется найти в нем множество путей от одной или нескольких вершин к другим. В случае, когда важно лишь знать о существовании этих путей, говорят, что требуется решить задачу достижимости в графе. При этом на множество искомых путей могут накладываться некоторые ограничения, заданные с помощью формальных языков. В итоге в помеченном графе достижимыми будут считаться только те вершины, путь к которым будет образовывать слова, принадлежащие заданному формальному языку.

Для формального описания свойств меток путей в графе используются регулярные и контекстно-свободные грамматики. Регулярные ограничения на пути в графе стали ключевым компонентом языков запросов к графовым базам данных [9]. С помощью контекстно-свободных языков можно выразить более широкий класс ограничений, по этой причине они нашли широкое применение и в других областях таких, как биоинформатика [12], статический анализ кода [11] и др.

Несмотря на то, что регулярные и контекстно-свободные запросы к базам данных активно используются на практике, существующие алгоритмы показывают невысокую производительность на реальных входных данных, что было показано в работе [7] для контекстно-свободных грамматик и в работе [14] для регулярных грамматик. В работе [14] было исследовано два основных подхода к регулярным запросам и выявлено, что каждый из этих подходов эффективен только для конкретных видов графов.

В то же время, актуальным становится применение классических алгоритмов анализа графов, представленных с помощью операций линейной алгебры, исследования которых показывают их эффективность на реальных данных. Благодаря их популярности появляются стандарты [13] такие, как GraphBLAS [5], определяющие примитивы для реализации алгоритмов в терминах линейной алгебры. Существует множество реализаций этих примитивов, которые удобно использовать для создания высокоэффективных алгоритмов.

Как и классические алгоритмы поиска путей в графе, алгоритмы достижимости с формальными ограничениями разделяются на несколько видов. Алгоритмы для одной исходной вершины (single source), нескольких исходных вершин (multiple source) или всех вершин (all paths). Алгоритмы для нескольких вершин являются менее изученными, при этом имеют активно применяются в графовых базах данных.

Тем самым, на основе классических алгоритмов анализа графов, кажется возможным построить эффективную реализацию решения задачи достижимости с ограничениями в виде формальных языков. Разработка такого алгоритма для нескольких стартовых вершин позволит глубже исследовать производительность матричных алгоритмов для решения задачи достижимости с несколькими стартовыми вершинами, а также создать высокоэффективное решение, применимое к реальным графовым данным.

Таким образом, в рамках данной работы проводится разработка алгоритма для задачи регулярной достижимости для нескольких стартовых вершин. Главной особенностью алгоритма является то, что он основан на операциях линейной алгебры и за его основу взят один из классических алгоритмов анализа графов — обход в ширину. Предлагается реализация алгоритма и проводится экспериментальное исследование эффективности разработанного решения.

2. Постановка задачи

Целью данной работы является разработка алгоритма для задачи достижимости с регулярными ограничениями, который будет основан на операциях линейной алгебры и будет эффективно решать задачу для небольшого количества стартовых вершин.

Для выполнения этой цели в рамках данной работы были поставлены следующие задачи.

1. Провести обзор существующих алгоритмов, решающих задачу достижимости в графе с ограничениями в виде формальных языков.

2. Разработать алгоритм, решающий задачу достижимости в случае нескольких стартовых вершин для регулярных ограничений, в терминах линейной алгебры.

3. Реализовать разработанный алгоритм с помощью операций линейной алгебры.

4. Провести экспериментальное исследование эффективности реализованного алгоритма.

3. Обзор

В обзоре приводится основная терминология, используемая в работе. После чего рассматриваются существующие алгоритмы решающую задачу достижимости с формальными ограничениями. Подробнее разбираются алгоритмы, основанные на операциях линейной алгебры.

3.1. Основные термины

Определим основные объекты из теории формальных языков, которые используются в описании исследуемых и разрабатываемых алгоритмов, а также формально определим задачу достижимости с регулярными ограничениями.

Определение *Конечным автоматом* называется формальная система $\langle Q, \Sigma, P, Q_{src}, F \rangle$, где

- Q — конечное непустое множество состояний,
- Σ — конечный входной алфавит,
- P — отображение $Q \times \Sigma \to Q$,
- $Q_{src} \subset Q$ — множество начальных состояний,
- F — множество конечных состояний

Теперь определим грамматики, используемые для задания ограничений в графе.

Определение *Формальной грамматикой* называется четверка $\langle V_N, V_T, P, S \rangle$, где

- V_N, V_T — конечные и непересекающиеся алфавиты нетерминалов и терминалов соответственно,
- P — конечное множество правил,
- S — стартовый нетерминал.

Определение *Регулярной грамматикой* называется формальная грамматика, правила которой могут быть заданы как $A \to aB$, $A \to a$, либо как $A \to Ba$, $A \to a$ где $a \in V_T$, $A, B \in V_N$.

3.2. Формулировка задачи достижимости

Теперь сформулируем задачу регулярной достижимости (RPQ) и её частный случай для нескольких стартовых вершин.

Определение *Задачей регулярной достижимости в графе* называется следующая задача: имея помеченный граф D и регулярный язык G, требуется найти такое множество всех пар вершин, для которых существует хотя бы один путь от начальной вершины к конечной, что слово, полученной конкатенацией меток ребер графа D будет принадлежать данному регулярному языку G.

Для нескольких стартовых вершин задачу можно определить разными способами. При этом в каждом из этих определений входными данными являются граф D с метками на ребрах и регулярный язык, заданный грамматикой G. В графе выбирается некоторое множество начальных вершин V_{src}, из которого требуется найти достижимые вершины.

Определение *Постановка задачи 1.* Необходимо найти такое множество вершин графа, что для каждой вершины из этого множества существует хотя бы один путь, начало которого содержится в множестве начальных вершин. При этом метки на ребрах этого пути при конкатенации образуют слово, принадлежащее языку грамматики G.

Эту задачу можно переформулировать другим способом, желая найти конкретную исходную вершину для каждой достижимой вершины.

Определение *Постановка задачи 2.* Найти множество пар (v, w) вершин, такое что $v \in V_{src}$, $w \notin V_{src}$, существует хотя бы один путь из v в w такой, что метки на ребрах этого пути принадлежат языку грамматики G.

3.3. Основные алгоритмы RPQ

Можно выделить две основные группы алгоритмов RPQ: основанные на реляционной алгебре и конечных автоматах [4].

3.3.1. Использование реляционной алгебры

Этот подход анализирует регулярное выражение, содержащее информацию о запросе к графовой базе данных.

Для решения задачи достижимости используется специальный оператор реляционной алгебры, который вычисляет транзитивное замыкание на множестве вершин графа. Тогда, представив регулярное выражение в виде синтаксического дерева, можно использовать этот оператор для обхода дерева регулярного выражения и выявления достижимых вершин. При этом вычисление транзитивного замыкания и построение дерева на его основе это трудоемкие операции, которые сказываются на эффективности алгоритма.

3.3.2. Использование автоматов

Известно, что регулярное выражение можно представить в виде конечного автомата, принимающего тот же язык, что и регулярное выражение. Тогда, представив граф в виде автомата, обозначив его вершины за состояния, а ребра за переходы, можно получить два автомата, пересечение которых будет содержать информацию о достижимых вершинах. В автомате пересечения каждая пара, состоящая из начального состояния и соответствующего ему конечного состояния, будет образовывать искомое множество достижимых вершин в графе.

Для получение пар достижимых вершин не обязательно строить полный автомат пересечения, достаточно лишь устроить обход в ширину построенных изначально автоматов. Перед тем как совершить переходы в графе к новому фронту вершин путь до текущих вершин во фронте проверяется на автомате регулярного выражения. Тогда в новый фронт обхода графа попадают только те вершины, которые были приняты автоматом.

Данный подход можно оптимизировать, совершая обход одновременно по двум автоматам. Более того, известно как представить обход в ширину с помощью операций линейной алгебры на основе матричного умножения, что на больших графах может существенно ускорить описанный алгоритм.

3.4. Алгоритмы, основанные на линейной алгебре

К настоящему времени существует лишь несколько достаточно эффективных реализаций матричных алгоритмов поиска путей в графе с формальными ограничениями. Многие из них были разработаны для контекстно-свободных грамматик и будут разобраны в этом разделе. Для регулярных грамматик аналогичных матричных алгоритмов найдено не было.

3.4.1. Алгоритмы для всех пар вершин

В упомянутом [7] сравнительном исследовании алгоритмов был исследован алгоритм Рустама Азимова [1]. За основу этого алгоритма взято вычисление транзитивного замыкания.

Алгоритм принимает на вход граф и контекстно-свободную грамматику, выраженную в ослабленной нормальной форме Хомского. Далее он оперирует над множеством булевых матриц, соответствующих каждому нетерминальному символу. Тем самым, при реализации алгоритма большое влияние на скорость алгоритма влияет использование эффективных библиотек примитивов линейной алгебры.

Помимо этого алгоритма в работе [2] был разработан алгоритм основанный на тензорном произведении. Он также использует операции матричного умножения, но в отличии от предыдущего алгоритма не требует модификации изначальной контекстно-свободной грамматики. Этот алгоритм основывается на вычислении пересечения автоматов, каждый из которых выражает представление графа и грамматики соответственно.

На вход алгоритму подается конечный автомат, описывающий сам

граф, где вершины являются состояниями, а ребра описывают переходы в автомате. Вторым аргументом алгоритм получает рекурсивный автомат, описывающий ограничения на метки в графе. Известно, что благодаря тензорному произведению можно вычислить пересечение двух автоматов, именно с помощью этой операции строится новая матрица, описывающая переходы в новом автомате. После чего полученный автомат пересечения транзитивно замыкается, чтобы его переходы не содержали нетерминальных символов. Эти операции применяются в цикле для матриц смежности входных автоматов пока любая из матриц смежности автомата графа меняется.

Операции вычисления транзитивного замыкания и тензорного произведения находят пути в графе для всех пар вершин. По этой причине наивная реализация алгоритмов для нескольких стартовых вершин этими методами будет неэффективна, так как все вершины графа буду считаться начальными и для всех них посчитаются пути до других вершин. В связи с этим возникает идея отказаться от вычисления тензорного произведения и использовать классические алгоритмы обхода графа, такие как обход в ширину, для матричного представления автоматов. Для того, чтобы вычислять их пересечение, нужно синхронизировать обход в ширину для соответствующих матриц.

3.4.2. Алгоритмы для нескольких стартовых вершин

Используя различные алгебраические структуры, можно модернизировать представленные алгоритмы для решения задачи достижимости с несколькими стартовыми вершинами. Так, например, можно ограничить вычисляемые пути, производя поиск только по нескольким стартовым вершинам, и не хранить информацию об остальных вершинах при обходе графа. В одной из таких работ [8] была представлена модификация алгоритма Рустама Азимова, созданная для нескольких стартовых вершин и встроенная в графовую базу данных.

Из всего этого следует, что ведется активная работа в исследовании алгоритмов поиска путей в графе с формальными ограничениями. На основе уже известных алгоритмов, таких как пересечение двух

автоматов графа и грамматики через их синхронный обход, кажется возможным создать эффективный алгоритм для нескольких стартовых вершин, основанный на классических алгоритмах обхода графа, используя при этом менее затратные алгебраические операции и оптимизируя его под конкретную задачу. Тем не менее, остается важным использовать специализированные матричные библиотеки для ускорения вычисления операций линейной алгебры.

4. Предлагаемое решение

В данном разделе представлено подробное описание разработанного алгоритма. Автором предлагается новый алгоритм, основанный на классическом матричном алгоритме обхода в ширину для нескольких стартовых вершин [6]. Он использует операции умножения матрицы на матрицу, применение масок к полученному результату для выделения текущего фронта обхода и другие классические матричные операций стандарта GraphBLAS. В конце рассматривается модификация этого алгоритма для решения второй поставленной задачи.

Далее представлено описание алгоритма, решающего первую поставленную задачу.

4.1. Входные данные

Алгоритм принимает на вход граф $\mathcal{G}$, детерминированный конечный автомат $\mathcal{R}$, описывающий регулярную грамматику, и множество начальных вершин V_{src} графа.

Граф $\mathcal{G}$ и автомат $\mathcal{R}$ можно представить в виде булевых матриц смежности. Так, в виде словаря для каждой метки графа заводится булева матрица смежности, на месте (i, j) ячейки которой стоит 1, если i и j вершины графа соединены ребром данной метки. Такая же операция проводится для автомата грамматики $\mathcal{R}$.

Далее, мы оперируем с двумя словарями, где ключом является символ метки ребра графа или символ алфавита автомата, а значением — соответствующая им булевая матрица.

Для каждого символа из пересечения этих множеств строится матрица $\mathfrak{D}$, как прямая сумма булевых матриц. То есть, строится матрица $\mathfrak{D} = Bool_{\mathcal{R}_a} \bigoplus Bool_{\mathcal{G}_a}$, которая определяется как

$$\mathfrak{D} = \begin{bmatrix} Bool_{\mathcal{R}_a} & 0 \\ 0 & Bool_{\mathcal{G}_a} \end{bmatrix} \tag{1}$$

Где $\mathcal{R}_a$ и $\mathcal{G}_a$ матрицы смежности соответствующих символов автомата грамматики $\mathcal{R}$ и графа $\mathcal{G}$ для символа $a \in A_{\mathcal{R}} \cap A_{\mathcal{G}}$, $A_{\mathcal{R}} \cap A_{\mathcal{G}}$ — пе-

ресечение алфавитов. Такая конструкция позволяет синхронизировать алгоритм обхода в ширину одновременно для графа и грамматики.

Далее вводится матрица M, хранящая информацию о фронте обхода. Она нужна для выделения множества пройденных вершин и не допускает зацикливание алгоритма.

$$M^{k\times(k+n)} = \begin{bmatrix} Id_k & Matrix_{k\times n} \end{bmatrix} \tag{2}$$

Где Id_k — единичная матрица размера k, k — количество вершин в автомате $\mathcal{R}$, $Matrix_{k\times n}$ — матрица, хранящая в себе маску пройденных вершин в автомате графа, n — количество вершин в графе $\mathcal{G}$.

4.2. Выходные данные

На выходе строится множество $\mathcal{P}$ пар вершин (v, w) графа $\mathcal{G}$ таких, что вершина w достижима из множества начальных вершин, при этом $v \in V_{src}$, $w \notin V_{src}$. Это множество представляется в виде матрицы размера $|V| \times |V|$, где (i, j) ячейка содержит 1, если пара вершин с индексами $(i, j) \in \mathcal{P}$.

4.3. Процесс обхода графа

Алгоритм обхода заключается в последовательном умножении матрицы M текущего фронта на матрицу $\mathfrak{D}$. В результате чего, находится матрица M' содержащая информацию о вершинах, достижимых на следующем шаге. Далее, с помощью операций перестановки и сложения векторов M' преобразуется к виду матрицы M и присваивается ей. Итерации продолжаются пока M' содержит новые вершины, не содержащиеся в M. На листинге 1 представлен этот алгоритм.

В алгоритме 1, в 10 строке происходит трансформация строчек в матрице M'. Это делается для того, чтобы представить полученную во время обхода матрицу M', содержащую новый фронт, в виде матрицы M. Для этого требуется так переставить строчки M', чтобы она содержала корректные по своему определению значения. То есть, имела

Algorithm 1 Алгоритм достижимости в графе с регулярными ограничениями на основе поиска в ширину, выраженный с помощью операций матричного умножения

```
1: procedure BFSBasedRPQ(R = ⟨Q, Σ, P, F, q⟩, G = ⟨V, E, L⟩, V_src)
2:     P ← Матрица смежности графа
3:     D ← Bool_R ⊕ Bool_G                          ▷ Построение матриц D
4:     M ← CreateMasks(|Q|, |V|)                    ▷ Построение матрицы M
5:     M' ← SetStartVerts(M, V_src)                 ▷ Заполнение нач. вершин
6:     while Матрица M меняется do
7:         M ← M'⟨¬M⟩                               ▷ Применение комплементарной маски
8:         for all a ∈ (Σ ∩ L) do
9:             M' ← M any.pair D                    ▷ Матр. умножение в полукольце
10:            M' ← TransformRows(M')               ▷ Приведение M' к виду M
11:        Matrix ← extractRightSubMatrix(M')
12:        V ← Matrix.reduceVector()                ▷ Сложение по столбцам
13:        for k ∈ 0 ... |V_src| − 1 do
14:            W ← P.getRow(k)
15:            P.setRow(k, V + W)
16:    return P
```

единицы на главной диагонали, а все остальные значения в первых k столбцах были нулями. Подробнее эта процедура описана в листинге 2.

Algorithm 2 Алгоритм трансформации строчек

```
1: procedure TransformRows(M)
2:     T ← extractLeftSubMatrix(M)
3:     Ix, Iy ← итераторы по индексам ненулевых элементов T
4:     for i ∈ 0 ... |Iy| do
5:         R ← M.getRow(Ix[i])
6:         M'.setRow(Iy[i], R + M'.getRow(Iy[i]))
```

Algorithm 3 Модификация алгоритма для поиска конкретной исходной вершины

```
1: procedure BFSBasedRPQ(𝓡 = ⟨Q, Σ, P, F, q⟩, 𝓖 = ⟨V, E, L⟩, V_src)
2:     𝒫 ← Матрица смежности графа
3:     𝔇 ← Bool_𝓡 ⨁ Bool_𝓖
4:     𝔐 ← CreateMasks(|Q|, |V|)
5:     𝔐′ ← SetStartVerts(𝔐, V_src)
6:     while Матрица 𝔐 меняется do
7:         𝔐 ← 𝔐′⟨¬𝔐⟩
8:         for all a ∈ (Σ ∩ L) do
9:             𝔐′ ← 𝔐 any.pair 𝔇
10:            for all M ∈ 𝔐′ do
11:                M ← TransformRows(M)
12:        for all M_k ∈ 𝔐′ do
13:            Matrix ← extractSubMatrix(M)
14:            V ← Matrix.reduceVector()
15:            W ← 𝒫.getRow(k)
16:            𝒫.setRow(k, V + W)
17:    return 𝒫
```

4.4. Модификации алгоритма

Рассмотрим V_{src} — множество начальных вершин, состоящее из r элементов. Для каждой начальной вершины $v^i_{src} \in V_{src}$ отметим соответствующие индексы в матрице M единицами, получив матрицу $M(v^i_{src})$, и построим матрицу $\mathfrak{M}$ следующим образом.

$$\mathfrak{M}^{(k*r)\times(k+n)} = \begin{bmatrix} M(v^1_{src}) \\ M(v^2_{src}) \\ M(\dots) \\ M(v^r_{src}) \end{bmatrix} \tag{3}$$

Матрица $\mathfrak{M}$ собирается из множества матриц $M(v^i_{src})$ и позволяет хранить информацию о том, из какой начальной вершины достигаются новые вершины во время обхода.

В листинге 3 представлен модифицированный алгоритм. Основное его отличие заключается в том, что для каждой достижимой вершины находится конкретная исходная вершина, из которой начинался обход.

Таким образом, алгоритмы 1 и 3 решают сформулированные в пункте 3.2 задачи достижимости.

5. Эксперимент

В этом разделе представлены результаты замеров производительности разработанного решения.

5.1. Условия эксперимента

При проведении экспериментального исследования использовался компьютер со следующими характеристиками: процессор Intel Core i7-10750H 6 × 2.60 GHz, 16 Gb DDR4 RAM.

В качестве исходных данных были взяты RDF графы из репозитория CFPQ_Data, которые собраны на реальных данных. Из репозитория выбирались графы, которые содержат хотя бы 10 уникальных меток на ребрах. Это условие требовалось для генерации более сложных запросов к этим графам. Такому условию удовлетворяют в большинстве своем графы, собранные на биологических данных.

Характеристики выбранных графов представлены в таблице 1.

Таблица 1: Датасет

Graph	# Vertices	# Edges
core	1 323	2 752
enzyme	48 815	86 543
eclass	239 111	360 248
go	582 929	1 437 437

Регулярные запросы к графам были сгенерированы на основе самых популярных шаблонов запросов, собранных в работе [10]. Для каждого графа отбиралось 10 самых популярных меток, после чего по ним на основе шаблонов генерировалось случайным образом 5 запросов. В общей сложности по 23 шаблонам было сгенерировано 115 запросов к каждому графу.

В ходе проведения эксперимента каждый запрос исполнялся 5 раз, после чего вычислялось среднее значение исполненных запросов.

Всего было произведено 6 экспериментов: 1 эксперимент оценивающий эффективность алгоритма для одной исходной вершины (single

source) и 5 экспериментов для нескольких стартовых вершин (multiple source). Для single source эксперимента 1000 раз, не повторяясь, выбиралась одна стартовая вершина. Для multiple source эксперимента выбиралось случайное множество, состоящее из 10, 50, 100, 500 или 1000 стартовых вершин.

5.2. Результаты

Результаты эксперимента представлены на рисунке 1, где штрихами обозначены медианы для каждого из запросов.

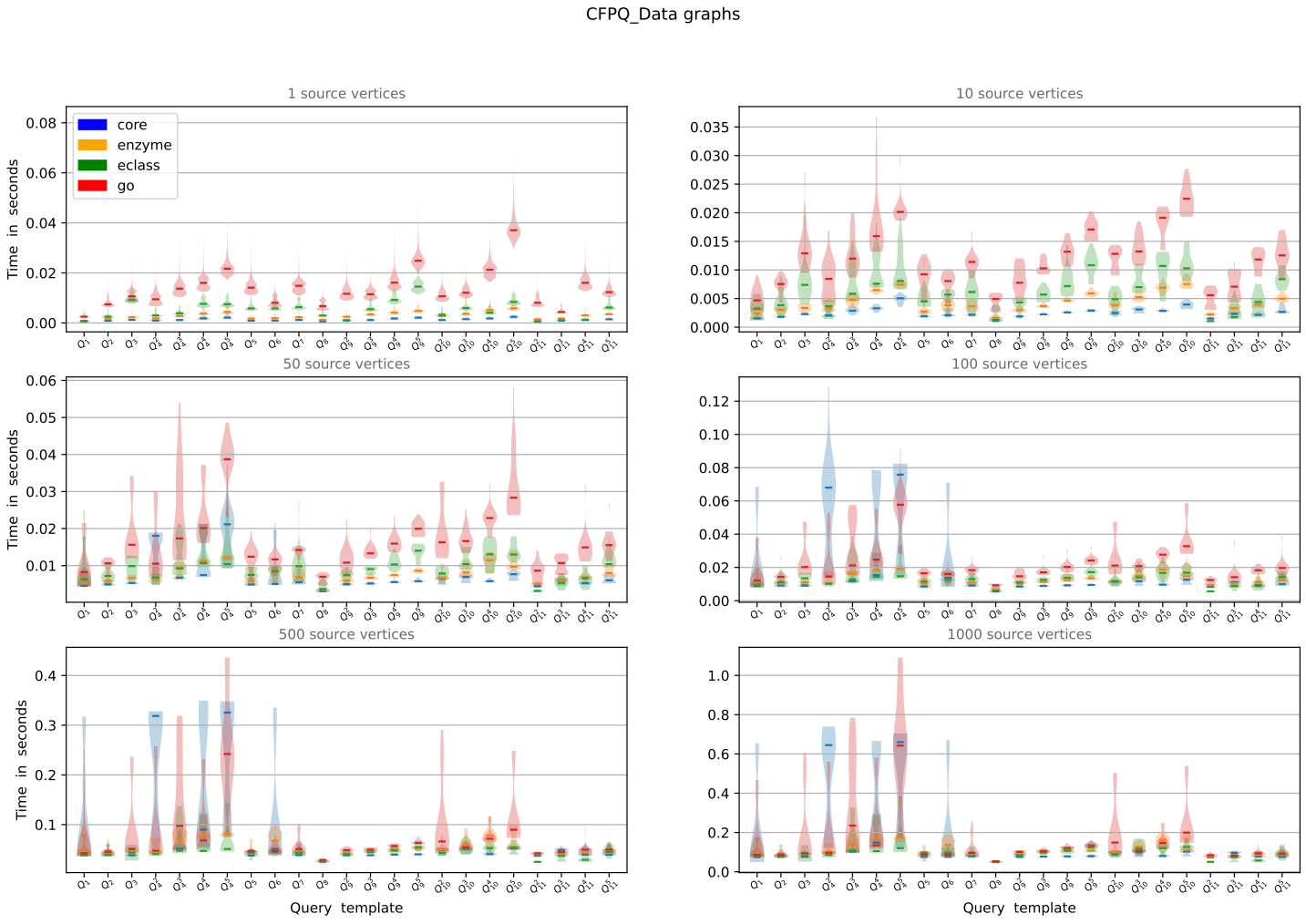

Рис. 1: Результаты экспериментов

На них продемонстрировано, что разработанный алгоритм достигает скорости работы в доли секунды, что соответствует времени работы

аналогичных алгоритмов в научных статьях [3]. Это говорит о том, что разработанное решение является конкурентным среди аналогов и может быть эффективно использовано в решении практических задач. Более того, можно сделать следующие выводы:

- Производительность алгоритма зависит от сложности запроса. Так, запросы по шаблонам Q_4^4, Q_4^5 и Q_{10}^4, Q_{10}^5, содержащие в себе большее количество меток, чем остальные запросы, исполняются в среднем дольше остальных.

- Не всегда время исполнения запроса напрямую зависит от размера графа. Так, при существенном увеличении числа стартовых вершин по отношению к общему числу вершин, алгоритм показывает меньшую производительность. Это заметно на графиках для 100, 500 и 1000 вершин, где вычисления для графа *core* производятся дольше, чем для других графов, имеющих большее число вершин.

- Время исполнения запроса редко превышает одну секунду и существенно растет при увеличении числа стартовых вершин.

6. Заключение

В рамках данной работы были получены следующие результаты:

- Проведен обзор предметной области и, в частности, алгоритмов, основанных на методах линейной алгебры, решающих задачу достижимости с ограничениями в виде формальных языков.
- Разработан алгоритм, решающий задачу достижимости для регулярных грамматик в терминах линейной алгебры.
- Написана реализация[1] алгоритма с помощью операций линейной алгебры.
- Проведено экспериментальное исследование разработанного решения.

В дальнейшем работа может быть развита в следующих направлениях:

- Проведение экспериментального исследования на более широком наборе данных, который включал бы в себя искусственно сгенерированные графы, реальные графы большего размера и графы из других прикладных областей.
- Полученное решение может быть интегрировано в графовые базы данных, что позволит провести полноценное сравнение алгоритма с аналогами и лучше оценить его эффективность.

[1] https://github.com/JetBrains-Research/CFPQ_PyAlgo/pull/34

Список литературы

[1] Azimov Rustam, Grigorev Semyon. Context-Free Path Querying by Matrix Multiplication // Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences amp; Systems (GRADES) and Network Data Analytics (NDA). — GRADES-NDA '18. — New York, NY, USA : Association for Computing Machinery, 2018. — 10 p. — URL: https://doi.org/10.1145/3210259.3210264.

[2] Context-Free Path Querying by Kronecker Product / Egor Orachev, Ilya Epelbaum, Rustam Azimov, Semyon Grigorev. — 2020. — 08. — P. 49–59.

[3] Distributed Pregel-Based Provenance-Aware Regular Path Query Processing on RDF Knowledge Graphs / Xin Wang, Simiao Wang, Yueqi Xin et al. // World Wide Web. — 2020. — may. — Vol. 23, no. 3. — P. 1465–1496. — URL: https://doi.org/10.1007/s11280-019-00739-0.

[4] Angela Bonifati, George Fletcher, Hannes Voigt et al. // Querying Graphs. — 2018. — P. 131–134.

[5] GraphBLAS community. The GraphBLAS. Welcome to the GraphBLAS Forum. — 2021. — [Online; accessed 24-February-2022]. URL: https://graphblas.org/.

[6] A GraphBLAS solution to the SIGMOD 2014 Programming Contest using multi-source BFS / Márton Elekes, Attila Nagy, Dávid Sándor et al. // 2020 IEEE High Performance Extreme Computing Conference (HPEC). — 2020. — P. 1–7.

[7] Kuijpers Jochem. An Experimental Study of Context-Free Path Query Evaluation Methods // SSDMB '19. — 2019. — . — Vol. 1. — P. 121–132. — URL: https://doi.org/10.1145/3335783.3335791.

[8] Multiple-Source Context-Free Path Querying in Terms of Linear Algebra / Arseniy Terekhov, Vlada Pogozhelskaya, Vadim Abzalov et al. // EDBT. — 2021.

[9] Nolé Maurizio, Sartiani Carlo. Regular Path Queries on Massive Graphs // Proceedings of the 28th International Conference on Scientific and Statistical Database Management. — SSDBM '16. — New York, NY, USA : Association for Computing Machinery, 2016. — 12 p. — URL: https://doi.org/10.1145/2949689.2949711.

[10] Shemetova Ekaterina, Azimov Rustam, Orachev Egor et al. One Algorithm to Evaluate Them All: Unified Linear Algebra Based Approach to Evaluate Both Regular and Context-Free Path Queries. — 2021. — URL: https://arxiv.org/abs/2103.14688.

[11] 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: https://doi.org/10.1145/3335783.3335791.

[12] Sevon Petteri, Eronen Lauri. Subgraph Queries by Context-free Grammars // Journal of Integrative Bioinformatics. — 2008. — Vol. 5, no. 2. — P. 157–172. — URL: https://doi.org/10.1515/jib-2008-100.

[13] Standards for graph algorithm primitives / Tim Mattson, David Bader, Jon Berry et al. // 2013 IEEE High Performance Extreme Computing Conference (HPEC). — 2013. — Sep. — P. 1–2.

[14] Yakovets Nikolay, Godfrey Parke, Gryz Jarek. Query Planning for Evaluating SPARQL Property Paths // Proceedings of the 2016 International Conference on Management of Data. — SIGMOD '16. — New York, NY, USA : Association for Computing Machinery, 2016. — P. 1875–1889. — URL: https://doi.org/10.1145/2882903.2882944.