Синтаксический анализ графов с использованием конъюнктивных грамматик

Р.Ш. Азимов <rustam.azimov19021995@gmail.com>
С.В. Григорьев <Semen.Grigorev@jetbrains.com>
Санкт-Петербургский государственный университет,
199034, Россия, г. Санкт-Петербург, Университетская наб., д. 7/9

Аннотация. Графы используются в качестве структуры данных для представления больших объемов информации в компактной и удобной для анализа форме в различных областях – биоинформатике, графовых базах данных, статическом анализе кода и др. При этом оказывается необходимо вычислять запросы к большим графам с целью выявления зависимостей между их вершинами. Ответом на такие запросы обычно является множество всех троек (A, *m*, *n*), для которых существует путь в графе от вершины *m* до вершины *n* такой, что метки на ребрах этого пути образуют строку, выводимую из нетерминала A в некоторой контекстно-свободной грамматике. Говорят, что такой тип запросов вычисляется с использованием реляционной семантики запросов. Кроме того, существуют конъюнктивные грамматики, образующие более широкий класс грамматик, чем традиционные контекстно-свободные грамматики. Использование конъюнктивных грамматик в задаче синтаксического анализа графов позволяет формулировать более сложные запросы к графу и решать более широкий круг задач. Известно, что задача вычисления запросов к графу с использованием реляционной семантики и конъюнктивных грамматик является неразрешимой. В данной работе предлагается алгоритм, вычисляющий приближенное решение этой задачи, а именно, аппроксимацию сверху. Предложенный алгоритм основан на матричных операциях, и проведенные эксперименты показывают, что его производительность может быть существенно повышена, благодаря использованию вычислений на графическом процессоре.

Ключевые слова: синтаксический анализ графов; конъюнктивные грамматики; транзитивное замыкание; матричные операции; вычисления на графическом процессоре.

DOI: 10.15514/ISPRAS-2018-30(2)-8

Для цитирования: Азимов Р.Ш., Григорьев С.В. Синтаксический анализ графов с использованием конъюнктивных грамматик. Труды ИСП РАН, том 30, вып. 2, 2018 г., стр. 149-166. DOI: 10.15514/ISPRAS-2018-30(2)-8

1. Введение

Графы используются во многих областях, например, в биоинформатике [1], в графовых базах данных [2], при статическом анализе программ [3], в навигационных сервисах для средств визуального моделирования [4]. При этом оказывается необходимым вычислять запросы к большим графам с целью выявления сложных зависимостей между их вершинами. Результатом вычисления таких запросов является множество неявных отношений между вершинами графа, то есть путей [5].

Естественно помечать ребра графа символами из некоторого конечного алфавита и выделять пути с помощью формальных грамматик над тем же алфавитом (регулярные выражения, контекстно-свободные грамматики). Говорят, что такой тип запросов вычислен с использованием реляционной семантики запросов. Наиболее популярными являются запросы, которые используют контекстно-свободные (КС) грамматики. Также существуют конъюнктивные грамматики [6], задающие более широкий класс языков, чем контекстно-свободные.

Использование конъюнктивных грамматик в задаче синтаксического анализа графов позволяет формулировать более сложные запросы к графам и решать более широкий круг задач, например, задачи поиска псевдонимов и уязвимостей в исходном коде [3]. Необходимо отметить, что задача вычисления запросов к графу с использованием реляционной семантики и конъюнктивных грамматик является неразрешимой [5]. Один из распространенных способов найти приближенное решение неразрешимой задачи – построить аппроксимацию решения.

В данной работе предложен алгоритм, вычисляющий приближенное решение задачи синтаксического анализа графов с использованием реляционной семантики запросов и конъюнктивных грамматик. Предложенный алгоритм основан на матричных операциях, что позволяет повысить производительность, используя для вычислений графический процессор.

Статья организована следующим образом. В разд. 2 приведены основные определения, связанные с задачей синтаксического анализа графов, и рассмотрены основные подходы данной области; в разд. 3 проанализированы существующие решения данной задачи; в разд. 4 представлен алгоритм синтаксического анализа графов, использующий реляционную семантику запросов и конъюнктивные грамматики, а также доказана его корректность; в разд. 5 работа предложенного алгоритма разобрана на небольшом примере; в разд. 6 представлены результаты проведенных экспериментов; заключение и направления будущих исследований приведены в разд. 7.

2. Обзор

В этом разделе мы определим задачу синтаксического анализа графов и обсудим основные подходы, применяемые для ее решения.

Пусть Σ – конечное множество терминальных символов. *Помеченным графом* будем называть пару $D = (V, E)$, где V является множеством вершин, а $E \subseteq V \times \Sigma \times V$ – множеством ребер с метками из алфавита Σ. Для пути π в графе D мы будем использовать обозначение $l(\pi)$, чтобы указать на слово, полученное конкатенацией меток на ребрах данного пути. Кроме того, запись $m\pi n$ будет обозначать, что в рассматриваемом графе существует путь из вершины $m \in V$ в вершину $n \in V$.

Результатом работы алгоритма синтаксического анализа графов с использованием формальной грамматики G обычно является множество всех троек (A, m, n), для которых путь $m\pi n$ таков, что строка $l(\pi)$ выводима из нетерминала A в грамматике G. Говорят, что такой тип запросов вычислен с использованием *реляционной семантики запросов* [5].

Традиционно в качестве грамматики G используются регулярные выражения [2, 7, 8, 9, 10]. Но в последнее время стало популярным использовать КС-грамматики [5, 11, 12, 13], так как некоторые полезные запросы не могут быть описаны с помощью регулярных грамматик. Примером таких запросов являются классические запросы поиска всех вершин в графе, находящихся на одном уровне иерархии [14]. Все рассмотренные алгоритмы синтаксического анализа графов принимают на вход КС-грамматики в *нормальной форме Хомского* [15].

Существует ряд алгоритмов синтаксического анализа графов с использованием реляционной семантики запросов и КС-грамматик [5, 12, 13], которые основаны на методе динамического программирования. Данные алгоритмы обобщают такие алгоритмы синтаксического анализа, как CYK [16, 17] и Earley [18]. В работе [5] для заданного графа $D = (V, E)$ и КС-грамматики $G = (N, \Sigma, P)$ для каждого $A \in N$ определяются *контекстно-свободные отношения* $R_A \subseteq V \times V$, являющиеся множеством пар вершин, между которыми существует путь, образующий строку, выводимую из нетерминала A. Такие отношения определяются следующим образом:

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

где $L(G_A)$ – это язык, порожденный грамматикой G со стартовым нетерминалом A.

В [5] представлен алгоритм, вычисляющий контекстно-свободные отношения R_A. Кроме того, существует алгоритм синтаксического анализа графов с использование реляционной семантики запросов и КС-грамматик, вычисляющий данные контекстно-свободные отношения R_A используя матричное транзитивное замыкание [11]. Данный алгоритм обобщает алгоритм Вэлианта [19] и сводится к умножению булевых матриц.

Также существуют *конъюнктивные грамматики* [6], образующие более широкий класс грамматик, чем контекстно-свободные. *Конъюнктивная грамматика* – это тройка $G = (N, \Sigma, P)$, где N – конечное множество

нетерминальных символов, Σ – конечное множество терминальных символов и P – конечное множество правил следующего вида:

- $A \to B_1 C_1 \& \dots \& B_m C_m$, для $m \geq 1, A, B_i, C_i \in N$,
- $A \to x$, для $A \in N, x \in \Sigma$.

Как и в случае КС-грамматик, мы рассматриваем только конъюнктивные грамматики в бинарной нормальной форме [20], так как для каждой конъюнктивной грамматики можно построить эквивалентную ей грамматику в данной форме. Кроме того, мы не выделяем стартовый нетерминал, так как его можно будет определить во время синтаксического анализа графа.

Мы будем писать $A \to^* w$, чтобы указать, что строка $w \in \Sigma^*$ может быть получена из нетерминала A некоторой последовательностью применений правил конъюнктивной грамматики. Отношение $\to$ определим следующим образом.

- При применении правила $(A \to B_1 C_1 \& \dots \& B_m C_m) \in P$ любой подтерм A любого терма может быть заменен подтермом $B_1 C_1 \& \dots \& B_m C_m$:
$$\dots A \dots \to \dots (B_1 C_1 \& \dots \& B_m C_m) \dots$$
- Конъюнкция нескольких одинаковых строк из Σ^* может быть перезаписана одной строкой, т.е. для любого $w \in \Sigma^*$
$$\dots (w \& \dots \& w) \dots \to \dots w \dots$$

Языком, задаваемым конъюнктивной грамматикой $G = (N, \Sigma, P)$ со стартовым нетерминалом $S \in N$, будем называть $L(G_S) = \{w \in \Sigma^* | S \to^* w\}$.

Определение контекстно-свободных отношений R_A естественным образом обобщается до определения конъюнктивных отношений R_A путем замены КС-грамматики на конъюнктивную. Таким образом, задача синтаксического анализа графов с использованием реляционной семантики запросов и конъюнктивных грамматик сводится к поиску конъюнктивных отношений R_A для всех нетерминалов A.

Но также известен факт, что данная задача является неразрешимой [5]. Поэтому единственный известный алгоритм для решения данной задачи, описанный в [3], дает приближенный результат. Данный алгоритм используется для статического анализа программ и аппроксимирует сверху конъюнктивные отношения R_A, то есть точное решение является подмножеством предложенного.

3. Существующие работы

Имеется ряд алгоритмов для синтаксического анализа графов с использованием реляционной семантики запросов и КС-грамматик [5, 12, 13], которые, однако, демонстрируют низкую производительность на больших графах, так как имеют в худшем случае кубическую временную сложность. Одной из самых популярных техник, используемых для увеличения производительности при работе с большими объемами данных, является использование графического процессора. В работе [11] представлен алгоритм синтаксического анализа

графов, использующий реляционную семантику запросов и КС-грамматики и вычисляющий матричное транзитивное замыкание с применением матричных операций. Кроме того, известно, что для вычислений матричных операции можно эффективно использовать графический процессор [21].

Однако все перечисленные алгоритмы работают только с КС-грамматиками и не позволяют обрабатывать запросы, описанные с помощью конъюнктивных грамматик. В работе [3] описан алгоритм, работающий с конъюнктивными грамматиками. Но данный алгоритм принимает на вход только определенный подкласс конъюнктивных грамматик, а именно, линейные конъюнктивные грамматики, которые имеют не более одного нетерминального символа в каждом конъюнкте правила.

Таким образом, задача построения алгоритма синтаксического анализа графов, использующего реляционную семантику запросов и работающего с произвольными конъюнктивными грамматиками, является открытой.

4. Алгоритм

В данном разделе мы представляем алгоритм синтаксического анализа графов, использующий реляционную семантику запросов и конъюнктивные грамматики. Предложенный алгоритм находит аппроксимацию сверху для конъюнктивных отношений R_A, вычисляя некоторое матричное транзитивное замыкание.

Пусть даны помеченный граф $D = (V, E)$ и конъюнктивная грамматика $G = (N, \Sigma, P)$. Определим *конъюнктивное матричное умножение* $a \circ b = c$, где a и b – матрицы подходящего размера, элементами которых являются подмножества множества нетерминалов N, следующим образом:

$$c_{i,j} = \{A \mid \exists (A \to B_1 C_1 \& \dots \& B_m C_m) \in P, \text{т.ч.} (B_k, C_k) \in d_{i,j}\},$$

где матрица d является результатом декартового произведения матриц a и b, т.е. $d_{i,j} = \bigcup_{k=1}^{n} a_{i,k} \times b_{k,j}$.

Кроме того, определим *конъюнктивное транзитивное замыкание* квадратной матрицы a как $a^{conj} = a^{(1)} \cup a^{(2)} \cup \dots$, где $a^{(i)} = a^{(i-1)} \cup (a^{(i-1)} \circ a^{(i-1)})$ для $i \geq 2$ и $a^{(1)} = a$.

Чтобы построить аппроксимацию сверху для конъюнктивных отношений R_A, мы построим матрицу T, используя ребра графа D и грамматику G. Для начала пронумеруем вершины графа D от 0 до (|V| - 1) и будем ассоциировать вершины с их номерами. Инициализируем каждый элемент |V|×|V| матрицы T пустым множеством $\emptyset$. Далее, для каждых i и j мы присваиваем в ячейки матрицы следующие значения:

$$T_{i,j} = \{A_k \mid \exists x(((i, x, j) \in E) \wedge ((A_k \to x) \in P))\}.$$

Наконец, мы вычисляем конъюнктивное транзитивное замыкание $T^{conj} = T^{(1)} \cup T^{(2)} \cup \dots$, где $T^{(i)} = T^{(i-1)} \cup (T^{(i-1)} \circ T^{(i-1)})$ для $i \geq 2$, а $T^{(1)}$ – это

построенная нами матрица T. Спецификация алгоритма представлена на листинге 1.

```
/* D - Входной помеченный граф
/* G – Входная конъюнктивная грамматика
/* T – Результат
1  n ← количество вершин графа D
2  E ← множество помеченных ребер графа D
3  P ← множество правил грамматики G
4  T ← матрица n × n с каждым элементом равным ∅
5  for each (i, x, j) ∈ E
6     T_{i,j} ← T_{i,j} ∪ {A| (A → x) ∈ P}
7  while матрица T изменяется do
8     T ← T ∪ (T ∘ T) /* вычисление транзитивного замыкания
```

Листинг 1. Алгоритм синтаксического анализа графов
Listing 1. A path querying algorithm

Для представленного алгоритма справедливы следующие утверждения.

Лемма 1. Пусть даны помеченный граф $D = (V, E)$ и конъюнктивная грамматика $G = (N, \Sigma, P)$ в нормальной форме. Рассмотрим вершины i, j такие, что $(i, j) \in R_A$ и существует путь $i\pi j$ такой, что для строки $l(\pi)$ существует дерево вывода грамматики $G = (N, \Sigma, P)$ с некоторым стартовым нетерминалом A с высотой $h \leq k$. Тогда $A \in T_{i,j}^{(k)}$.

Доказательство.

База индукции. Покажем, что утверждение леммы верно при $k = 1$. Для любых i, j и для любого нетерминала $A \in N$, если $(i, j) \in R_A$ и существует путь $i\pi j$ такой, что для строки $l(\pi)$ существует дерево вывода грамматики $G = (N, \Sigma, P)$ со стартовым нетерминалом A с высотой $h \leq 1$, то путь π состоит из единственного ребра, ведущего из вершины i в вершину j, и $(A \to x) \in P$, где $x = l(\pi)$. Таким образом, $A \in T_{i,j}^{(1)}$ и утверждение леммы при $k = 1$ доказано.

Индукционный переход. Предположим, что утверждение леммы верно для всех $k \leq (p - 1)$, где $p \geq 2$. Покажем, что утверждение леммы также верно при $k = p$.

Пусть существует путь $i\pi j$, образующий строку $l(\pi)$, для которой существует дерево вывода грамматики $G = (N, \Sigma, P)$ со стартовым нетерминалом A с высотой $h \leq p$. Если $h < p$, то утверждение леммы верно по индукционному предположению. Пусть $h = p$, и правило, использованное для корня этого дерева выглядит так: $(A \to B_1 C_1 \& \dots \& B_m C_m)$. Тогда все поддеревья для нетерминалов $(B_1, C_1, \dots B_m, C_m)$ имеют высоту меньше, чем p. Пусть каждая пара нетерминалов в конъюнктах $(B_1 C_1, \dots B_m C_m)$ разбивает в вершинах

$t_1, \dots, t_m$ путь $i\pi j$ на два подпути. По индукционному предположению получаем, что $B_x \in T_{i,t_x}^{(p-1)}$, а $C_x \in T_{t_x,j}^{(p-1)}$ для всех $1 \le x \le m$. Таким образом, все пары нетерминалов $(B_1, C_1), \dots (B_m, C_m)$ принадлежат $d_{i,j}$, где $d_{i,j} = \bigcup_{k=1}^{n} T_{i,k}^{(p-1)} \times T_{k,j}^{(p-1)}$. Значит, по определению конъюнктивного матричного умножения получаем, что $A \in T_{i,j}^{(p)}$, где $T^{(p)} = T^{(p-1)} \cup (T^{(p-1)} \circ T^{(p-1)})$. Таким образом, утверждение леммы также верно для $k = p$, ч. т. д.

Теорема 1. (*Корректность алгоритма*). Пусть даны помеченный граф $D = (V, E)$ и конъюнктивная грамматика $G = (N, \Sigma, P)$ в нормальной форме. Тогда для любых i, j и для любого нетерминала $A \in N$, если $(i, j) \in R_A$, то $A \in T_{i,j}^{conj}$.

Доказательство.

Пусть $(i, j) \in R_A$. По определению конъюнктивных отношений R_A получаем, что существует путь $i\pi j$ такой, что строка $l(\pi)$ принадлежит языку, порожденному грамматикой G со стартовым нетерминалом A. Рассмотрим некоторое дерево вывода этой грамматики для строки $l(\pi)$. Пусть высота этого дерева равна h. Тогда, по лемме 1 получаем, что $A \in T_{i,j}^{(h)}$. А так как $T^{conj} = T^{(1)} \cup T^{(2)} \cup \dots$, то $A \in T_{i,j}^{conj}$, ч. т. д.

Теорема 2. (*Конечность алгоритма*). Пусть даны помеченный граф $D = (V, E)$ и конъюнктивная грамматика $G = (N, \Sigma, P)$ в нормальной форме. Тогда конъюнктивное транзитивное замыкание матрицы T, построенной по графу D и грамматике G, может быть вычислено за конечное число операций.

Доказательство.

Если для некоторого $p \ge 2$, $T^{(p)} = T^{(p-1)}$, то для любого $k \ge p$, $T^{(k)} = T^{(p-1)}$ и $T^{conj} = T^{(p-1)}$. Таким образом, для нахождения матрицы T^{conj} достаточно вычислять матрицы $T^{(i)}$ до тех пор, пока не встретятся две подряд идущие одинаковые матрицы. По определению конъюнктивного транзитивного замыкания, в каждой ячейке матрицы $T^{(i+1)}$ количество нетерминалов больше либо равно, чем в соответствующей ячейке матрицы $T^{(i)}$. А так как общее количество нетерминалов в любой матрице $T^{(i)}$ не может быть больше, чем $|V|^2|N|$, то $T^{conj} = T^{|V|^2|N|}$. Таким образом, матрица T^{conj} может быть вычислена за конечное число операций, ч. т. д.

Таким образом, мы показали, как аппроксимация сверху всех конъюнктивных отношений R_A может быть найдена с помощью вычисления конъюнктивного транзитивного замыкания T^{conj}.

5. Пример работы алгоритма

В данном разделе мы продемонстрируем работу предложенного алгоритма на небольшом примере.

Пример основан на грамматике G со стартовым нетерминалом S и правилами, представленными на рис. 1.

$$S \to AB\&DC$$
$$A \to a$$
$$B \to BC$$
$$B \to b$$
$$C \to c$$
$$D \to AD$$
$$D \to b$$

Рис. 1. Пример входной конъюнктивной грамматики
Fig. 1. An example of the input conjunctive grammar

Конъюнкт AB порождает язык $L_{AB} = \{abc^*\}$, а конъюнкт DC порождает язык $L_{DC} = \{a^*bc\}$. Таким образом, конъюнктивная грамматика G со стартовым нетерминалом S порождает язык $L_S = L_{AB} \cap L_{DC} = \{abc\}$.

Рассмотрим работу предложенного алгоритма с входной грамматикой G и графом D, представленным на рис. 2.

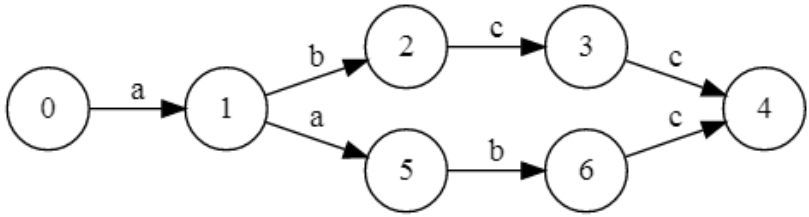

Рис. 2. Пример входного графа
Fig. 2. An example of the input graph

Сначала матрица T инициализируется в строках 5-6 листинга 1. Мы будем индексировать состояния матрицы T. Обозначим через T_0 состояние матрицы T, после инициализации. Матрица T_0 приведена на рис. 3.

$$\begin{pmatrix} \emptyset & \{A\} & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \{B,D\} & \emptyset & \emptyset & \{A\} & \emptyset \\ \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{B,D\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \end{pmatrix}$$

Рис. 3. Матрица T_0
Fig. 3. The matrix T_0

Далее в строках 7-8 листинга 1 вычисляются последующие состояния матрицы T до тех пор, пока не найдется такое $k \geq 1$, что $T_k = T_{k-1}$. Таким образом, матрица T_k будет является результатом работы алгоритма.

Для вычисления матрицы $T_1 = T_0 \cup (T_0 \circ T_0)$ необходимо вычислить матрицу d, являющуюся декартовым произведением матрицы T_0 с собой же. Матрица d представлена на рис. 4.

$$\begin{pmatrix} \emptyset & \emptyset & \{(A,B),(A,D)\} & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{(B,C),(D,C)\} & \emptyset & \emptyset & \{(A,B),(A,D)\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{(C,C)\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{(B,C),(D,C)\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

Рис. 4. Матрица d
Fig. 4. The matrix d

По матрице d строится матрица $T_1 = T_0 \cup (T_0 \circ T_0)$, представленная на рис. 5. Новые нетерминалы матрицы T_1, по сравнению с матрицей T_0, выделены жирным шрифтом.

$$\begin{pmatrix} \emptyset & \{A\} & \{\mathbf{D}\} & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \{B,D\} & \{\mathbf{B}\} & \emptyset & \{A\} & \{\mathbf{D}\} \\ \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{\mathbf{B}\} & \emptyset & \{B,D\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \end{pmatrix}$$

Рис. 5. Матрица $T_1 = T_0 \cup (T_0 \circ T_0)$
Fig. 5. The matrix $T_1 = T_0 \cup (T_0 \circ T_0)$

Далее вычисляются последующие состояния матрицы T до T_4 включительно, так как $T_4 = T_3$. Все последующие состояния матрицы T представлены на рис. 6. Новые нетерминалы матрицы T_i, по сравнению с матрицей T_{i-1}, выделены жирным шрифтом.

$$T_2 = \begin{pmatrix} \emptyset & \{A\} & \{D\} & \{\mathbf{S}\} & \emptyset & \emptyset & \{\mathbf{D}\} \\ \emptyset & \emptyset & \{B,D\} & \{B\} & \{\mathbf{S},\mathbf{B}\} & \{A\} & \{D\} \\ \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{B\} & \emptyset & \{B,D\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \end{pmatrix}$$

$$T_3 = \begin{pmatrix} \emptyset & \{A\} & \{D\} & \{S\} & \{\mathbf{S}\} & \emptyset & \{D\} \\ \emptyset & \emptyset & \{B,D\} & \{B\} & \{S,B\} & \{A\} & \{D\} \\ \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{B\} & \emptyset & \{B,D\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \end{pmatrix}$$

$$T_4 = \begin{pmatrix} \emptyset & \{A\} & \{D\} & \{S\} & \{S\} & \emptyset & \{D\} \\ \emptyset & \emptyset & \{B,D\} & \{B\} & \{S,B\} & \{A\} & \{D\} \\ \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{B\} & \emptyset & \{B,D\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \end{pmatrix}$$

Рис. 6. Последующие состояния матрицы T
Fig. 6. Remaining states of the matrix T

Таким образом, результатом работы предложенного алгоритма является матрица T_4. Далее мы можем сконструировать множества R'_A, которые являются аппроксимацией сверху конъюнктивных отношений R_A для всех нетерминалов $A \in N$, т.е. $R'_A \supseteq R_A$. Например, так как в ячейке (0,3) результирующей матрицы T_4 имеется нетерминал S, то $(0,3) \in R'_S$. Сконструированные множества R'_A для всех нетерминалов $A \in N$ представлены на рис. 7.

$$\begin{aligned} R'_S &= \{(0,3),(0,4),(1,4)\} \\ R'_A &= \{(0,1),(1,5)\} \\ R'_B &= \{(1,2),(1,3),(1,4),(5,4),(5,6)\} \\ R'_C &= \{(2,3),(3,4),(6,4)\} \\ R'_D &= \{(0,2),(0,6),(1,2),(1,6),(5,6)\} \end{aligned}$$

Рис. 7. Аппроксимация конъюнктивных отношений
Fig. 7. An approximation of the conjunctive relations

Данный пример демонстрирует, что не всегда удается получить точное решение. Например, пара вершин (0,4) принадлежит R'_S, хотя не существует пути из вершины 0 в вершину 4, образующего строку, выводимую из нетерминала S (таковой является единственная строка *abc*). Лишние пары вершин добавляются в том случае, если существуют различные пути из вершины *i* в вершину *j*, которые суммарно соответствуют конъюнктам одного правила, но не существует пути из вершины *i* в вершину *j*, который одновременно соответствовал бы всем конъюнктам этого правила. Например, для конъюнктов правила $S \to AB\&DC$ существует путь из вершины 0 в

вершину 4, образующий строку *abcc*, и существует также путь из вершины 0 в вершину 4, образующий строку *aabc*. Первый путь соответствует конъюнкту *AB*, так как строка *abcc* принадлежит языку $L_{AB} = \{abc^*\}$, а второй путь соответствует конъюнкту *DC*, так как строка *aabc* принадлежит языку $L_{DC} = \{a^*bc\}$. Однако, очевидно, что не существует пути из вершины 0 в вершину 4, образующего строку *abc*.

6. Эксперименты

В этом разделе мы приводим результаты экспериментов, целью которых является демонстрация практической применимости предложенного алгоритма и возможности его значительного ускорения при использовании для вычислений графического процессора. Для достижения данной цели мы реализовали алгоритм двумя способами: вычисляя матричные операции на ЦПУ и вычисляя их на графическом процессоре. Мы запустили данные реализации на классических конъюнктивных грамматиках [6] и случайно сгенерированных графах.

6.1 Постановка экспериментов

Эксперименты проводились на рабочей станции со следующими характеристиками: операционная система – Microsoft Windows 10 Pro x64-based PC, ЦПУ – Intel i7-4790, 3601 Mhz, 4 Core(s), 4 Logical Processor(s), оперативная память – 16 GB, графический процессор – NVIDIA GeForce GTX 1070 (CUDA Cores: 1920, Core clock: 1556 MHz, Memory data rate:8008 MHz, Memory interface: 256-bit, Memory bandwidth: 256.26 GB/s, Dedicated video memory: 8192 MB GDDR5).

Было выполнено две реализации алгоритма на языке программирования F# [22].

- Реализация *onCPU* использует ЦПУ для вычисления матричных операций. Для хранения матриц в разреженном формате и вычисления матричных операций использована библиотека Math.Net Numerics [23].
- Реализация *onGPU* использует графический процессор для вычисления матричных операций. Для вычисления матричных операций на графическом процессоре использована обёртка CUSPARSE-библиотеки, взятую из библиотеки managedCuda [24].

Сравнение производительности двух выполненных реализаций позволит определить эффективность ускорения предложенного алгоритма, с использованием вычислений матричных операций на графическом процессоре.

Для экспериментов мы использовали два запроса к случайно сгенерированным графам, соответствующим двум классическим конъюнктивным грамматикам.

Запрос 1. Данный запрос основан на конъюнктивной грамматике G со стартовым нетерминалом S, которая порождает язык $\{a^n b^n c^n | n > 0\}$. Правила данной грамматики представлены на рис. 8.

$$S \to AB\&DC$$
$$A \to AA \mid a$$
$$B \to bBc \mid bc$$
$$C \to CC \mid c$$
$$D \to aDb \mid ab$$

Рис. 8. Конъюнктивная грамматика, порождающая язык $\{a^n b^n c^n | n > 0\}$
Fig. 8. An example of the input conjunctive grammar for language $\{a^n b^n c^n | n > 0\}$

Представленная грамматика приводится в бинарную нормальную форму и подается на вход алгоритма вместе со случайно сгенерированными графами, метки на ребрах которых принадлежат алфавиту $\Sigma = \{a, b, c\}$. Предложенный алгоритм выделяет пары вершин в найденной аппроксимации конъюнктивного отношения R'_S, как потенциальные, между которыми существует путь, формирующий строку из языка $\{a^n b^n c^n | n > 0\}$.

Запрос 2. Данный запрос основан на конъюнктивной грамматике G со стартовым нетерминалом S, которая порождает язык $\{wcw \mid w \in \{a, b\}^*\}$. Правила данной грамматики представлены на рис. 9.

$$S \to C\&D$$
$$C \to aCa \mid aCb \mid bCa \mid bCb \mid c$$
$$D \to aA\&aD \mid bB\&bD \mid cE$$
$$A \to aAa \mid aAb \mid bAa \mid bAb \mid cEa$$
$$B \to aBa \mid aBb \mid bBa \mid bBb \mid cEb$$
$$E \to aE \mid bE \mid \epsilon$$

Рис. 9. Конъюнктивная грамматика, порождающая язык $\{wcw \mid w \in \{a, b\}^*\}$
Fig. 9. An example of the input conjunctive grammar for language $\{wcw \mid w \in \{a, b\}^*\}$

Представленная грамматика также приводится в бинарную нормальную форму и генерируются графы с метками на ребрах из алфавита $\Sigma = \{a, b, c\}$. Предложенный алгоритм выделяет пары вершин в найденной аппроксимации конъюнктивного отношения R'_S, как потенциальные, между которыми существует путь, формирующий строку из языка $\{wcw \mid w \in \{a, b\}^*\}$.

6.2 Результаты

Результаты вычислений запроса 1 представлены в табл. 1, где $|V|$ является количеством вершин сгенерированного графа, $|E|$ – количеством ребер, *#results* – это количество пар вершин в найденной аппроксимации

конъюнктивного отношения R'_S. Для двух выполненных реализаций в соответствующих столбцах представлено время работы алгоритма в миллисекундах.

Табл. 1. Результаты вычислений запроса 1
Table 1. Evaluation results for query 1

|V|	**|E|**	**#results**	**onCPU (ms)**	**onGPU (ms)**
100	*25*	*0*	*2*	*7*
100	*75*	*0*	*10*	*20*
100	*200*	*79*	*101*	*213*
1000	*250*	*1*	*265*	*25*
1000	*750*	*13*	*2781*	*102*
1000	*2000*	*731*	*12050*	*347*
10000	*2500*	*4*	*26595*	*41*
10000	*7500*	*136*	*241087*	*213*
10000	*20000*	*4388*	*1305177*	*1316*

Результаты вычислений запроса 2 представлены в табл. 2. Для двух предложенных реализаций в соответствующих столбцах представлено время работы алгоритма в миллисекундах. Время работы реализации *onCPU* для входного графа с 10000 вершин и 10000 ребер опущено из-за слишком низкой производительности данной реализации на больших графах.

6.3 Анализ результатов

Результаты вычислений запроса 1 показывают, что реализация, вычисляющая матричные операции на ЦПУ, более производительная только на нескольких небольших графах. Это связано с временными затратами на выделение памяти и обмен данными с графическим процессором. Кроме того, прирост производительности при использовании графического процессора для вычисления матричных операций значительно увеличивается с ростом размера входного графа.

Табл. 2. Результаты вычислений запроса 2
Table 2. Evaluation results for query 2

|V|	**|E|**	**#results**	**onCPU (ms)**	**onGPU (ms)**
100	*25*	*9*	*14*	*67*

100	*75*	*29*	*114*	*129*
100	*100*	*47*	*254*	*483*
1000	*250*	*82*	*2566*	*127*
1000	*750*	*279*	*21394*	*530*
1000	*1000*	*438*	*64725*	*1951*
10000	*2500*	*829*	*268843*	*257*
10000	*7500*	*2796*	*3380046*	*1675*
10000	*10000*	*27668*	*–*	*3017*

Аналогично, результаты вычислений запроса 2 показывают, что реализация *onCPU* более производительная по сравнению с реализацией *onGPU* только на нескольких небольших графах и прирост производительности при использовании графического процессора также значительно увеличивается с ростом размера входного графа. Кроме того, сравнение результатов вычислений запросов 1 и 2 показывает, что вычисление запроса 2 требует значительно большего времени, чем вычисление запроса 1 при аналогичных входных графах. Одной из причин этого является наличие большего количества правил в грамматике, приведенной в нормальную форму, для запроса 2, чем в аналогичной грамматике для запроса 1.

Таким образом, поставленные эксперименты демонстрируют практическую применимость предложенного алгоритма и показывают наличие возможности его значительного ускорения благодаря использованию вычислений на графическом процессоре.

7. Заключение

В данной работе был предложен алгоритм, вычисляющий приближенное решение задачи синтаксического анализа графов с использованием реляционной семантики запросов и конъюнктивных грамматик. Кроме того, были проведены эксперименты, демонстрирующие практическую применимость предложенного алгоритма и возможность повысить его производительность, используя для вычислений графический процессор.

Кроме того, мы можем определить несколько открытых проблем для будущих исследований. В данной работе была рассмотрена только одна семантика запросов – реляционная. Но существуют другие семантики запросов [25], в которых необходим предоставлять один или даже все пути, образующие строки, выводимые из нетерминалов входной грамматики. Может ли предложенный алгоритм быть обобщен до данных семантик запросов?

Также, существуют Булевы грамматики [26], которые принадлежат более широкому и выразительному классу грамматик, чем конъюнктивные.

Существует также матричный алгоритм [27] синтаксического анализа строк, использующий Булевы грамматики. Может ли наш алгоритм быть обобщен до алгоритма синтаксического анализа графов использующего реляционную семантику запросов и Булевы грамматики?

Список литературы

[1]. Anderson J. W. et al. Quantifying variances in comparative RNA secondary structure prediction. BMC bioinformatics, vol. 14, № 1, 2013.
[2]. Mendelzon A., Wood P. Finding Regular Simple Paths in Graph Databases. SIAM J. Computing, vol. 24, № 6, 1995, pp. 1235–1258.
[3]. Zhang Q., Su Z. Context-sensitive data-dependence analysis via linear conjunctive language reachability. Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, 2017, pp. 344–358.
[4]. Кознов Д.В., Ларчик Е.В., Терехов А.Н. Трансформация динамических представлений в предметно-ориентированном визуальном моделировании. Программирование, том 41, № 4, 2015 г., стр. 3-12
[5]. Hellings. J. Conjunctive context-free path queries. In: Proc. of ICDT'14, 2014, pp.119–130.
[6]. Okhotin A. Conjunctive grammars. Journal of Automata, Languages and Combinatorics, vol. 6, № 4, 2001, pp. 519–535.
[7]. Abiteboul S., Vianu V. Regular path queries with constraints. In Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, 1997 pp. 122–133.
[8]. Fan W., Li J., Ma S., Tang N, and Wu Y. 2011. Adding regular expressions to graph reachability and pattern queries. In 27th Data Engineering International Conference, 2011, pp. 39–50.
[9]. Nolé M. and Sartiani C. Regular path queries on massive graphs. In Proceedings of the 28th International Conference on Scientific and Statistical Database Management, 2016.
[10]. Reutter J., Romero M., and Vardi M. Regular queries on graph databases. Theory of Computing Systems, vol. 61, № 1, 2017, pp. 31–83
[11]. Azimov R. Sh., Grigorev S. V. Context-Free Path Querying by Matrix Multiplication. arXiv preprint, arXiv:1707.01007v2, 2017.
[12]. Sevon P., Eronen L. Subgraph queries by context-free grammars. Journal of Integrative Bioinformatics, vol. 5, № 2, 2008.
[13]. Zhang X. et al. Context-free path queries on RDF graphs. International Semantic Web Conference, 2016, pp. 632–648.
[14]. Abiteboul S., Hull R., Vianu V. Foundations of databases: the logical level. Addison-Wesley Longman Publishing Co., Inc., 1995.
[15]. Chomsky N. On certain formal properties of grammars. Information and control, vol. 2, № 2, 1959, pp. 137–167.
[16]. Kasami T. An Efficient Recognition and Syntax-Analysis Algorithm for Context-Free Languages. Report of University of Hawaii, Contract No. AF 19(628)-4379, No. 2, July, 1965.
[17]. Younger D.H. Recognition and parsing of context-free languages in time n3, Information and control, vol. 10, № 2, 1967, pp. 189–208.
[18]. Grune D., Jacobs C. J. H. Parsing Techniques (Monographs in Computer Science). Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[19]. Valiant L.G. General context-free recognition in less than cubic time. Journal of computer and system sciences, vol., №2, pp. 308– 315.
[20]. Okhotin A. Conjunctive and Boolean grammars: the true general case of the context-free grammars. Computer Science Review, vol. 9, 2013. pp. 27–59.
[21]. Che S., Beckmann B.M., Reinhardt S.K. Programming GPGPU Graph Applications with Linear Algebra Building Blocks. International Journal of Parallel Programming, vol. 45, Issue 3, June 2017, pp 657–679.
[22]. Syme D., Granicz A., and Cisternino A. Expert F# 3.0. Springer, 2012.
[23]. The Math.Net Numerics WebSite. Доступно по ссылке: https://numerics.mathdotnet.com/, 20.03.2018.
[24]. The managedCuda library. Доступно по ссылке: https://kunzmi.github.io/managedCuda/, 20.03.2018.
[25]. Hellings J. Querying for Paths in Graphs using Context-Free Path Queries. arXiv preprint arXiv:1502.02242, 2015.
[26]. Okhotin A. 2004. Boolean grammars. Information and Computation, vol. 194, issue 1, 2004, pp. 19–48.
[27]. Okhotin A. Parsing by matrix multiplication generalized to Boolean grammars. Theoretical Computer Science, vol. 516, 2014, pp. 101–120.

Path querying using conjunctive grammars

R.Sh. Azimov <rustam.azimov19021995@gmail.com>
S.V. Grigorev <Semen.Grigorev@jetbrains.com>
Saint Petersburg State University,
7/9, Universitetskaya nab., St. Petersburg, 199034, Russia

Abstract. Graphs are used as a data structure to represent large volumes of information in a compact and convenient for analysis form in many areas: bioinformatics, graph databases, static code analysis, etc. In these areas, it is necessary to evaluate a queries for large graphs in order to determine the dependencies between the nodes. The answer to such queries is usually a set of all triples (A, m, n) for which there is a path in the graph from the vertex m to the vertex n, such that the labels on the edges of this path form a string derivable from the nonterminal A in some context-free grammar. This type of query is calculated using relational query semantics and context-free grammars but there are conjunctive grammars, which form a broader class of grammars than traditional context-free grammars. The use of conjunctive grammars in the path querying allows us to formulate more complex queries to the graph and solve a wider class of problems. It is known that the path querying using relational query semantics and conjunctive grammars is undecidable problem. In this paper, we propose a path querying algorithm which uses relational query semantics and conjunctive grammars, and approximates the exact solution. The proposed algorithm is based on matrix operations, and our evaluation shows that it is possible to significantly improve the performance of the algorithm by using a graphics processing unit.

Keywords: path querying; conjunctive grammars; transitive closure; matrix operations; GPGPU.

DOI: 10.15514/ISPRAS-2018-30(2)-8

For citation: Azimov R.Sh., Grigorev S.V. Path querying using conjunctive grammars. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue. 2, 2018, pp. 149-166 (in Russian). DOI: 10.15514/ISPRAS-2018-30(2)-8

References

[1]. Anderson J. W. et al. Quantifying variances in comparative RNA secondary structure prediction. BMC bioinformatics, vol. 14, № 1, 2013.
[2]. Mendelzon A., Wood P. Finding Regular Simple Paths in Graph Databases. SIAM J. Computing, vol. 24, № 6, 1995, pp. 1235–1258.
[3]. Zhang Q., Su Z. Context-sensitive data-dependence analysis via linear conjunctive language reachability. Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, 2017, pp. 344–358.
[4]. Koznov D.V., Larchik E.V., Terekhov A.N. View to view transformations in domain specific modeling. Programming and Computer Software, vol. 41, № 4, 2015, pp. 208-214. DOI: 10.1134/S0361768815040039.
[5]. Hellings. J. Conjunctive context-free path queries. In: Proc. of ICDT'14, 2014, pp.119–130.
[6]. Okhotin A. Conjunctive grammars. Journal of Automata, Languages and Combinatorics, vol. 6, № 4, 2001, pp. 519–535.
[7]. Abiteboul S., Vianu V. Regular path queries with constraints. In Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, 1997 pp. 122–133.
[8]. Fan W., Li J., Ma S., Tang N, and Wu Y. 2011. Adding regular expressions to graph reachability and pattern queries. In 27th Data Engineering International Conference, 2011, pp. 39–50.
[9]. Nolé M. and Sartiani C. Regular path queries on massive graphs. In Proceedings of the 28th International Conference on Scientific and Statistical Database Management, 2016.
[10]. Reutter J., Romero M., and Vardi M. Regular queries on graph databases. Theory of Computing Systems, vol. 61, № 1, 2017, pp. 31–83
[11]. Azimov R. Sh., Grigorev S. V. Context-Free Path Querying by Matrix Multiplication. arXiv preprint, arXiv:1707.01007v2, 2017.
[12]. Sevon P., Eronen L. Subgraph queries by context-free grammars. Journal of Integrative Bioinformatics, vol. 5, № 2, 2008.
[13]. Zhang X. et al. Context-free path queries on RDF graphs. International Semantic Web Conference, 2016, pp. 632–648.
[14]. Abiteboul S., Hull R., Vianu V. Foundations of databases: the logical level. Addison-Wesley Longman Publishing Co., Inc., 1995.
[15]. Chomsky N. On certain formal properties of grammars. Information and control, vol. 2, № 2, 1959, pp. 137–167.
[16]. Kasami T. An Efficient Recognition and Syntax-Analysis Algorithm for Context-Free Languages. Report of University of Hawaii, Contract No. AF 19(628)-4379, No. 2, July, 1965.
[17]. Younger D.H. Recognition and parsing of context-free languages in time n3, Information and control, vol. 10, № 2, 1967, pp. 189–208.
[18]. Grune D., Jacobs C. J. H. Parsing Techniques (Monographs in Computer Science). Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[19]. Valiant L.G. General context-free recognition in less than cubic time. Journal of computer and system sciences, vol., №2, pp. 308– 315.
[20]. Okhotin A. Conjunctive and Boolean grammars: the true general case of the context-free grammars. Computer Science Review, vol. 9, 2013. pp. 27–59.
[21]. Che S., Beckmann B.M., Reinhardt S.K. Programming GPGPU Graph Applications with Linear Algebra Building Blocks. International Journal of Parallel Programming, vol. 45, Issue 3, June 2017, pp 657–679.
[22]. Syme D., Granicz A., and Cisternino A. Expert F# 3.0. Springer, 2012.
[23]. The Math.Net Numerics WebSite. Доступно по ссылке: https://numerics.mathdotnet.com/, 20.03.2018.
[24]. The managedCuda library. Доступно по ссылке: https://kunzmi.github.io/managedCuda/, 20.03.2018.
[25]. Hellings J. Querying for Paths in Graphs using Context-Free Path Queries. arXiv preprint arXiv:1502.02242, 2015.
[26]. Okhotin A. 2004. Boolean grammars. Information and Computation, vol. 194, issue 1, 2004, pp. 19–48.
[27]. Okhotin A. Parsing by matrix multiplication generalized to Boolean grammars. Theoretical Computer Science, vol. 516, 2014, pp. 101–120.