МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждение
высшего образования
«Южный федеральный университет»

Институт математики, механики и компьютерных наук им.
И.И.Воровича
Кафедра алгебры и дискретной математики

Мурадьян Илья Валерьевич

Задача поиска графа-паттерна на помеченном графе

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
по направлению 01.03.02 — Прикладная математика и информатика

Научный руководитель –
доцент, к.ф.-м.н. Скороходов Владимир Александрович

Ростов-на-Дону – 2018

Содержание

1 Введение

Теория графов - относительно молодой и быстроразвивающийся раздел современной математики. В ней присутствует достаточно много серьёзных теоретических проблем, таких как, например, гипотеза Харари[5] о том, что если граф имеет более трёх рёбер, то его можно однозначно восстановить по подграфам, полученным удалением единственного ребра. Но вместе с этим теория графов успешно применяется для решения прикладных задач, возникающих в теории компьютерных сетей, машинном обучении, при проектировании и эксплуатации транспортных систем, в теории игр и т.д.

Одной из классических задач теории графов является задача об изоморфизме графов. В ней даны два графа G и G', для которых необходимо построить два биективных отображения δ' и γ таким образом, чтобы они переводили один граф в другой. Понятно, что такие отображения построить можно не всегда. Долгое время для этой задачи не могли найти хорошего алгоритма, и лучшей оставалась оценка $T(n) = \exp(O(\sqrt{n \log n}))$, где n - число вершин, но в 2015 году Ласло Бабай опубликовал статью [6], в которой показывается оценка $T(n) = \exp((\log n)^{O(1)})$, которая очень близка к полиномиальной. Тем не менее, приведённый в этой статье алгоритм всё ещё имеет слишком большую вычислительную сложность для того, чтобы применяться на практике для больших графов. Тем более, задача зачастую стоит немного по-другому: в графе G надо найти частичный подграф [1], изоморфный графу G', что сразу усложняет любой существующий – переборный или более продвинутый – алгоритм.

В приложениях, однако, часто можно ослабить условие задачи изоморфизма, определенным образом пометив вершины или дуги графов G и G', то есть сопоставив каждой вершине этих графов какой-то элемент множества L, а каждой дуге – элемент множества M. В этом случае на отображения δ и γ накладываются достаточно жесткие условия: δ может перевести x в x', а γ – u в u', только если вершины x и x' и дуги u и u' помечены одинаково. В случае, если множества меток L и M достаточно велики, количество перебираемых для отображений δ и γ вариантов сильно сокращается, что улучшает вычислительную сложность используемых для решения этой задачи алгоритмов.

Задачу изоморфизма помеченных графов также можно поставить более общем и полезном на практике виде: найти такой частичный подграф графа G, который будет изоморфен в обозначенном выше смысле графу G'. Такую задачу называют задачей поиска паттерна на помеченном графе.

В статье [9] приведён алгоритм нахождения граф-паттерна на помеченном графе в случае, если дуги графа не помечены (то есть в формулировке нет множества M), а метки вершин графа G' – уникальны. В подготовленной мной выпускной бакалаврской работе приведено описание этого алгоритма, а также его модификация для случая, когда метки графа G' не уникальны, и допускаются метки на дугах. Такая модификация расширяет применение рассмотренных в статье [9] методов на целый класс прикладных задач. Например, при поиске паттернов на фотографиях часто бывает так, что разыскиваемый паттерн состоит из нескольких одинаковых частей. Таков, например, паттерн лица или солнцезащитных очков. Более того, на таких паттернах зачастую важно учитывать расстояние между их частями, которое отлично моделируется весами на дугах графов-паттернов. В данной работе показано, как учитывать веса на дугах графов-паттернов для нахождения соответствующего данному паттерну частичного подграфа.

Кроме распознавания образов на фотографиях, поиск паттернов на помеченных графах может использоваться и как инструмент социальной инженерии. Так, например, в 2016 году в средствах массовой информации широко освещалась деятельность так называемых групп смерти ([2]), получивших распространение в различных социальных сетях. При наличии полученной административными методами переписки жителей определённого города или района, можно построить граф, вершинами которого будут аккаунты пользователей, а дугами будут связаны те пользователи, которые вели друг с другом переписку. Особым образом следует пометить вершины, соответствующие аккаунтам «целевой аудитории» групп смерти – подростков в возрасте от 13 до 19 лет, а также дуги, соответствующие переписке с различными стоп-словами типа «суицид», «4:20» и пр. После этого следует составить граф-паттерн, выглядящий, например, как граф-звезда (т.е. связный граф, степень всех вершин которого, кроме одной, равна 1 [3]) с 10 листьями-подростками, а дуги графа-паттерна должны быть помечены в обозначенном выше смысле. После того, как паттерн построен, можно применить алгоритм из данной работы и найти все соответствующие ему подграфы, после чего, проводя их анализ, установить координаторов «групп смерти».

Все представленные в данной работе алгоритмы были реализованы на языке Python с использованием библиотеки GraphTool. Оба этих проекта – проекты с открытым исходным кодом и исключительно полной документацией ([8], [10]), которая достаточно активно использовалась в процессе разработки. Для контроля версий и работы за разными машинами также использовалась система контроля версий Git, исходный код которой также

открыт. Несмотря на обилие открытых источников, для более подробного ознакомления с этой системой я выбрал книгу [7].

2 Основные понятия

Дадим основные определения, которые будут использоваться в этой работе.

Определение 1. *Ориентированным графом (в дальнейшем – просто **графом**) будем называть двойку $G(V, E)$, где:*

▷ $V, |V| > 0$ - *множество вершин графа;*

▷ $E \subset V \times V$ - *множество дуг графа;*

Определение 2. *Граф $G(V, E)$ будем называть конечным, если множества V и E конечны.*

Определение 3. *Тройку $G(V, E, \omega)$ будем называть взвешенным графом, если $G'(V, E)$ – граф, а $\omega : E \to \mathbb{R}$ – весовая функция.*

Определение 4. *Будем говорить, что дуга $e = (u, v)$ исходит из вершины u и заходит в вершину v. Вершину u будем называть началом, а v – концом дуги.*

Определение 5. *Будем говорить, что вершина u смежна с вершиной v, если $\exists e = (u, v) \in E$. Обозначим множество вершин, смежных с u, через $\Gamma(u)$. Множество $Inc(u) = \{(u, v) \in E | v \in V\}$ назовём множеством инцидентных u дуг.*

Введём понятия пути и цепи.

Определение 6. *Путём называется последовательность дуг $\mu = (e_i = (u_i, u_{i+1}))_{i=0}^{m-1}$. Число m назовём длиной пути и обозначим через $|\mu|$. Будем говорить, что $\mu.s = u_0$ – это начало пути μ, а $\mu.t = u_m$ – это конец пути μ. Для удобства будем считать, что на графе G существует $|V|$ путей нулевой длины, каждый из которых представляет собой пустую последовательность дуг, а его начало и одновременно конец суть некоторая вершина графа G. Путь ненулевой длины μ называется контуром, если его начало совпадает с концом. Если граф не содержит контуров, он называется бесконтурным.*

Если разрешить двигаться по дугам графа в обратном направлении, из определения пути мы получим определение цепи на графе.

Определение 7. *Цепью на графе G называется последовательность дуг $\lambda = (e_i)_{i=0}^{m-1}$ такая, что $\forall i \in [0; m-2]_{\mathbb{N}}$ дуги e_i и e_{i+1} имеют ровно одну общую вершину (но необязательно начало дуги e_i совпадает с концом e_{i+1}). Число m назовём длиной цепи и обозначим через $|\lambda|$. Будем говорить, что вершина дуги e_0, не инцидентная дуге e_q – это начало цепи λ, а вершина дуги e_{m-1}, не инцидентная дуге e_{m-2} – это конец цепи λ. Цепь ненулевой длины λ называется циклом, если её начало совпадает с концом.*

Необходимо также ввести некоторые определения, специфичные для данной работы.

Определение 8. *Расстоянием $d(u, v)$ между вершинами графа u и v называется длина наименьшего пути, началом которого является вершина u, а концом - вершина v, или наоборот. Если ни одного такого пути не существует, то расстояние между вершинами полагается равным* $\inf$.

Определение 9. *Неориентированным расстоянием $dn(u, v)$ между вершинами графа u и v называется длина наименьшей цепи, началом которой является вершина u, а концом - вершина v. Если ни одной такой цепи не существует, то неориентированное расстояние между вершинами полагается равным* $\inf$.

Определение 10. *Диаметром $diam(G)$ связного графа G называется максимальное из неориентированных расстояний между его вершинами.*

Заметим, что в связном конечном графе неориентированные расстояния между любыми двумя вершинами всегда конечны, поэтому и диаметр такого графа также будет конечен. В данной работе мы будем рассматривать лишь связные и конечные графы, поэтому все они будут иметь конечный диаметр.

Нам также понадобятся определения, связанные с подграфами и частичными графами.

Определение 11. *Граф $G'(V', E')$ называется частичным графом графа $G(V, E)$, если $V' = V, E' \subset E$.*

Определение 12. *Граф $G'(V', E')$ называется подграфом графа $G(V, E)$, если $V' \subset V, E' = \{(v, u) \in E | v \in V', u \in V'\}$.*

Определение 13. *Граф $G'(V', E')$ называется частичным подграфом графа $G(V, E)$, если $V' \subset V, E' \subset \{(v, u) \in E | v \in V', u \in V'\}$.*

3 Задача о поиске паттерна на помеченном графе

3.1 Постановка задачи

Пусть L – непустое конечное множество **(множество меток)**. Пусть $G(V, E)$, $G'(V', E')$ – ориентированные связные графы. Будем называть граф G архивным графом, граф G' – графом-паттерном, или шаблонным графом.

Введём отображения $l : V \to L$, $l' : V' \to L$, сопоставляющие вершинам архивного и шаблонного графов соответствующие метки. При этом мы требуем, чтобы введённые на шаблонном графе метки были уникальны, т.е. отображение l' – инъективно.

Определение 14. *Совпадением на графе G будем называть частичный подграф $\widehat{G}(\widehat{V}, \widehat{E})$ графа G такой, что:*

1. *Существует биективное отображение $m_{\widehat{G}} : V' \to \widehat{V}$.*
2. $\forall v' \in V' : l'(v') = l(m_{\widehat{G}}(v'))$
3. $\forall (v'_1, v'_2) \in E' : (m_{\widehat{G}}(v'_1), m_{\widehat{G}}(v'_2)) \in E$

Пусть для вершины $v \in V$ существует некоторое совпадение $\widehat{G}(\widehat{V}, \widehat{E})$ на графе G такое, что $v \in \widehat{V}$. Тогда вершину v будем называть подходящей паттерну G' по совпадению $\widehat{G}$, иначе – неподходящей. В случае, если вершина v подходит по совпадению $\widehat{G}$, вершину $m^{-1}_{\widehat{G}}(v) \in V'$ назовём соответствующей данной вершине v. Заметим, что вершина v может подходить паттерну по нескольким совпадениям, но в силу инъективности отображения l' ей может соответствовать лишь одна вершина $v' \in V'$.

Через $\widetilde{V} \subseteq V$ обозначим множество всех подходящих паттерну G' вершин. Наша задача и будет состоять в отыскании этого подмножества. Опишем используемый нами алгоритм.

Пусть $T = \emptyset \cup \mathcal{C}^1_{V'}$ – все 0-элементные и 1-элементные подмножества множества вершин графа-паттерна. Построим отображение $f_0 : V \to T$, заданное следующим:

$$v' \in f_0(v) \Leftrightarrow l(v) = l'(v'). \tag{1}$$

Нетрудно убедиться, что в силу инъективности отображения l, введённое отображение f_0 действительно имеет областью значений множество T.

Алгоритм будет строиться на изменении отображения f_0, поэтому для удобства нам потребуется ввести операцию над подобными отображениями. Пусть $f_1, f_2 : V \to 2^{V'}$. Обозначим $f_2 = Exclude(f_1, v_0(\in V), v'_0(\in V'))$, если выполнено следующее:

1. $\forall v \neq v_0 \in V : f_1(v) = f_2(v)$.

2. $v'_0 \notin f_2(v_0)$.

3. $\{v'_0\} \cup f_2(v_0) = f_1(v_0)$

Ясно, что по отображению f_1 легко построить отображение $Exclude(f_1, v_0, v'_0)$, просто исключая вершину v'_0 из множества $f_1(v_0)$.

3.2 Алгоритм исключения по локальным условиям

Для начала дадим несколько определений.

Определение 15. *Пусть дан граф $G(V, E)$. Пусть $v \in V$. Тогда обозначим через $Adj(v)$ множество смежных с ней вершин: $Adj(v) = \{u \in V | \exists (v, u) \in E\}$. Если $\widetilde{V} \subset V$, то $Adj(\widetilde{V}) = \bigcup\limits_{\widetilde{v} \in \widetilde{V}} Adj(\widetilde{v})$. В частности, $Adj(\emptyset) = \emptyset$.*

Определение 16. *Пусть $G(V, E)$, $G'(V', E')$ – архивный и шаблонный графы соответственно. Пусть задано отображение $f : V \to T$, где множество T определено выше. Путь также $v \in V$. Назовём предикатом локальных условий следующий предикат:*

$$LCC(f, v) = \forall u' \in Adj(f(v)) \exists u \in Adj(v) : u' \in f(u). \quad (2)$$

Ниже приведён алгоритм LCCE – исключения по локальным условиям.

Вход: графы $G(V, E)$, $G'(V', E')$, отображение $f_0 : V \to T$, число итераций N
Выход: измененённое отображение f_N

```
1 начало LCCE
2   для i = 1, 2, .., N :
3     f_i := f_{i-1} для v ∈ V :
4       если ¬LCC(f_i, v) тогда
5         f_i(v) := ∅
6   вернуть f_N
```

Алгоритм 3.1: Алгоритм исключения по локальным условиям

Данный алгоритм на каждой итерации «прореживает» отображение f_0, то есть всегда выполнено $\forall v \in V f_i(v) \subset f_{i-1}$. Такое прореживание исключает на каждой итерации i из списка кандидатов те вершины v, для которых оказывается, что хотя бы для одной из вершин $u' \in Adj(f_i(v))$ не существует такой вершины $u \in Adj(v)$, что $f(u) = u'$. Заметим, что схожая идея используется в различных классических алгоритмах теории графов, например, в алгоритме Беллмана-Форда. Следуя той же схеме доказательства, которая используется, например, при обосновании того факта, что

алгоритм Беллмана-Форда находит на бесконтурном графе кратчайшие пути за количество итераций, равное его диаметру, мы придём к тому, что для бесконтурного графа G' алгоритм $LCCE$ отработает «до конца» за $diam(G')$ итераций. Говоря «до конца», мы имеем в виду, во-первых, что последующие итерации алгоритма более не удалят из отображения f ни одного кандидата, а во-вторых, все вершины $v \in V$, для которых $f(v)$ не пусто, будут действительно вершинами некого графа-совпадения, ведь для каждой вершины $u = f(v)$, каждый начинающийся в ней путь (а в силу бесконтурности графа, этот путь конечный, и его длина не превышает диаметра графа) имеет некий соотносящийся с этим путём путь в графе G, начинающийся в вершине v.

Для графов-паттернов с циклами всё обстоит несколько сложнее. На таких графах существуют бесконечные пути, а потому для них применение приведённого выше алгоритма со сколь угодно большим числом итераций не гарантирует того, что оставшиеся вершины-кандидаты будут в действительности вершинами некоторого графа-совпадения. Поэтому для графов с циклами мы будем применять, помимо алгоритма $LCCE$, ещё и алгоритм, описанный ниже.

3.3 Алгоритм проверки контуров

Пусть $\mathcal{K}_0$ – это набор контуров графа G'. В этот набор достаточно включить все простые контуры этого графа. Пусть $\mathcal{C}_0 \in \mathcal{K}_0$ – какой-то из рассматриваемых контуров. Тогда если $\mathcal{C}_0 = ((v'_0, v'_1), ..., (v'_{r-1}, v'_0))$ и $v'_0 \in f(v_0)$, то алгоритм CCE, приведённый ниже, исключит вершину v'_0 из множества $f(v_0)$, если окажется, что нет контура, начинающегося в вершине

v_0 и соответствующего (в смысле отображения f) контуру $\mathcal{C}_0$.

Вход: графы $G(V, E)$, $G'(V', E')$, отображение $f_0 : V \to T$
Выход: измененённое отображение f_N

начало CCE
  для каждого $\mathcal{C}_0 \in \mathcal{K}_0$ **выполнять**
    Пусть (v'_0, v'_1) – первая дуга конутра $\mathcal{C}_0$.
    $\mathcal{A} := \emptyset$
    $\mathcal{A}_0 := \emptyset$
    для всех $v_0 : f(v_0) \neq \emptyset$ **выполнять**
      $\mathcal{A}_0 := \mathcal{A}_0 \cup \{v_0\}$
      $\mathcal{A} := \mathcal{A} \cup \{(v_0, v_0, 0)\}$
    для $s = 1, 2, ..., |\mathcal{C}_0|$ **:**
      Пусть (q_0, q_1) – s-я дуга конутра $\mathcal{C}_0$.
      $\mathcal{B} := \emptyset$
      для каждого $(v, v_0, s-1) \in \mathcal{A}$ **выполнять**
        для $v' \in Adj(v)$ **:**
          если $q_1 \in f(v')$ **тогда**
            $\mathcal{B} := \mathcal{B} \cup \{(v', v_0, s)\}$
      $\mathcal{A} := \mathcal{B}$
    для каждого $v_0 \in \mathcal{A}$ **выполнять**
      если $(v_0, v_0, |\mathcal{C}_0|) \notin \mathcal{A}$ **тогда**
        $f(v_0) = \emptyset$
  вернуть f_N

Алгоритм 3.2: Алгоритм проверки контуров

Нахождение всех контуров графа и такой их обход – задачи с очень высокой вычислительной сложностью, но на практически встречающихся графах, тем не менее, основная работа будет сделана алгоритмом $LCCE$. Впрочем, ничего страшного не произойдёт, если рассматриваться будут не все циклы – можно попытаться выделить подграф совпадения и по неточному списку кандидатов. Здесь много зависит от конкретного вида графов и стоящей перед нами задачи.

Сами авторы статьи ожидают, что граф-паттерн будет содержать достаточно мало циклов, поэтому приводимая ими оценка $O(|\mathcal{K}_0| n_t (|V| + |E|))$, где $n_t = \max\limits_{\mathcal{C}_0 \in \mathcal{K}_0} n_{\mathcal{C}_0}$, где $n_{\mathcal{C}_0}$ – это количество вершин в V, соответствующих контуру $\mathcal{C}_0$.

3.4 Общий алгоритм исключения вершин-кандидатов

Два вышеприведённых алгоритма объединяются в следующем цикле, который выполняется до тех пор, пока из отображения f ещё удаляются какие-то вершины-кандидаты.

Вход: графы $G(V, E)$, $G'(V', E')$, отображение $f : V \to T$
Выход: измененённое отображение f
1 **начало** Exclusion
2 **до тех пор, пока** *вершины-кандидаты продолжают удаляться из отображения* f **выполнять**
3 $LCCE(G, G', f)$ $CCE(G, G', f)$
4 **вернуть** f

Алгоритм 3.3: Алгоритм исключения вершин-кандидатов

Формируя начальное отображение f исходя из совпадений по меткам архивного графа и графа-паттерна, мы передаём его алгоритму Exclusion и получаем на выходе новое, «прореженное» отображение, состоящее исключительно из истинных кандидатов.

4 Модифицированная задача

4.1 Постановка задачи

Пусть, как и прежде, L – непустое конечное множество, называемое **множеством меток**. Пусть $G(V, E)$, $G'(V', E')$ – ориентированные графы. Граф G', кроме того, связный. Будем называть граф G архивным графом, граф G' – графом-паттерном, или шаблонным графом.

Введём отображения $l : V \to L$, $l' : V' \to L$, сопоставляющие вершинам архивного и шаблонного графов соответствующие метки. Никаких дополнительных требований на эти отображения мы уже не накладываем.

Пусть также $\mathcal{X}$ – это непустое (возможно, бесконечное) множество **(множество характеристик)** с заданным на нём бинарным отношением $\rho : \mathcal{X} \times \mathcal{X} \to \{0, 1\}$

Мы вводим множество характеристик для того, чтобы, например, работать со взвешенными графами. Действительно, если положить

$$\mathcal{X} = \mathbb{R}$$

,

$$\rho(x, y) = \begin{cases} 1, & |x - y| < \varepsilon \\ 0, & |x - y| \geq \varepsilon \end{cases}, \varepsilon > 0$$

,

мы получим отношение ρ, заданное, как отношение «примерного равенства». Его мы и будем использовать для сравнения весов дуг архивного графа и графа-паттерна.

Введём также «помечающие отображения» для дуг графа: $\chi : E \to \mathcal{X}$, $\chi' : E' \to \mathcal{X}$.

Несколько изменится и определение совпадения.

Определение 17. *Совпадением на графе G будем называть частичный подграф $\widehat{G}(\widehat{V}, \widehat{E})$ графа G такой, что:*

1. *Существует биективное отображение $m_{\widehat{G}} : V' \to \widehat{V}$.*
2. $\forall v' \in V' : l'(v') = l(m_{\widehat{G}}(v'))$
3. $\forall e' = (v'_1, v'_2) \in E' : \rho(\chi'(e'), \chi((m_{\widehat{G}}(v'_1), m_{\widehat{G}}(v'_2)))) = 1$
4. $\forall (v'_1, v'_2) \in E' : (m_{\widehat{G}}(v'_1), m_{\widehat{G}}(v'_2)) \in E$

Пусть для вершины $v \in V$ существует некоторое совпадение $\widehat{G}(\widehat{V}, \widehat{E})$ на графе G такое, что $v \in \widehat{V}$. Тогда вершину v будем называть подходящей паттерну G' по совпадению $\widehat{G}$, иначе – неподходящей. В случае, если вершина v подходит по совпадению $\widehat{G}$, вершину $m^{-1}_{\widehat{G}}(v) \in V'$ назовём соответствующей данной вершине v.

Через $\widetilde{V} \subseteq V$ обозначим множество всех подходящих паттерну G' вершин. Наша задача и будет состоять в отыскании этого подмножества. Опишем используемый нами алгоритм.

Пусть $T = \emptyset \cup \mathcal{C}^1_{V'}$ – все подмножества множества вершин графа-паттерна. Построим отображение $f_0 : V \to T$, заданное следующим:

$$v' \in f_0(v) \Leftrightarrow l(v) = l'(v'). \tag{3}$$

Нетрудно убедиться, что в силу инъективности отображения l, введённое отображение f_0 действительно имеет областью значений множество T.

Алгоритм будет строиться на изменении отображения f_0, поэтому для удобства нам потребуется ввести операцию над подобными отображениями. Пусть $f_1, f_2 : V \to 2^{V'}$. Обозначим $f_2 = Exclude(f_1, v_0(\in V), v'_0(\in V'))$, если выполнено следующее:

1. $\forall v \neq v_0 \in V : f_1(v) = f_2(v)$.
2. $v'_0 \notin f_2(v_0)$.
3. $\{v'_0\} \cup f_2(v_0) = f_1(v_0)$

Ясно, что по отображению f_1 легко построить отображение $Exclude(f_1, v_0, v'_0)$, просто исключая вершину v'_0 из множества $f_1(v_0)$.

4.2 Измененный алгоритм исключения по локальным условиям

4.3 Измененный алгоритм проверки циклов

5 Список литературы

1. Берж, К. Теория графов и ее применения. / Берж К.– М.: Издательство иностранной литературы, 1962.– 320 с.

2. Галина Мурсалиева. Группы смерти (18+) / Галина Мурсалиева // Новая газета.– URL: https://www.novayagazeta.ru/articles/2016/05/16/68604-gruppy-smerti-18.– 2016.

3. Емцева, Е.Д., Солодухин, К.С. Дискретная математика. Часть 3 (Курс лекций). / Емцева, Е.Д., Солодухин, К.С. // URL: https://abc.vvsu.ru/books/l_diskrmat3/page0008.asp

4. Ерусалимский, Я.М. Дискретная математика. Теория, задачи, приложения: учеб. пособие / Я.М. Ерусалимский.– М.: Вузовская книга, 2009.– 288 с.

5. Лекции по теории графов. / Емеличев В. А. [и др.] – М.: Наука, 1990. – 384 с.

6. Laszlo Babai. Graph Isomorphism in Quasipolynomial Time / Laszlo Babai // 2015.

7. Loeliger, J., McCullough, M. Version Control with Git. / Jon Loeliger, Matthew McCullough.– Sebastopol: O'Reilly, 2012.– 436p.

8. Python 3.6.5 documentation. // URL: https://docs.python.org/3/index.html.

9. Towards Practical and Robust Labeled Pattern Matching in Trillion-Edge Graphs / Tahsin Reza [и др.] // 2017.

10. Welcome to graph-tool's documentation! // URL: https://graph-tool.skewed.de/static/doc/index.html.