

Задача поиска путей в ациклических графах с ограничениями в терминах булевых грамматик

¹ С.В. Григорьев <s.v.grigoriev@spbu.ru>

² Е.Н. Шеметова <katyacyfra@gmail.com>

¹ Санкт-Петербургский государственный университет,
199034, Россия, г. Санкт-Петербург, Университетская наб., д. 7/9.

² Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики (Университет ИТМО),
197101, Россия, г. Санкт-Петербург, Кронверкский пр., 49

Аннотация. Графовая модель данных используется в таких областях как графовые базы данных, биоинформатика, анализ социальных сетей, статический анализ кода. Одной из задач анализа графовых данных является задача поиска специфичных путей. Естественным способом задать ограничения на пути является использование формальной грамматики над метками рёбер графа. Запрос к графу называется запросом с использованием реляционной семантики, если ответом на него является множество всех троек (A, v_1, v_2) , для которых существует путь в графе от вершины v_1 до вершины v_2 такой, что метки на ребрах этого пути образуют строку, выводимую из нетерминала A в данной грамматике. Использование булевых грамматик в качестве формальной грамматики над метками рёбер позволяет формулировать более сложные запросы по сравнению с традиционно используемыми регулярными и контекстно-свободными грамматиками. Известно, что задача вычисления запросов к графу с использованием реляционной семантики и булевых грамматик является неразрешимой. В данной работе предложен приближённый алгоритм поиска путей в ориентированных графах без циклов с ограничениями в виде булевых грамматик и с использованием реляционной семантики запросов. Благодаря ограничению на тип анализируемых графов, предложенный алгоритм является более асимптотически оптимальным, чем наивный итерационный алгоритм.

Ключевые слова: поиск путей с ограничениями; булевы грамматики; матричные операции; ациклический граф; булевы матрицы; произведение матриц.

DOI: [10.15514/ISPRAS-2016-1\(2\)-33](https://doi.org/10.15514/ISPRAS-2016-1(2)-33)

Для цитирования: Григорьев С.В., Шеметова Е.Н. Задача поиска путей с ограничениями в терминах булевых грамматик на ациклических графах. Труды ИСП РАН, 2019, том ? вып. ?, с. ?-?. DOI: [10.15514/ISPRAS-2016-1\(2\)-33](https://doi.org/10.15514/ISPRAS-2016-1(2)-33)

1. Введение

Графовая модель данных широко используется в таких областях как графовые базы данных [1, 2], биоинформатика [3], моделирование и анализ социальных сетей [4, 5], статический анализ кода [6, 7, 8].

Одной из задач анализа данных, представленных в виде графа, является задача поиска специфичных путей. При этом существуют различные формальные средства задания ограничений на пути. Одним из достаточно естественных способов достижения этой цели является использование формальных языков: если рёбра графа содержат метки, то путь задаёт слово, которое получается конкатенацией меток вдоль него. Таким образом, в качестве критерия для поиска можно использовать факт принадлежности полученного слова заданному языку. В результате, возникает задача поиска путей с ограничениями, заданными в терминах формальных языков.

Многие современные языки запросов к графам, такие как Cypher [9], Gremlin [10], SPARQL [11], предоставляют средства задания ограничений в терминах регулярных языков или регулярных выражений. Однако регулярных ограничений не достаточно для решения многих задач статического анализа кода, анализа иерархических зависимостей и поиска подобных элементов.

В результате возникает необходимость использовать более выразительные классы языков для задания ограничений: контекстно-свободные, конъюнктивные [12], булевы [13].

Вопрос использования контекстно-свободных грамматик в качестве ограничений активно исследуется в настоящее время [14, 15, 16], предложены эффективные алгоритмы выполнения соответствующих запросов к графам [17, 18, 19]. При использовании более выразительных, чем контекстно-свободные, языков, возникает вопрос о разрешимости соответствующей задачи. Показано, что задача поиска путей в произвольных ориентированных графах с ограничениями в виде конъюнктивных и булевых языков неразрешима [15], однако для конъюнктивных языков предложен алгоритм, строящий приближение ответа сверху [20], что делает его применимым для приближённого решения прикладных задач. При этом, предложенный алгоритм является наивным итеративным алгоритмом. Возможность построения алгоритма, строящего приближённое решение для задачи поиска путей с ограничениями в виде булевых языков, не исследована.

Также отметим, что представленные выше алгоритмы работают с графами произвольной структуры. Если же исходный граф является ациклическим, то можно воспользоваться известными свойствами ациклических графов, чтобы

предоставить более производительный алгоритм для решения задачи поиска путей с ограничениями для данной структуры графа. Пример подобного подхода рассмотрен в работе [8].

В данной работе предложен приближённый алгоритм поиска путей в ориентированных графах без циклов (Direct Acyclic Graph, DAG) с ограничениями в виде булевых грамматик. При этом алгоритм реализует так называемую реляционную семантику [14]: результатом работы алгоритма является отношение на вершинах графа и нетерминалах. Элемент отношения — тройка (A, v_1, v_2) означает, что в заданном графе существует путь из вершины v_1 в вершину v_2 , такой, что соответствующее ему слово выводимо из нетерминала A в заданной грамматике. Доказано, что данный алгоритм строит приближение сверху. Благодаря ограничению на тип анализируемых графов, удалось построить более асимптотически оптимальный, чем наивный итерационный, алгоритм, использующий идеи Валианта [21] и Охотина [22].

Работа организована следующим образом.

В разделе 2 даны основные определения, связанные с задачей поиска путей с ограничениями в терминах булевых грамматик; в разделе 3 представлена адаптация алгоритма Охотина [22], находящая аппроксимацию решения задачи поиска путей с ограничениями в терминах булевых грамматик с использованием реляционной семантики запросов для ациклических графов, также доказана корректность применения данного алгоритма для поставленной задачи; в разделе 4 работа предложенного алгоритма продемонстрирована на примере; заключение и направления будущих исследований приведены в разделе 5.

2. Основные определения

Для начала определим задачу поиска путей с ограничениями в терминах булевых грамматик с использованием реляционной семантики запросов.

Рассмотрим ориентированный ациклический граф $D = (V, E)$ и формальную грамматику G . Пусть у каждого ребра графа есть метка, множество всех меток обозначим Σ . Тогда каждый путь в D будет обозначать слово над алфавитом из Σ , полученное конкатенацией меток рёбер, включенных в этот путь. На рисунке 1 изображен помеченный ациклический ориентированный граф с $\Sigma = \{a, b, c\}$. Для графа D и формальной грамматики $G = (\Sigma, N, P)$ мы обозначим отношения $R_A \subseteq V \times V$ для каждого $A \in N$ следующим образом:

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

где π_{nt} — путь из вершины n в t , $l(\pi)$ — слово, полученное конкатенацией меток рёбер, принадлежащих пути π , а $L(G_A)$ — язык, порожденный грамматикой G со стартовым нетерминалом A .

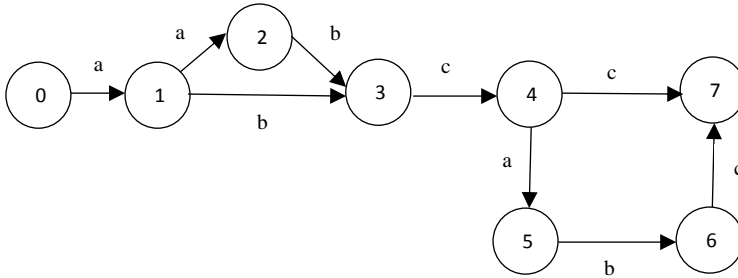


Рис. 1. Пример помеченного ориентированного ациклического графа с $\Sigma = \{a, b, c\}$.

Fig. 1. Example of directed labelled acyclic graph with $\Sigma = \{a, b, c\}$.

Таким образом, задача поиска путей в графе D с ограничениями в терминах формальной грамматики G и с использованием реляционной семантики запросов сводится к нахождению всех троек (A, n, t) , таких, что существует путь π_{nt} , такой, что строка $l(\pi)$ выводима из нетерминала A в грамматике G , то есть вычислению всех отношений R_A для любого $A \in N$.

В качестве формальной грамматики G будут использованы булевы грамматики [13]. Булева грамматика является классом формальных грамматик, расширяющим класс контекстно-свободных грамматик с помощью логических операций конъюнкции и отрицания.

Булева грамматика формально определена следующим образом:

$G = (\Sigma, N, P)$, где G — булева грамматика, Σ — терминальный алфавит, N — нетерминальный алфавит (множество нетерминальных символов $\{A_1, A_2, \dots, A_n\}$), P — множество правил грамматики.

Тогда правила булевой грамматики можно представить в виде:

$$A \rightarrow \alpha_1 \& \alpha_2 \& \dots \& \alpha_m \& \neg \beta_1 \& \neg \beta_2 \& \dots \& \neg \beta_n,$$

где A — нетерминал, $m + n \geq 1$, $\alpha_1, \alpha_2, \dots, \alpha_m, \beta_1, \beta_2, \dots, \beta_n \in (\Sigma \cup N)^*$.

Отметим, что в данном определении булевой грамматики не выделен стартовый нетерминал, так как его можно будет определить во время поиска путей с ограничениями. На рисунке 2 показана булева грамматика для языка $L(G_S) = \{a^k b c \mid k \neq 1\}$. В данном случае конъюнкт DC порождает язык $L(G_{DC}) = \{a^k b^i c^j \mid i = j = 1, k \geq 0\}$, а конъюнкт AB порождает язык $L(G_{AB}) =$

$\{a^k b^i c^j \mid k = i = 1, j \geq 0\}$. Тогда $L(G_S) = L(G_{DC}) \cap \overline{L(G_{AB})} = \{a^k b^i c^j \mid i = j = 1 \text{ и } k \neq i \neq 1\} = \{a^k bc \mid k \neq 1\}$.

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

Рис. 2. Пример булевой грамматики.

Fig. 2. Example of Boolean grammar.

Булева грамматика $G = (\Sigma, N, P)$ находится в двоичной нормальной форме, если каждое правило в P находится в одной из следующих форм:

- $A \rightarrow B_1 C_1 \& \dots \& B_m C_m \& \neg D_1 E_1 \& \dots \& \neg D_n E_n$ ($m \geq 1, n \geq 0$)
- $A \rightarrow a$ ($a \in \Sigma$)

Например, грамматика на рисунке 2 находится в двоичной нормальной форме. Любая булева грамматика может быть переведена в эквивалентную ей булеву грамматику в двоичной нормальной форме [13]. Далее любое упоминание булевой грамматики подразумевает, что она находится в двоичной нормальной форме.

3. Алгоритм

В работе [22] предложен алгоритм синтаксического анализа для булевых грамматик, основанный на матричных операциях. В данном разделе будет предложено расширение этого алгоритма для нахождения приближенного решения задачи поиска путей в ациклических графах с ограничениями в терминах булевых грамматик и использованием реляционной семантики, а также показана его корректность.

Матричные алгоритмы синтаксического анализа позволяют достичь более высокой производительности за счет использования эффективных методов перемножения матриц [21, 22]. Также за счет использования матриц возможно более компактное хранение данных. Например, в случае, если входной ациклический граф является деревом, между любой парой вершин графа будет существовать всего один путь. Если же ациклический граф не является деревом, то между одной парой вершин может быть в худшем случае $O(2^n)$ путей, поэтому хранить и обрабатывать данные для каждого пути отдельно

неэффективно. Так как информация о всех путях из вершины i в вершину j хранится в одной ячейке, то с помощью конъюнкции могут быть объединены нетерминалы, соответствующие разным путям из i в j , но при этом будут объединены также нетерминалы, соответствующие одному и тому же пути, что и требуется для решения задачи. В итоге результат работы алгоритма будет содержать приближение ответа сверху.

3.1 Описание алгоритма

Входные данные алгоритма:

- помеченный ориентированный ациклический граф $D = (V, E)$ с n вершинами. Без потери общности предположим, что n является степенью двойки.
- $G = (\Sigma, N, P)$ — булева грамматика в двоичной нормальной форме.

Результат работы алгоритма — матрица T , каждый элемент $T_{i,j}$ которой содержит множество нетерминалов $\{A_1, A_2, \dots, A_m\}$, где $A_k \in N$, являющееся надмножеством множества нетерминалов, таких, что $(i, j) \in R_{A_k}$.

Алгоритм использует следующие структуры данных:

1) Таблица T размером $n \times n$, представляющая собой верхнетреугольную матрицу, где каждый элемент $T_{i,j}$ является множеством нетерминалов.

Пусть $X \in (2^N)^{m \times l}$ и $Y \in (2^N)^{l \times n}$ — матрицы, элементами которой являются подмножества N , а $m, l, n \geq 1$. Тогда произведением матриц $X \times Y$ будет матрица $Z \in (2^N)^{m \times n}$, такая, что каждый её элемент $Z_{i,j}$ вычисляется по следующей форме:

$$Z_{i,j} = \bigcup_{k=1}^l X_{i,k} \times Y_{k,j}$$

Данное произведение можно представить в виде произведения $|N|^2$ булевых матриц: для всех пар нетерминалов $B, C \in N$ рассмотрим булеву матрицу Z^{BC} , элемент $Z_{i,j}^{BC}$ которой означает наличие пары (B, C) в ячейке матрицы $Z_{i,j}$. При этом Z^{BC} можно получить как произведение булевых матриц Z^B и Z^C . Благодаря такому переходу становится возможным свести решение задачи к вычислению произведения булевых матриц, для осуществления которого существуют весьма эффективные алгоритмы [23, 24].

2) Таблица M , каждый элемент $M_{i,j}$ которой принадлежит множеству пар нетерминалов $N \times N$.

Используя значения $M_{i,j}$ и правила грамматики, из множества пар нетерминалов можно получить множество нетерминалов:

$$T_{i,j} = f(M_{i,j}),$$

где $f: 2^{N \times N} \rightarrow 2^N$ для булевых грамматик определяется как

$$f(M) = \bigcup_{k=1}^{2^{|M|}} \{A \mid \exists A \rightarrow B_1 C_1 \& \dots \& B_m C_m \& \neg D_1 E_1 \& \dots \& \neg D_m' E_m' \in P: (B_t, C_t) \in M^k \text{ и } (D_t, E_t) \notin M^k \text{ для } \forall t, \}$$

где M^k — подмножество множества M .

Псевдокод модифицированного алгоритма Охотина приведен в листинге 1.

Элементы матрицы M рассчитываются группами с помощью выше определенного произведения подматриц из таблицы T , дающих аналогичный результат с поэлементным умножением. Схема расположения подматриц представлена на рисунках 3 и 4.

Алгоритм состоит из двух рекурсивных процедур:

- 1) *compute*(l, m) — рассчитывает значения $T_{i,j}$ для любых $l \leq i < j < m$.
- 2) *complete*(l, m, l', m') — определена для $l \leq m < l' \leq m'$ при $m - l = m' - l'$ и $m - l$ без потери общности являющимися степенью двойки.

Четыре входных параметра процедуры обозначают координаты подматрицы матрицы T , содержащей все элементы $T_{i,j}$, где $l \leq i < j < m$ и $l' \leq j < m'$. Координаты интерпретируются следующим образом: l, m — индексы начальной и конечной строки в таблице T , l', m' — индексы начального и конечного столбца.

Координаты обозначают пути в графе, номер начала которых лежит между l и m , а конец между l' и m' . Так как ранее уже посчитаны $T_{i,j}$ для $l \leq i < j < m$ и $l' \leq i < j < m'$, а также $M_{i,j}$ для $l \leq i < m$ и $l' \leq j < m'$, процедура *complete*(l, m, l', m') получает значения $T_{i,j}$ для $l \leq i < m$ и $l' \leq j < m'$.

Основные структуры данных, с которыми работает алгоритм, являются верхнетреугольными матрицами, поэтому на вершинах графа D должен быть задан линейный порядок, такой, что любое его ребро ведёт от вершины с меньшим номером к вершине с большим номером. Если D является ациклическим графом, то данного свойства легко добиться с помощью топологической сортировки графа. Алгоритм Тарьяна [25] позволяет осуществлять топологическую сортировку за линейное от количества вершин графа время.

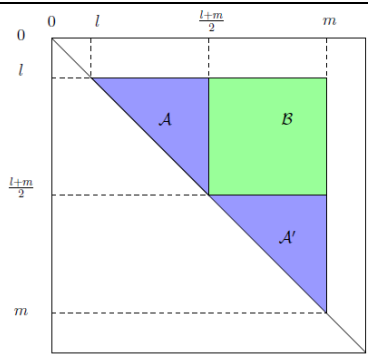


Рис. 3. Расположение подматриц для расчёта процедурой *compute* элементов таблицы M .

Fig. 3. Submatrices for calculating elements of M by «compute» procedure

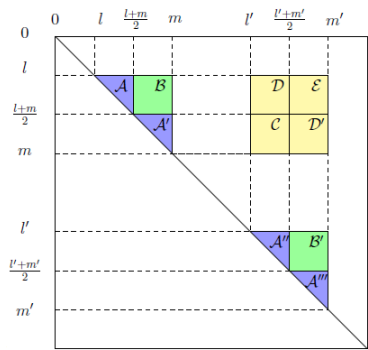


Рис. 4. Расположение подматриц для расчёта процедурой *complete* элементов таблицы M .

Fig. 4. Submatrices for calculating elements of M by «complete» procedure

```

/* D — входной помеченный ориентированный ациклический граф
/* G — входная булева грамматика в двоичной нормальной форме
1:  Main:
2:  n ← число вершин в D
3:  E ← {(i, j, a) | ребро из вершины i в j, помеченное символом a входит во
    множество рёбер D}
4:  N ← множество нетерминалов грамматики G
5:  P ← множество правил грамматики G
6:  T ← пустая матрица размером n × n
7:  M ← пустая матрица размером n × n
8:  D ← TopologicalSorting(D)
9:  for each (i, j, a) ∈ E
10:     Ti,j ← {A | A → a ∈ P}
11:     compute(0, n + 1)

12:  compute(l, m):
13:  if m - l ≥ 4 then
14:     compute(l,  $\frac{l+m}{2}$ )
15:     compute( $\frac{l+m}{2}$ , m)
16:  complete(l,  $\frac{l+m}{2}$ ,  $\frac{l+m}{2}$ , m)

17:  complete(l, m, l', m')
18:  if m - l = 1 and m < l' then
19:     Tl,l' ← f(Ml,l')
20:  else if m - l > 1 then
21:     B ← (l,  $\frac{l+m}{2}$ ,  $\frac{l+m}{2}$ , m)
22:     B' ← (l',  $\frac{l'+m'}{2}$ ,  $\frac{l'+m'}{2}$ , m')
23:     C ← ( $\frac{l+m}{2}$ , m, l',  $\frac{l'+m'}{2}$ )
24:     D ← (l,  $\frac{l+m}{2}$ , l',  $\frac{l'+m'}{2}$ )
25:     D' ← ( $\frac{l+m}{2}$ , m,  $\frac{l'+m'}{2}$ , m')
26:     E ← (l,  $\frac{l+m}{2}$ ,  $\frac{l'+m'}{2}$ , m')
27:     complete(C)
28:     MD ← MD ∪ (TB × TC)
29:     complete(D)
30:     MD' ← MD' ∪ (TC × TB')
31:     complete(D')
32:     ME ← ME ∪ (TB × TD')
33:     ME ← ME ∪ (TD × TB')
34:     complete(E)

```

Листинг. 1. Алгоритм поиска путей в ациклических графах с ограничениями в терминах булевых грамматик.

Listing. 1. Path querying on acyclic graphs using Boolean grammars.

3.2 Время работы и корректность алгоритма

Как было сказано выше, результатом работы алгоритма является матрица T , в ячейках которой будет содержаться некоторое множество нетерминалов. Покажем, что это множество будет содержать решение задачи.

Теорема 1. (*Корректность алгоритма для поиска путей в ациклических графах с ограничениями в терминах булевых грамматик с использованием реляционной семантики запросов*). Пусть даны помеченный ориентированный ациклический граф $D = (V, E)$ и булева грамматика $G = (\Sigma, N, P)$. Тогда для любых вершин i, j и для любого нетерминала $A \in N$, если $(i, j) \in R_A$, то $A \in T_{i,j}$.

Доказательство.

Рассмотрим такие вершины i, j , что $(i, j) \in R_A$ и существует $i\pi j$, такой, что $l(\pi) \in L(G_A)$ (т.е. существует дерево разбора для строки $l(\pi)$ и грамматики G с корнем в нетерминале A). Докажем индукцией по высоте дерева разбора строки $l(\pi)$, что $A \in T_{i,j}$.

База индукции. Если (i, j) — ребро графа (высота дерева разбора равна 1), то алгоритм корректен, исходя из инициализации матрицы T (строки 9-10 листинга 1).

Индукционный переход. Предположим, утверждение верно для всех деревьев разбора высотой h . Докажем, что теорема верна для деревьев разбора высотой $h + 1$. Рассмотрим дерево разбора для $l(\pi)$ высотой $h + 1$. Так как грамматика в нормальной форме, то у данного дерева будут поддеревья, выводящие подстроки $l(\pi)$ (возможно пересекающиеся). По свойству топологической сортировки для всех индексов k, t этих подстрок (начала и конца соответствующих путей) выполняется неравенство $i \leq k < t \leq j$. $T_{i,j}$ вычисляется как $f(M_{i,j})$. $M_{i,j}$, исходя из построения алгоритма, может быть получено тремя способами, в зависимости от положения подматрицы, в которой оно задано. Пусть $M_{i,j}$ — ячейка матрицы M_Z . В любом из случаев M_Z вычисляется как $M_Z \cup (T_X \times T_Y)$. По определению произведения $T_X \times T_Y$, для каждого пути из i в j будут получены все произведения нетерминалов из конъюнктов, выводящих все подстроки $l(\pi)$, такие, что путь $i\pi j$ разбит на две части вершиной k , такой, что $i < k < j$. По индукционному предположению, т.е. если $(i, k) \in R_B$, то $B \in T_{i,k}$ и если $(k, j) \in R_C$, то $C \in T_{k,j}$. Тогда произведение матриц $T_X \times T_Y$ даст все возможные пары нетерминалов $\{(B, C)\} \in M_{i,j}$ для всех подстрок. После применения функции f (подбора соответствующего правила грамматики), $T_{i,j}$ будет содержать нетерминалы,

полученные после применения правил. Тогда, исходя из правил грамматики, получим, что если $(i, j) \in R_A$, то $A \in T_{i,j}$ (объединяем существующие поддеревья для подстрок в одно дерево). А это значит, что утверждение верно для дерева разбора высотой $h + 1$. \square

Время работы алгоритма аналогично времени работы алгоритма Охотина [22] и составляет $O(|G|BMM(n) \log n)$, где $|G|$ — размер входной булевой грамматики, n — число вершин в графе D , $BMM(n)$ — время умножения булевых матриц размера $n \times n$.

4. Пример работы алгоритма

В данном разделе мы продемонстрируем работу предложенного алгоритма на небольшом примере. В качестве входного графа будет использован граф, изображенный на рисунке 1, а в качестве входной булевой грамматики — грамматика с рисунка 2.

В строках 9-10 листинга 1 происходит инициализация таблицы T , заполнение происходит на основании информации о рёбрах графа. Состояние T после инициализации показано на рисунке 5.

	0	1	2	3	4	5	6	7
0		{A}	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset
1			{A}	{B, D}	\emptyset	\emptyset	\emptyset	\emptyset
2				{B, D}	\emptyset	\emptyset	\emptyset	\emptyset
3					{C}	\emptyset	\emptyset	\emptyset
4						{A}	\emptyset	{C}
5							{B, D}	\emptyset
6								{C}
7								

Рис. 5. Таблица T после инициализации.

Fig. 5. Matrix T after initialization

После инициализации произойдет ряд рекурсивных вызовов процедур *compute* и *complete*. Дерево рекурсивных вызовов показано на рисунке 6.

Вызовы *compute*(0,2), *compute*(2,4), *compute*(4,6) и *compute*(6,8) из-за небольшого размера задачи будут обработаны тривиально — вызовом процедур *complete*(0, 1, 1, 2), *complete*(2, 3, 3, 4), *complete*(4, 5, 5, 6) и *complete*(6, 7, 7, 8) соответственно (строка 16 листинга 1). Так как для каждого из вызовов выполняется равенство $m = l'$ для координат подматриц, в

соответствующих ячейках матрицы останутся текущие множества нетерминалов без изменений.

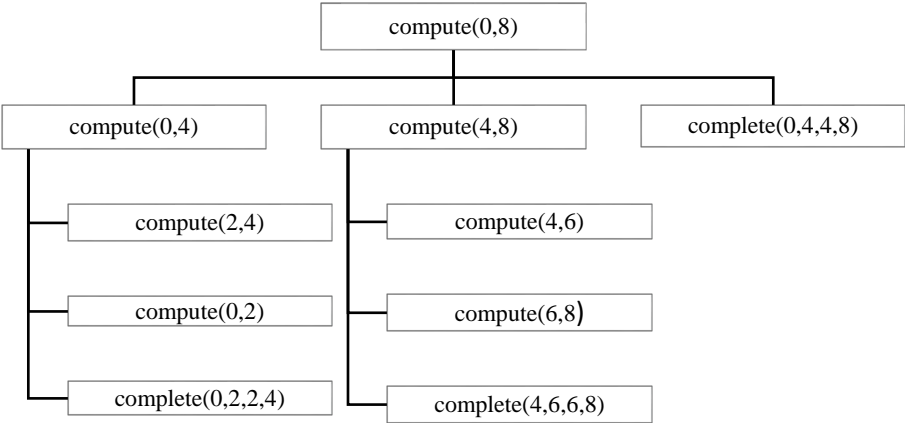


Рис. 6. Дерево рекурсивных вызовов процедур *compute* и *complete*.

Fig. 6. Recursive call tree for «*compute*» and «*complete*».

На рисунке 7 показаны подматрицы *T*, для которых будут производиться дальнейшие вычисления.

	0	1	2	3	4	5	6	7
0		{A}	complete(0,2,2,4)		complete(0,4,4,8)			
1								
2				{B,D}				
3					complete(4,6,6,8)			
4								
5						{A}		
6							{C}	
7								

Рис. 7. Подматрицы *T*, для которых будет вызвана процедура *complete*.

Fig. 6. Submatrices of *T* which will be processed by «*complete*».

Разберем подробнее вызов *complete*(0, 2, 2, 4).

1. Сначала будут определены координаты подматриц матрицы *T* (строки 21-26). $B = (0, 1, 1, 2), B' = (2, 3, 3, 4), C = (1, 2, 2, 3), D = (0, 1, 2, 3)$,

$$D' = (1, 2, 3, 4).$$

2. Затем произойдет вызов $complete(C) = complete(1, 2, 2, 3)$. В результате, множество в ячейке $T_{1,2}$ останется неизменным, $T_{1,2} = \{A\}$.
3. Далее вычисляем ячейку $M_{0,2}$ таблицы M (строка 28 листинга 1).
 $M_{0,2} = M_D = M_D \cup (T_B \times T_C) = M_{0,2} \cup (T_{0,1} \times T_{1,2}) = \{(A, A)\}.$
4. Выполняем $complete(D) = complete(0, 1, 2, 3)$. Согласно строкам 18-19 листинга 1, $T_{0,2} = f(M_{0,2}) = \emptyset$, так как во входной грамматике отсутствует подходящее правило.
5. $M_{D'} = M_{1,3} = M_{D'} \cup (T_C \times T_{B'}) = M_{1,3} \cup (T_{1,2} \times T_{2,3}) = \{B, D\} \cup (\{A\} \times \{B, D\}) = \{(B), (D), (A, B), (A, D)\}.$
6. Выполняем $complete(D') = complete(1, 2, 3, 4)$. Согласно строкам 18-19 листинга 1, $T_{1,3} = f(M_{1,3}) = \{B, D\}$.
7. Вычисляем ячейку $M_{0,3}$ (строки 32-33).
 $M_{0,3} = M_E = M_E \cup (T_B \times T_{D'}) = M_{0,3} \cup (T_{0,1} \times T_{1,3})$
 $= \{(A, B), (A, D)\}.$
 $M_{0,3} = M_E \cup (T_D \times T_{B'}) = M_{0,3} \cup (T_{0,2} \times T_{2,3})$
 $= \{(A, B), (A, D)\} \cup (\emptyset \times \{B, D\}) = \{(A, B), (A, D)\}.$
8. Выполняем $complete(E) = complete(0, 1, 3, 4)$.
 $T_{0,3} = f(M_{0,3}) = \{D\}.$

Вычисление $complete(4, 6, 6, 8)$ происходит аналогично. Состояние таблицы T после окончания работы процедур $complete(0, 2, 2, 4)$ и $complete(4, 6, 6, 8)$ приведено на рисунке 8.

	0	1	2	3	4	5	6	7
0		{A}	\emptyset	{D}	\emptyset	\emptyset	\emptyset	\emptyset
1			{A}	{B, D}	\emptyset	\emptyset	\emptyset	\emptyset
2				{B, D}	\emptyset	\emptyset	\emptyset	\emptyset
3					{C}	\emptyset	\emptyset	\emptyset
4						{A}	{D}	{C, S}
5							{B, D}	{B, S}
6								{C}
7								

Рис. 8. Таблица T после окончания работы $complete(0, 2, 2, 4)$ и $complete(4, 6, 6, 8)$.

Fig. 8. Matrix T after $complete(0, 2, 2, 4)$ и $complete(4, 6, 6, 8)$.

Теперь всё готово для работы процедуры $complete(0, 4, 4, 8)$. Разбиение T на подматрицы показано на рисунке 9, синим цветом выделены матрицы B (левый верхний угол) и матрица B' (правый нижний угол).

	0	1	2	3	4	5	6	7
0		{A}	\emptyset	{D}	\mathcal{D}		\mathcal{D}'	
1			{A}	{B, D}				
2				{B, D}	\mathcal{C}		\mathcal{D}'	
3								
4						{A}	{D}	{C, S}
5							{B, D}	{B, S}
6								{C}
7								

Рис. 9. Разбиение на подматрицы таблицы T для вычисления $complete(0, 4, 4, 8)$.

Fig. 9. Partition of matrix T for $complete(0, 4, 4, 8)$.

После определения координат подматриц, вызовется процедура $complete(C) = complete(2, 4, 4, 6)$. Расчёт элементов T_C происходит аналогично ранее описанным расчетам для матриц 2×2 , поэтому сразу приведем результат:

$$T_C = \begin{pmatrix} \{B, S\} & \emptyset \\ \{C\} & \emptyset \end{pmatrix}.$$

Далее вычисляем элементы оставшихся подматриц:

$$1. \quad M_D = M_D \cup (T_B \times T_C) = \begin{pmatrix} \emptyset & \{D\} \\ \{A\} & \{B, D\} \end{pmatrix} \times \begin{pmatrix} \{B, S\} & \emptyset \\ \{C\} & \emptyset \end{pmatrix} = \begin{pmatrix} \{(D, C)\} & \emptyset \\ \{(A, B), (A, S), (B, C), (D, C)\} & \emptyset \end{pmatrix}.$$

$$\text{После } complete(D) = complete(0, 2, 4, 6) \quad T_D = \begin{pmatrix} \{S\} & \emptyset \\ \{B, S\} & \emptyset \end{pmatrix}.$$

$$2. \quad M_{D'} = M_{D'} \cup (T_C \times T_{B'}) = \begin{pmatrix} \{B, S\} & \emptyset \\ \{C\} & \emptyset \end{pmatrix} \times \begin{pmatrix} \{D\} & \{C, S\} \\ \{B, D\} & \{B, S\} \end{pmatrix} = \begin{pmatrix} \{(B, D), (S, D)\} & \{(B, C), (B, S), (S, C), (S, S)\} \\ \{(S, D)\} & \{(C, C), (C, S)\} \end{pmatrix}.$$

$$\text{После } complete(D') = complete(2, 4, 6, 8) \quad T_{D'} = \begin{pmatrix} \emptyset & \{B\} \\ \emptyset & \emptyset \end{pmatrix}.$$

$$3. \quad \text{Теперь вычисляем } M_E = M_E \cup (T_B \times T_{D'}) = \begin{pmatrix} \emptyset & \{D\} \\ \{A\} & \{B, D\} \end{pmatrix} \times \begin{pmatrix} \emptyset & \{B\} \\ \emptyset & \emptyset \end{pmatrix} = \begin{pmatrix} \emptyset & \emptyset \\ \emptyset & \{(A, B)\} \end{pmatrix}.$$

$$\begin{aligned} \text{Затем } M_E &= M_E \cup (T_D \times T_{B'}) = \\ &= \begin{pmatrix} \emptyset & \emptyset \\ \emptyset & \{(A, B)\} \end{pmatrix} \cup \begin{pmatrix} \{S\} & \emptyset \\ \{B, S\} & \emptyset \end{pmatrix} \times \begin{pmatrix} \{D\} & \{C, S\} \\ \{B, D\} & \{B, S\} \end{pmatrix} = \\ &= \begin{pmatrix} \{(S, D)\} & \{(S, C), (S, S)\} \\ \{(B, D), (S, D)\} & \{(A, B), (B, C), (B, S), (S, C), (S, S)\} \end{pmatrix}. \\ \text{После } complete(E) &= complete(0, 2, 6, 8) \quad T_E = \begin{pmatrix} \emptyset & \emptyset \\ \emptyset & \{B\} \end{pmatrix}. \end{aligned}$$

Финальное состояние таблицы T представлено на рисунке 10.

	0	1	2	3	4	5	6	7
0		{A}	∅	{D}	{S}	∅	∅	∅
1			{A}	{B, D}	{B, S}	∅	∅	{B}
2				{B, D}	{B, S}	∅	∅	{B}
3					{C}	∅	∅	∅
4						{A}	{D}	{C, S}
5							{B, D}	{B, S}
6								{C}
7								

Рис. 10. Результирующее состояние таблицы T .

Fig. 10. Final state of matrix T .

Данный пример демонстрирует тот факт, что не всегда получается найти точное решение. Например, согласно таблице на рисунке 10, между вершинами 4 и 7 (ячейка $T_{4,7}$) проходит путь, конкатенация меток которого составляет слово из языка $L(G_S)$. На самом деле, между вершинами 4 и 7 проходят два пути: первый путь — ребро графа, помеченное символом “с”, а второй путь вида $4 \rightarrow 5 \rightarrow 6 \rightarrow 7$, метки рёбер которого составляют строку “abc”. И, если нетерминал $C \in T_{4,7}$ действительно выводит символ “с”, то $S \in T_{4,7}$ не выводит строку “abc” по причине того, что она выводится конъюнктом AB , отрицание которого есть в правиле $S \rightarrow DC \& \neg AB$. Это получается из-за того, что, хотя пара (A, B) и присутствовала в множестве нетерминалов $\{(A, B), (D, C)\}$ в ячейке $M_{4,7}$, функция f , которая рассматривает все подмножества данного множества, на подмножестве $\{(D, C)\}$ вернёт нетерминал S .

Несмотря на то, что стратегия рассматривания отдельных подмножеств пар нетерминалов дает лишние ответы, она гарантирует наличие в результатах настоящего ответа на задачу. Продемонстрируем это на следующем примере.

Рассмотрим пару вершин $(0, 4)$ в графе на рисунке 1. Между этой парой вершин есть два пути: путь $0 \rightarrow 1 \rightarrow 3 \rightarrow 4$, метки которого составляют строку “abc” и

путь $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4$, метки которого составляют строку “aabc”. Заметим, что “abc” $\in L(G_{AB})$, $L(G_{DC})$, а “aabc” $\in L(G_{DC})$, поэтому $(0, 4) \in R_S$. Во время работы алгоритма, в ячейке $M_{0,4}$ окажется множество $\{(A, B), (D, C)\}$. Если бы мы рассматривали всё множество сразу, то из-за отрицания (A, B) не нашлось бы подходящего правила, и тогда ячейка $T_{0,4}$ в результате содержала бы пустое множество. Но, как показано выше, в графе между парой вершин 0 и 4 есть уникальный путь, слово из меток рёбер которого выводится конъюнктом DC , а конъюнкт AB на самом деле выводит слово из меток рёбер другого пути между этими же вершинами. Именно поэтому, чтобы рассмотреть все возможные случаи и не отсечь настоящее решение, алгоритм рассматривает все подмножества, что дает аппроксимацию ответа сверху.

5. Заключение

В данной работе был предложен алгоритм, находящий приближенное решение задачи поиска путей в ациклических графах с ограничениями в терминах булевых грамматик и использованием реляционной семантики запросов. Была показана применимость предложенного алгоритма для поиска приближенного решения данной задачи. За счёт ограничения на тип входного графа и использования матричных операций, алгоритм является более асимптотически оптимальным, чем наивный итерационный алгоритм.

Определим несколько направлений для будущих исследований. Так как алгоритм работает только с ациклическими графами, открытым остается вопрос, возможно ли модифицировать алгоритм для работы с произвольными графами и сохранением такой же производительности. Также интерес представляет вопрос возможности эффективного преобразования произвольного входного графа в форму, позволяющую обработку данным алгоритмом, например, какой-либо вариант декомпозиции графа.

Благодарности

Авторы выражают признательность Кознову Дмитрию Владимировичу за оказанную помощь при написании настоящей статьи. Данная работа выполнена при поддержке гранта РФФИ 18-11-00100 и JetBrains Research.

Список литературы

- [1]. Barceló Baeza P. Querying graph databases. In Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems (PODS '13), 2013, pp. 175-188.

- [2]. Mendelzon A., Wood P. Finding Regular Simple Paths in Graph Databases. SIAM J. Computing, vol. 24, № 6, 1995, pp. 1235-1258.
- [3]. Anderson J. W. et al. Quantifying variances in comparative RNA secondary structure prediction. BMC bioinformatics, vol. 14, № 1, 2013.
- [4]. Chaudhary A., Faisal A. Role of graph databases in social networks. 2016.
- [5]. Warchał, Ł. Using Neo4j graph database in social network analysis. Studia Informatica, vol. 33, № 2A, 2012, pp. 271-279.
- [6]. Reps T. Program analysis via graph reachability. In Proceedings of the 1997 international symposium on Logic programming (ILPS '97), 1997, pp. 5-19.
- [7]. 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.
- [8]. Yuan H., Eugster P. An Efficient Algorithm for Solving the Dyck-CFL Reachability Problem on Trees. In Proceedings of the 18th European Symposium on Programming Languages and Systems: Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009 (ESOP '09), 2009, pp. 175-189.
- [9]. Neo4j's Graph Query Language: An Introduction to Cypher. Доступно по ссылке: <https://neo4j.com/developer/cypher-query-language/>, 02.06.2019.
- [10]. Rodriguez M. A. The gremlin graph traversal machine and language (invited talk). In Proceedings of the 15th Symposium on Database Programming Languages, 2015, pp. 1–10.
- [11]. Prud E., Seaborne A., et al. SPARQL query language for RDF. 2006.
- [12]. Okhotin A. Conjunctive grammars. Journal of Automata, Languages and Combinatorics, vol. 6, № 4, 2001, pp. 519–535.
- [13]. Okhotin A. Boolean grammars. Information and Computation, vol. 194, issue 1, 2004, pp. 19–48.
- [14]. Hellings J. Querying for Paths in Graphs using Context-Free Path Queries. CoRR abs/1502.02242, 2015.
- [15]. Hellings. J. Conjunctive context-free path queries. In: Proc. of ICDT'14, 2014, pp.119–130.
- [16]. Sevon P., Eronen L. Subgraph queries by context-free grammars. Journal of Integrative Bioinformatics, vol. 5, № 2, 2008.
- [17]. Zhang X. et al. Context-free path queries on RDF graphs. International Semantic Web Conference, 2016, pp. 632–648.
- [18]. Grigorev S., Ragozina A. Context-free path querying with structural representation of result. In Proceedings of the 13th Central & Eastern European

- Software Engineering Conference in Russia (CEE-SECR '17), Article 10, 7 pages.
- [19]. Azimov R. Sh., Grigorev S. V. Context-Free Path Querying by Matrix Multiplication. CoRR, vol. abs/1707.01007, 2017.
- [20]. Azimov R. Sh., Grigorev S. V. Path querying using conjunctive grammars. Proceedings of ISP RAS, 2018, vol. 30, issue 2, 149–166.
- [21]. Valiant L.G. General context-free recognition in less than cubic time. Journal of computer and system sciences, vol., № 2, pp. 308–315.
- [22]. Okhotin A. Parsing by matrix multiplication generalized to Boolean grammars. Theoretical Computer Science, vol. 516, 2014, pp. 101–120.
- [23]. Arlazarov V. L., Dinitz Y. A., Kronrod M. A., Faradzhev I. A. On economical construction of the transitive closure of an oriented graph. Dokl. Akad. Nauk SSSR, 194:3, 1970, pp. 487–488.
- [24]. Vassilevska Williams V. Multiplying matrices faster than Coppersmith-Winograd. In Proceedings of the 44th Symposium on Theory of Computing, (STOC2012), 2012, pp. 887–898.
- [25]. Tarjan R. E. Depth-first search and linear graph algorithms. In Proceedings of the 12th Annual Symposium on Switching and Automata Theory (swat 1971), 1971, pp. 114–121.

Path querying on acyclic graphs using Boolean grammars

¹ S.V. Grigorev <s.v.grigoriev@spbu.ru>

² E.N. Shemetova <katyacyfra@gmail.com>

¹ Saint Petersburg State University,

7/9, Universitetskaya nab., St. Petersburg, 199034, Russia.

² Saint Petersburg National Research University of Information Technologies,
Mechanics and Optics
(ITMO University),

49, Kronverkskiy pr., St. Petersburg, 197101, Russia.

Abstract. Graph data models are widely used in different areas of computer science such as bioinformatics, graph databases, social networks and static code analysis. One of the problems in graph data analysis is querying for specific paths. Such queries are usually performed by means of a formal grammar that describes the allowed edge-labeling of the paths. Path query

is said to be calculated using relational query semantics if it is evaluated to triple (A, v_1, v_2) , such that there is a path from v_1 to v_2 such that the labels on the edges of this path form a string derivable from the nonterminal A . As the regular and context-free languages have limited expressive power, we focus on a more expressive languages, namely the Boolean languages that use Boolean grammars to describe the labeling of paths. Although path querying using relational query semantics and Boolean grammars is known to be undecidable, in this work we propose a path querying algorithm on acyclic graphs which uses relational query semantics and Boolean grammars, and approximates the exact solution. To achieve better performance in compare with the naive algorithm, considered classes of graphs were limited to acyclic graphs.

Keywords: path querying; Boolean grammars; matrix operations; acyclic graph; DAG; boolean matrix; matrix multiplication.

DOI: [10.15514/ISPRAS-2016-1\(2\)-33](https://doi.org/10.15514/ISPRAS-2016-1(2)-33)

For citation: Grigorev S.V., Shemetova E. N. Path querying on acyclic graphs using Boolean grammars. *Trudy ISP RAN/Proc. ISP RAS*, vol. 1, issue 2, 2019. pp. 3-4 (in Russian). DOI: [10.15514/ISPRAS-2016-1\(2\)-33](https://doi.org/10.15514/ISPRAS-2016-1(2)-33)

References

- [1]. Barceló Baeza P. Querying graph databases. In Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems (PODS '13), 2013, pp. 175-188.
- [2]. Mendelzon A., Wood P. Finding Regular Simple Paths in Graph Databases. *SIAM J. Computing*, vol. 24, № 6, 1995, pp. 1235-1258.
- [3]. Anderson J. W. et al. Quantifying variances in comparative RNA secondary structure prediction. *BMC bioinformatics*, vol. 14, № 1, 2013.
- [4]. Chaudhary A., Faisal A. Role of graph databases in social networks. 2016.
- [5]. Warchał, Ł. Using Neo4j graph database in social network analysis. *Studia Informatica*, vol. 33, № 2A, 2012, pp. 271-279.
- [6]. Reps T. Program analysis via graph reachability. In Proceedings of the 1997 international symposium on Logic programming (ILPS '97), 1997, pp. 5-19.
- [7]. 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.
- [8]. Yuan H., Eugster P. An Efficient Algorithm for Solving the Dyck-CFL Reachability Problem on Trees. In Proceedings of the 18th European Symposium on Programming Languages and Systems: Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009 (ESOP '09), 2009, pp. 175-189.

- [9]. Neo4j's Graph Query Language: An Introduction to Cypher. Доступно по ссылке: <https://neo4j.com/developer/cypher-query-language/>, 02.06.2019.
- [10]. Rodriguez M. A. The gremlin graph traversal machine and language (invited talk). In Proceedings of the 15th Symposium on Database Programming Languages, 2015, pp. 1–10.
- [11]. Prud E., Seaborne A., et al. SPARQL query language for RDF. 2006.
- [12]. Okhotin A. Conjunctive grammars. Journal of Automata, Languages and Combinatorics, vol. 6, № 4, 2001, pp. 519–535.
- [13]. Okhotin A. Boolean grammars. Information and Computation, vol. 194, issue 1, 2004, pp. 19–48.
- [14]. Hellings J. Querying for Paths in Graphs using Context-Free Path Queries. CoRR abs/1502.02242, 2015.
- [15]. Hellings. J. Conjunctive context-free path queries. In: Proc. of ICDT'14, 2014, pp.119–130.
- [16]. Sevon P., Eronen L. Subgraph queries by context-free grammars. Journal of Integrative Bioinformatics, vol. 5, № 2, 2008.
- [17]. Zhang X. et al. Context-free path queries on RDF graphs. International Semantic Web Conference, 2016, pp. 632–648.
- [18]. Grigorev S., Ragozina A. Context-free path querying with structural representation of result. In Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR '17), Article 10, 7 pages.
- [19]. Azimov R. Sh., Grigorev S. V. Context-Free Path Querying by Matrix Multiplication. CoRR, vol. abs/1707.01007, 2017.
- [20]. Azimov R. Sh., Grigorev S. V. Path querying using conjunctive grammars. Proceedings of ISP RAS, 2018, vol. 30, issue 2, 149–166.
- [21]. Valiant L.G. General context-free recognition in less than cubic time. Journal of computer and system sciences, vol., № 2, pp. 308–315.
- [22]. Okhotin A. Parsing by matrix multiplication generalized to Boolean grammars. Theoretical Computer Science, vol. 516, 2014, pp. 101–120.
- [23]. Arlazarov V. L., Diniz Y. A., Kronrod M. A., Faradzhev I. A. On economical construction of the transitive closure of an oriented graph. Dokl. Akad. Nauk SSSR, 194:3, 1970, pp. 487–488.
- [24]. Vassilevska Williams V. Multiplying matrices faster than Coppersmith-Winograd. In Proceedings of the 44th Symposium on Theory of Computing, (STOC2012), 2012, pp. 887–898.

- [25]. Tarjan R. E. Depth-first search and linear graph algorithms. In Proceedings of the 12th Annual Symposium on Switching and Automata Theory (swat 1971), 1971, pp. 114-121.