Санкт-Петербургский государственный университет

Программная инженерия

Абзалов Вадим Игоревич

Разработка и реализация инкрементального алгоритма решения динамического варианта задачи достижимости с контекстно-свободными ограничениями

Курсовая работа

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

Санкт-Петербург
2020

Оглавление

Введение

Графовое представление данных широко используется во многих областях. Например, внутри баз данных [4], в биоинформатике [5], в статическом анализе и т.д. В этих областях часто требуется обрабатывать запросы на довольно больших графах. Одним из наиболее распространенных типов запросов на графах является навигационный запрос. Результат обработки такого запроса, как правило, представляет собой набор отношений между узлами графа, то есть набор некоторых путей. Естественный способ указать эти отношения — указать пути, используя регулярные выражения или контекстно-свободные грамматики над алфавитом меток ребер. В настоящее время в запросах к графовым базам данных активно развивается применение именно контекстно-свободных грамматик во многом из-за ограниченной выразительной силы регулярных выражений.

На сегодняшний день графы также широко используются при моделировании изменяющихся во времени реальных процессов, таких, как взаимоотношение между людьми или взаимодействие нейронов человеческого мозга. В терминах графов изменения в этих моделях выражаются через операции добавления и удаления вершин и/или ребер.

С другой стороны проблемы достижимости с контекстно-свободными ограничениями сосредоточены на существовании путей в помеченных графах, символы на ребрах которых образуют слова, принадлежащие данной контекстно-свободной грамматике. Такие задачи находятся на границе двух областей. Так, теория формальных языков связана с обработкой языка как множества слов в отношении различных вопросов: является ли язык пустым, конечным или бесконечным? Содержит ли язык данное слово? В то время, как проблемы достижимости касаются существования путей в графах и рассматривают такие вопросы, как: существует ли в графе путь между двумя заданными вершинами? Насколько длинным может быть такой путь?

Динамическая формулировка проблемы достижимости с контекстно-свободными ограничениями сосредоточена на таких важных вопросах,

как: существует ли в графе путь между двумя заданными вершинами, удовлетворяющий контекстно-свободной грамматике? Как изменится такой, если добавить ребро в граф? Если удалить ребро в графе? Решение такого рода задача зачастую сосредоточены на эффективном обновлении результатов запроса в условиях, когда граф может активно изменяться.

Конечно, многие динамические постановки классических графовых задач уже изучаются [3]. Также изучаются и классические вопросы в области контекстно-свободной достижимости [2]. Но на практике существуют только статичные решения, то есть те, которые решают статичную постановку задачи. Одним из наиболее перспективных решений в этой области является матричный алгоритм Рустама Азимова [1].

Во многих реальных ситуациях, моделируемых с помощью графов, зачастую вершины и ребра только добавляются, для них нет потребности в построении алгоритма, реализующего операции удаления. Поэтому данный класс задач представляет особый интерес для настоящей работы. А именно, данная работа фокусируется на разработке и реализации инкрементального алгоритма решения динамического варианта задачи достижимости с контекстно-свободными ограничениями. Суть решаемой задачи заключается в построении и корректном поддержании ответа на контекстно-свободный запрос в ситуациях, когда в граф добавляются ребра и вершины.

1. Постановка задачи

Существует как минимум четыре вида алгоритмов для работы с динамически обновляемыми графами.

- Полностью динамический алгоритм — алгоритм, который может обрабатывать как операции добавления вершин и/или ребер, так и операции удаления вершин и/или ребер.
- Инкрементальный алгоритм — алгоритм, который поддерживает только операции добавления.
- Декрементальный алгортим — алгоритм, который поддерживает только операции удаления.
- Частично динамический алгортм — алгоритм, который является либо инкрементальным, либо декрементальным.

Целью данной работы является разработка частично динамического алгоритма решения задачи достижимости с контекстно-свободными ограничениями. А именно — разработка инкрементального алгоритма.

Для достижения поставленной цели были выделены две задачи.

- Разработка и доказательство корректности инкрементального алгоритма решения динамического варианта задачи достижимости с контекстно-свободными ограничениями.
- Реализация разработанного алгоритма.
- Экспериментальное исследование разработанного алгоритма.

2. Обзор

Для более глубокого понимания сути разрабатываемого алгоритма важно подробно исследовать предметную область. В частности, особенный интерес представляют результаты, полученные в классической теории графов и результаты, полученные в теории графов с контекстно-свободными ограничениями.

Очень важно понимать существующие решения в классической области вопросов динамической достижимости, то есть без контекстно-свободных ограничений. Так, например, в работе Dynamic shortest paths and transitive closure: Algorithmic techniques and data structures [3] рассматриваются полностью динамические алгоритмы для задач на ориентированных графах. В частности, проводится полный обзор решений двух фундаментальных проблем: проблемы построения и поддержания динамического транзитивного замыкания и проблемы динамического поиска кратчайших путей. Авторами этой статьи приложены особые усилия, чтобы абстрагировать некоторые комбинаторные и алгебраические свойства поставленных задач и важные распространенные инструменты для структурирования данных, которые лежат в основе этих методов. Описаны полностью динамические алгоритмы нахождения кратчайших путей и транзитивного замыкания в графе на основе вспомогательной древовидной структуры данных и на основе матричного представления. Решения и инструменты, представленные в статье, могут быть полезны при разработке алгоритма решения поставленной задачи, как методы, доказавшие свою важность и применимость в классической теории графов.

Кроме того, как уже упоминалось, активно изучается и область достижимости с контекстно-свободными ограничениями. Очень важный результат получен в работе Dynamic Complexity of the Dyck Reachability [2]. Там изучается проблема достижимости в графах с частным случаем контекстно-свободных ограничений — языками Дика. Рассматривается ориентированное и неориентированное представления помеченных графов. А также доказывается, что существует сильная дихотомия (то

есть сильное различие) между динамической сложностью таких задач в зависимости от размера алфавита языка Дика. Необходимо подчеркнуть, что также в этой статье определяется и доказывается асимптотическая сложности задачи динамического транзитивного замыкания с контекстно-свободными ограничениями. Так как изучается частный случай более общей задачи, то можно сделать выводы о том, что ожидаемая асимптотическая сложность разрабатываемого алгоритма будет не лучше, чем сложность частного случая.

В тоже время в области вопросов статичной достижимости с контекстно-свободными, существуют решения, показавшие свою эффективность на практике. Так, в работе Context-free path querying by matrix multiplication [1] предлагается первое обобщение алгоритма Лесли Вэлианта [6], основанного на матричном представлении графа, для контекстно-свободных запросов достижимости. Представленное обобщение не дает по-настоящему субкубического алгоритма (в худшем случае), существование которого до сих пор остается слабо раскрытым вопросом в этой области. Однако важная идея использования матричных операций (таких, как умножение матриц) в процессе обработки контекстно-свободных запросов достижимости позволяет применить богатый класс методов оптимизации, таких как применение GPGPU (вычисления общего назначения на графическом процессоре), параллельная обработка, разреженное матричное представление и т.д. Также в статье предоставляются сравнительные результаты работы алгоритма на наборе традиционных тестов, которые показывают, что алгоритм значительно превосходит по производительности существующие решения. Полученные результаты позволяют выдвинуть гипотезу о том, что данный алгоритм может быть распространен на динамический случай без потери эффективности и возможностей для оптимизаций. Поэтому крайне важно рассмотреть возможность построения динамического алгоритма, основываясь на полученных в работе результатах.

Таким образом, проведенный обзор позволяет сформулировать направление разработки алгоритма решения поставленной задачи. Исходя из результатов, полученных в работе [2], можно предположить воз-

можность обобщения полученных оценок динамической сложности на динамическую задачу с общим видом контекстно-свободной грамматики. А результаты, полученные в публикации [1], позволяют выразить надежду на практическую эффективность разрабатываемого алгоритма.

3. Определения

Определение 3.1. *Помеченный ориентированный граф* — это пара вида $D = (V, E)$, где

- V – множество вершин графа;
- E – множество помеченных ребер графа, причем $E \subseteq V \times \Sigma \times V$;

Множество вершин графа D обозначим как $V(D)$. Аналогично множество ребер обозначим как $E(D)$.

Определение 3.2. Для пути π в графе D определим слово, получаемое конкатенацией меток ребер этого пути, как $l(\pi)$. Также будем писать $n\pi m$, обозначая то, что путь π начинается в вершине $n \in V$ и заканчивается в вершине $m \in V$. Множество ребер пути π будем обозначать как $E(\pi)$.

Определение 3.3. *Контекстно-свободная грамматика* — это четвёрка вида $\langle \Sigma, N, P, S \rangle$, где

- Σ — это терминальный алфавит;
- N — это нетерминальный алфавит;
- P — это множество правил или продукций, таких что каждая продукция имеет вид $N_i \to \alpha$, где $N_i \in N$ и $\alpha \in \{\Sigma \cup N\}^* \cup \varepsilon$;
- S — стартовый нетерминал.

Отметим, что $\Sigma \cap N = \varnothing$.

Определение 3.4. Контекстно-свободная грамматика $\langle \Sigma, N, P, S \rangle$ находится в *Нормальной Форме Хомского*, если она содержит только правила следующего вида:

- $A \to BC$, где $A, B, C \in N$, S не содержится в правой части правила;
- $A \to a$, где $A \in N, a \in \Sigma$;

- $S \to \varepsilon$;

Определение 3.5. Контекстно-свободная грамматика $\langle \Sigma, N, P, S \rangle$ находится в *ослабленной Нормальной Форме Хомского*, если она содержит только правила следующего вида:

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

То есть ослабленная НФХ отличается от НФХ тем, что:

1. ε может выводиться из любого нетерминала;
2. S может появляться в правых частях правил;

Следуя статье [1], будем работать с *контекстно-свободной грамматикой без стартового нетерминала в ослабленной нормальной форме Хомского*, то есть с тройкой $\langle \Sigma, N, P \rangle$. Заметим, что данное определение отклоняется от классического определения контекстно-свободной грамматики в НФХ тем, что не рассматривается специальный стартовый нетерминал, который будет определен по ходу построения алгоритма. Также опускаются продукции вида $A \to \varepsilon$, где ε обозначает пустую строку, поскольку они соответствуют тривиальным путям вида $n\pi n$, которые удобнее учитывать отдельно.

Определение 3.6. *Язык, задаваемый контекстно-свободной грамматикой* — множество строк, выводимых в грамматике

$$L(G) = \{\omega \in \Sigma^* \mid S \Rightarrow^* \omega\}.$$

Определение 3.7. *Язык, задаваемый контекстно-свободной грамматикой с указанным стартовым нетерминалом A* — множество строк, выводимых в грамматике из нетерминала A

$$L(G_A) = \left\{ w \in \Sigma^* \mid A \xrightarrow{*} w \right\}.$$

Определение 3.8. $A \xrightarrow{*} w$ обозначает то, что строка $w \in \Sigma^*$ может быть получена из нетерминала A с помощью некоторой последовательности продукций из P.

Определение 3.9. *Контекстно-свободное отношение* для графа $D = (V, E)$, контекстно-свободной грамматики $G = (N, \Sigma, P)$ и зафиксированного стартового нетерминала A — это отношение $R_A \subseteq V \times V$ такое, что

$$R_A = \{(n, m) \mid \exists n \pi m \ (l(\pi) \in L(G_A))\}.$$

Также потребуется отношение, отражающее факт существования пути между двумя вершинами.

Определение 3.10. *Отношение достижимости* в графе: $(v_i, v_j) \in P \iff \exists v_i \pi v_j$.

Заметим, что отношение достижимости является транзитивным. Действительно, если существует путь из v_i в v_j и путь из v_j в v_k, то существует путь из v_i в v_k.

Один из способов задать граф — это задать его *матрицу смежности*.

Определение 3.11. *Матрица смежности* графа $\mathcal{G} = \langle V, E, L \rangle$ — это квадратная матрица M размера $n \times n$, где $|V| = n$ и ячейки которой содержат множества. При этом $l \in M[i, j] \iff \exists e = (i, l, j) \in E$.

Определение 3.12. *Транзитивным замыканием графа* называется транзитивное замыкание отношения достижимости по всему графу.

Как несложно заметить, транзитивное замыкание графа — это тоже граф.

Определение 3.13. Определим *бинарную операцию* $(\cdot)$ для произвольных подмножеств N_1, N_2 множества нетерминалов N контекстно-свободной грамматики $G = (N, \Sigma, P)$ как

$$N_1 \cdot N_2 = \{A \mid \exists B \in N_1, \exists C \in N_2 \text{ такие что } (A \to BC) \in P\}.$$

Определение 3.14. Определим *бинарную операцию* ($\cup$) для произвольных подмножеств N_1, N_2 множества нетерминалов N контекстно-свободной грамматики $G = (N, \Sigma, P)$ как стандартное объединение множеств

$$N_1 \cup N_2 = \{A \mid A \in N_1 \vee A \in N_2\}.$$

Определение 3.15. Используя ($\cdot$) как произведение, и ($\cup$) как сумму множеств, можем определить *произведение матриц* $A \times B = C$, где A и B матрицы подходящего размера, у которых элементы это некоторые подмножеств множества нетерминалов N контекстно-свободной грамматики $G = (N, \Sigma, P)$, как

$$C_{i,j} = \bigcup_{k=1}^{n} A_{i,k} \cdot B_{k,j}.$$

Определение 3.16. Будем использовать *поэлементную сумму матриц* A и B одинакового размера: $A + B = C$, где $C_{i,j} = A_{i,j} \cup B_{i,j}$ и *поэлементную разность матриц*: $A - B = C$, где $C_{i,j} = A_{i,j} \setminus B_{i,j}$.

Определение 3.17. *Транзитивным замыканием матрицы графа* называется матрица транзитивного замыкания этого графа. Будем обозначать транзитивное замыкание матрицы $A = \sum_i A^i$, как A^+.

4. Инкрементальный алгоритм

Во введенных обозначениях, задача динамической достижимости с контекстно-свободными ограничениями может быть сформулирована следующим образом. Для данного графа $D = (V, E)$ и контекстно-свободной грамматики $G = (N, \Sigma, P)$ необходимо поддерживать актуальные *контекстно-свободные отношения* R_A для любого $A \in N$ при двух операциях обновления графа D.

- $AddVertex(v)$ – добавление вершины v в граф D.
- $AddEdge(i, e, j)$ – добавление ребра (i, e, j) в граф D.

В реализации из статьи [1] можно заметить, что на очередной итерации алгоритма, многие посчитанные элементы матрицы транзитивного замыкания будут пересчитываться заново при обновлении графа. Значит, для построения инкрементального алгоритма необходимо как минимум научиться пересчитывать только те элементы матрицы, изменения которых, вызваны обновлением в графе. Это и будет ключевой идеей разрабатываемого алгоритма.

4.1. Построение алгоритма

Во введенных обозначениях, алгоритм может быть сформулирован следующим образом. Будем поддерживать транзитивное замыкание A^+ матрицы A графа D при операциях обновления $AddVertex$ и $AddEdge$.

Пусть в результате обновления графа D получен новый граф D' с матрицей A' и матрицей транзитивного замыкания $(A')^+$. Разберем отдельно каждую операцию с графом.

- $AddVertex(v)$ — добавление вершины в граф D изменяет лишь размерность матриц A и $^+$ и не влияет на сами элементы матриц. Следовательно, в этом случае, достаточно увеличить размерность матриц.

- $AddEdge(i, e, j)$ — добавление ребра в граф можно рассматривать как добавление ребра в соответствующее этому графу транзитивное замыкание. Тогда, достаточно пересчитать те ребра, которые добавятся в транзитивное замыкание, в следствие добавления ребра (i, e, j). Если обозначить B как матрицу, соответствующую графу G', у которого $V(G') = V(D')$ и $E(G') = \{(i, e, j)\}$, то вместо подсчета $(A')^+$ как $(A + B)^+$ достаточно вычислить $(A^+ + B)^+$, где A^+ уже была вычислена.

4.2. Корректность алгоритма

Теорема 4.1. Пусть в результате исполнения операции $AddVertex$ построена матрица A'' обновленного графа $D' = (V', E')$ с матрицей A' и матрицей транзитивного замыкания $(A')^+$. Тогда $\forall i, j \in V(D') \ : \ A''_{i,j} = (A')^+_{i,j}$.

Доказательство. Так как матрица A' получена в результате исполнения операции $AddVertex$, то

$$\forall i, \ j \in V(D) \ : \ A''_{i,j} = (A')^+_{i,j}$$

по определению процедуры $AddVertex(v)$. А также

$$\forall i \in V(D') \ : \ A''_{i,v} = A''_{v,i} = (A')^+_{i,v} = (A')^+_{v,i} = \{\varnothing\}$$

поскольку добавлялась только вершина v, а путей ведущих в нее в графе нет. Значит и в транзитивном замыкании нет путей ведущих в вершину v. Следовательно

$$\forall i, \ j \in V(D') \ : \ A''_{i,j} = (A')^+_{i,j}.$$

□

Теорема 4.2. Пусть в результате исполнения операции *AddEdge* построена матрица A'' обновленного графа $D' = (V', E')$ с матрицей A' и матрицей транзитивного замыкания $(A')^+$. Тогда $\forall i, j \in V(D') \ : \ A''_{i,j} = (A')^+_{i,j}$.

Доказательство. Так как матрица A'' получена в результате исполнения процедуры *AddEdge*, то докажем, что

$$((A')^+ + B)^+ = (A' + B)^+.$$

Заметим, что достаточно доказать, что

$$\nexists i, \ j \in V(D') \ : \ \exists X \in N \ : \ X \in ((A')^+ + B)_{i,j} \wedge X \notin ((A') + B)_{i,j}.$$

Что верно, по определению транзитивного замыкания графа по контекстно-свободной грамматике и

$$\forall i, \ j \in V(D') \ : \ ((A') + B)_{i,j} \subset ((A')^+ + B)_{i,j}.$$

Значит, все ребра из транзитивного замыкания матрицы $(A' + B)$ есть и в транзитивном замыкании матрицы $((A') + B)$ и, при этим, других ребер нет. То есть

$$\forall i, j \in V(D') \ : \ A''_{i,j} \subseteq (A')^+_{i,j} \ \text{ и}$$
$$\forall i, j \in V(D') \ : \ A''_{i,j} \supseteq (A')^+_{i,j}.$$

Следовательно

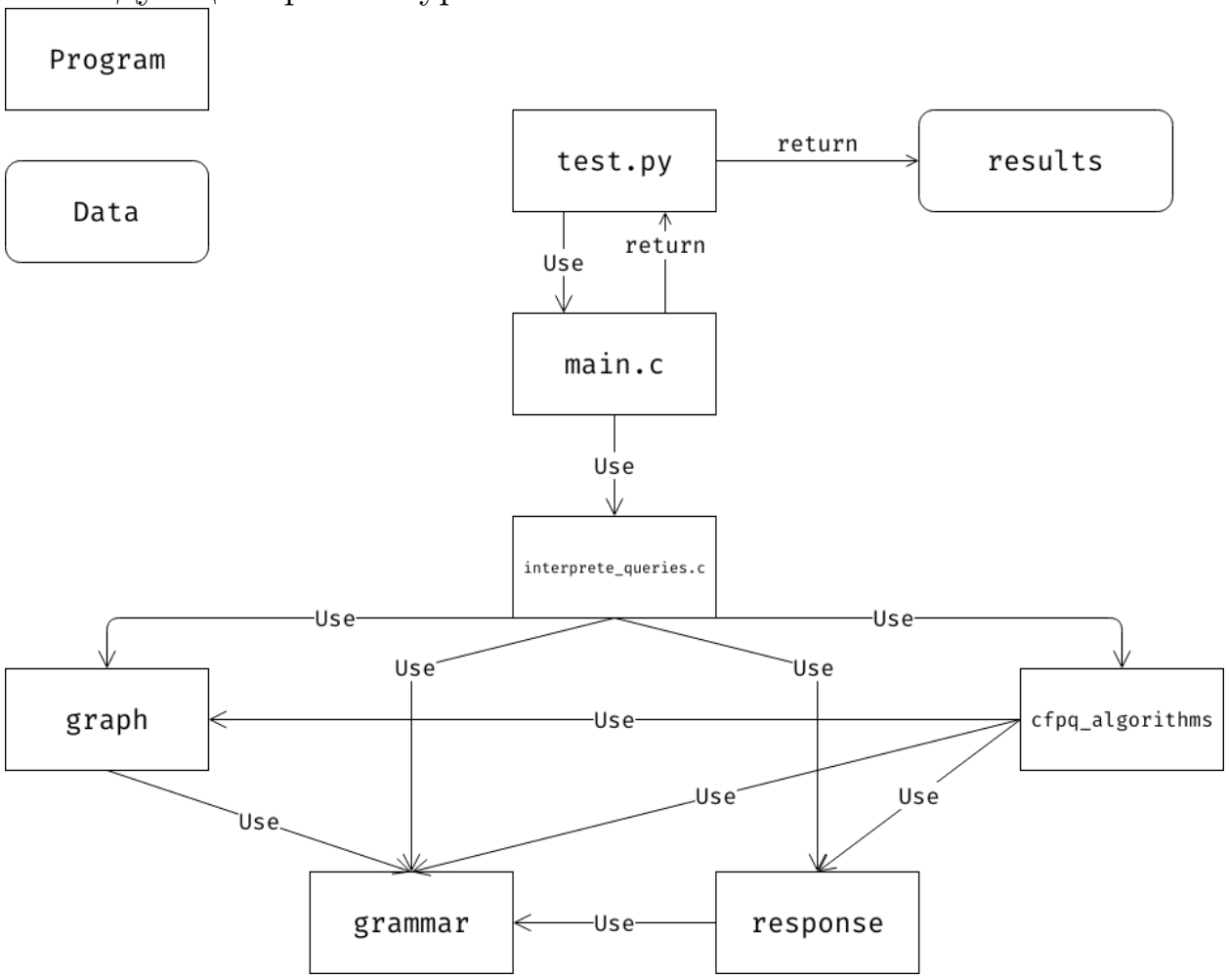
$$\forall i, j \in V(D') \ : \ A''_{i,j} = (A')^+_{i,j}.$$

. □

Следовательно обе операции обновления графа работают корректно и данный алгоритм представляет практический интерес.

5. Реализация

В качестве языка для реализации разработанного алгоритма был выбран C. Так как на этом языке написана библиотека $SuiteSparse$, которая предоставляет необходимую функциональность для работы с алгебраическими типами, такими как графы и контекстно-свободные грамматики, а так же позволяет выражать операции с этими типами в матричном виде. Реализация алгоритма была разделена на модули, каждый из которых отвечает за важную составляющую всего алгоритма, со следующей архитектурой.



Ключевой модуль всего решения это $cfpq\_algorithms$, в котором реализованы два алгоритма.

- Статичный алгоритм $cfpq\_static$ — алгоритм, описанный в статье [1].
- Динамический алгоритм — в виде функций $cfpq\_vertex\_added$

и $cfpq\_edge\_added$, которые отвечают за операции $AddVertex$ и $AddEdge$ соответственно.

Основные используемые модули для представления графа, грамматики и обработки запросов.

- Модуль $graph$ – отвечает за представление помеченного ориентированного графа в виде множества матриц, каждая из которых отвечает определенному терминалу грамматики.
- Модуль $grammar$ – отвечает за представление контекстно-свободной грамматики внутри программы.
- Модуль $response$ – отвечает за представление контекстно-свободных отношений в виде множества матриц, каждая из которых отвечает определенному нетерминалу грамматики.
- Модуль $interpreter$ – отвечает за обработку запросов на изменение графа.

А также вспомогательные модули:

- Модуль $mapper$ – отвечает за представление терминалов и нетерминалов с помощью индексов для унифицированного обращения внутри алгоритмов.
- Модуль $timer$ – отвечает за замеры времени с использованием функции $clock\_gettime$ из стандартной библиотеки $time.h$.
- Модуль $config$ – отвечает за конфигурацию решения, в том числе за максимальное поддерживаемое количество вершин и ребер графа, количество терминалов и нетерминалов и т.п.

В качестве языка для реализации экспериментального исследования был выбран $Python$. Так как на этом языке можно предельно ясно, но при этом и эффективно выразить все сценарии тестирования. Написана программа $test.py$, в которой последовательно запускаются следующие этапы экспериментального исследования.

- Осуществляется сборка реализованных алгоритмов командой $make$ с ключами $-O3 - march = native$ для получения данных близких к практике и унифицированного сравнение результатов. В файле $main.c$ последовательно загружаются сначала грамматика, потом граф и последовательность запросов.

- Осуществляется загрузка графов из репозитория $CFPQ\_Data$ для проведения экспериментальных замеров.

- По загруженным графам строятся последовательности запросов двух видов — запросы добавления ребра в граф с помощью статично алгоритма и с помощью разработанного динамического алгоритма.

- Производится по сто экспериментальных запусков реализованных алгоритмов на загруженных графах, вычисляется среднее время(с точностью в три знака после запятой) работы и результаты сохраняются в папку $results$.

6. Экспериментальное исследование

Для определения эффективности построенного алгоритма проведем экспериментальный анализ на сценарии построения графа "с нуля" с помощью операций добавления ребер. После каждой операции будем пересчитывать ответ на контекстно-свободный запрос. Такой сценарий отвечает инкрементальной сути задачи.

Для проведения экспериментального исследования были выбраны $WorstCase$ графы, как графы отражающие наихудший случай для статичного алгоритма, и RDF графы, для рассмотрения эффективности реализованного алгоритма на реальных данных.

Основной целью данных экспериментов является рассмотрение вопроса эффективности данного динамического алгоритма в сравнении с статичным аналогом и решение вопроса применимости на реальных данных.

В качестве измеримого результата выбрано среднее время на ста запусках исполнения сценария построения графа "с нуля".

Эксперименты проводились на ПК с операционной системой Ubuntu 18.04, процессором Intel core i7-6700 3.4GHz и оперативной памятью DDR4 64Gb.

6.1. Результаты эксперимента

В ходе проведенного экспериментального исследования доказана эффективность применения разработанного алгоритма на практике. На всех экспериментах предложенный алгоритм показал значительное превосходство во времени исполнения всех запросов. Так же, с ростом числа вершин в $WorstCase$ графе можно заметить и рост опережения разработанного алгоритма в сравнении со статичным аналогом. На RDF графах алгоритм работал не менее чем в два раза быстрее.

Таблица 1: Результаты сравнения на $WorstCase$ графах (время в секундах)

Граф	**Статичный алгоритм**	**Динамический алгоритм**
worstcase_4	<0.001	<0.001
worstcase_8	<0.001	<0.001
worstcase_16	0.002	0.002
worstcase_32	0.011	0.010
worstcase_64	0.062	0.060
worstcase_128	0.940	0.891

Таблица 2: Результаты сравнения на RDF графах (время в секундах)

Граф	**Статичный алгоритм**	**Динамический алгоритм**
skos	0.018	0.006
generations	0.027	0.007
travel	0.023	0.010
univ-bench	0.031	0.009
atom-primitive	0.040	0.023
biomedical-mesure-primitive	0.062	0.030
foaf	0.063	0.033
people_pets	0.068	0.025
funding	0.138	0.049
wine	0.192	0.088
pizza	0.234	0.113
core	0.337	0.145
pathways	13.005	10.325

По полученным результатам можно заметить, что динамический алгоритм заметно быстрее статичного как на $WorstCase$ графах, так и на RDF графах. Таким образом, можно сделать вывод о практической эффективности и применимости разработанного алгоритма.

7. Заключение

В данной работе были получены следующие результаты.

- Разработан инкрементальный алгоритм и доказана его корректность.
- Разработанный алгоритм реализован в виде программы на языке C.
- Проведено экспериментальное исследование разработанного алгоритма.

Кроме того, выделены несколько направлений будущих исследований.

- Эффективная реализация построенного алгоритма с использованием специализированных библиотек и параллельных вычислений. Поскольку в основе реализации алгоритма лежит применение матричной алгебры, это открывает большой простор для оптимизаций и параллельных вычислений. Так, применение GPGPU может сильно улучшить производительность алгоритма засчет ускорения матричных операций.
- Изучение возможных сценариев тестирования разработанного алгоритма. Представленный сценарий для экспериментального исследования хотя и репрезентативен с точки зрения инкрементальной сути задачи, но не является абсолютно точно моделирующим реальные ситуации работы с графом. Поэтому, имеет смысл изучать моделирование таких сценариев. Так, например, важно изучить тот порог, когда пересчет всего ответа сравним по времени с пересчетом всех скопившихся обновлений графа в сценарии, когда граф меняется без потребности пересчета ответа в этот же момент.

Список литературы

[1] Azimov Rustam, Grigorev Semyon. Context-free path querying by matrix multiplication. — 2018. — 06. — P. 1–10.

[2] Bouyer Patricia, Jugé Vincent. Dynamic Complexity of the Dyck Reachability. — 2016. — 10.

[3] Demetrescu Camil, Italiano Giuseppe. Dynamic shortest paths and transitive closure: Algorithmic techniques and data structures // Journal of Discrete Algorithms. — 2006. — 09. — Vol. 4. — P. 353–383.

[4] Mendelzon Alberto O., Wood Peter T. Finding Regular Simple Paths in Graph Databases // SIAM J. Comput. — 1995. — 12. — Vol. 24, no. 6. — P. 1235–1258. — URL: http://dx.doi.org/10.1137/S009753979122370X.

[5] Quantifying variances in comparative RNA secondary structure prediction / James Anderson, Adám Novák, Zsuzsanna Sükösd et al. // BMC bioinformatics. — 2013. — 05. — Vol. 14. — P. 149.

[6] Valiant Leslie G. General Context-free Recognition in Less Than Cubic Time // J. Comput. Syst. Sci. — 1975. — 04. — Vol. 10, no. 2. — P. 308–315. — URL: https://doi.org/10.1016/S0022-0000(75)80046-8.