Об одной задаче анализа топологии коммуникационных сетей

Б. Ф. Мельников, П. П. Стариков, Ю. Ю. Терентьева

***Аннотация*—В статье рассматриваются эвристические нелинейные алгоритмы проверки 2-кратной рёберной связности. Это вводимый нами термин, граф обладает 2-кратной рёберной связностью в том случае, когда между любыми двумя его вершиами существуют по крайней мере 2 рёберно-независимых пути; важно отметить, что это понятие не совпадает с более исследованным в литературе понятием рёберной 2-связности графа. Эти процедуры являются промежуточными для крайне востребованных алгоритмов проверки наличия двух географически разнесённых маршрутов для всех возможных направлений связи заданной сети.**

Для последней задачи в литературе ранее описаны полиномиальные алгоритмы – и небольшие их модификации дают и алгоритмы решения «нашей» задачи. Однако мы их не рассматриваем: они достаточно сложны для реализации, а нам нужны такие алгоритмы, в которых часто нужно вносить некоторые модификации в процессе его выполнения.

В рассматриваемых в настоящей статье задачах мы используем такие вспомогательные алгоритмы, которые можно назвать эвристическими – вследствие этого эвристическими можно назвать и полные версии рассматриваемых алгоритмов. В статье мы рассматриваем 3 различных группы алгоритмов, предназначенных для проверки 2-кратной рёберной связности.

А именно, сначала мы приводим явные алгоритмы такой проверки – т. е. алгоритмы, в котором нужные пути строятся явным образом. Далее, в следующем алгоритме, мы проверяем возможное наличие общего цикла для любых двух вершин заданного графа. Третий алгоритм заключается в том, что мы проверяем 2-кратную рёберную связность на основе, во-первых, отсутствия мостов и, во-вторых, наличия для любой вершины графа хоть какого-нибудь включающего её цикла.

Мы приводим схему доказательства эквивалентности этих трёх алгоритмов, а после этого – краткое описание вычислительных экспериментов.

***Ключевые слова*—коммуникационные сети, графы, рёберная связность, эвристические алгоритмы.**

I. ВВЕДЕНИЕ, МОТИВАЦИЯ И ПРЕДВАРИТЕЛЬНЫЕ СВЕ́ДЕНИЯ

В начале предлагаемой статьи приведём возможное для неё альтернативное название: «Эвристические нелинейные алгоритмы проверки 2-кратной рёберной связности». Прокомментируем такое альтернативное название «по частям».

Во-первых, 2-кратная рёберная связность графа. Это *вводимый нами* термин, в литературе, относящейся к теории графов ([1], [2], [3], [4], [5], [6] и мн. др.), мы соответствующего понятия не нашли. Согласно нашему определению, граф обладает 2-кратной рёберной связностью в том случае, когда между любыми двумя его вершиами существуют по крайней мере 2 рёберно-независимых пути. При этом отметим, что «похожее по названию» понятие рёберной 2-связности графа [1] – оно «похоже» ещё и по смыслу, но всё-таки существенно отличается. Сразу отметим, что в обоих случаях имеются естественные обобщения – получающиеся заменой 2 на k; о таких обобщениях мы в настоящей статье говорить не будем – но предполагаем к ним вернуться в одной из следующих публикаций.

Также сразу приведём ещё и такое замечание. Кроме рёберной связности – аналогичным образом определяется и 2-кратная вершинная связность [2], подход к моделированию которой мы предполагаем рассмотреть в одной из следующих публикаций. Таким образом, даже для «2-кратности» ($k = 2$) возможны *четыре* разные задачи – из которых в настоящей статье рассматривается одна.

Теперь вернёмся к различию между 2-кратной рёберной связностью и рёберной 2-связностью (понятно, примерно такие же различия возникают и при замене 2 на k). Самый простой пример, показывающий различие определений, – следующий: есть несколько вершин графа, имеющиеся рёбра образуют один цикл (можно сказать, Гамильтонов цикл) – а других рёбер нет. При этом между любыми вершинами существуют 2 рёберно-назависимых пути (т. е. «наше» определение выполняется), но для получения несвязанного графа достаточно удаление 1 ребра (т. е. определение рёберной 2-связности не выполняется).

Продолжим комментировать альтернативное название статьи; все предыдущие комментарии были «во-первых», теперь же – *во-вторых*, про нелинейные алгоритмы. Для «не нашей» задачи, т. е. задачи проверки рёберной 2-связности графа, существуют полиномиальные алгоритмы [3] – и важно отметить, что небольшие их модификации

Борис Феликсович Мельников, Центр информационных технологий и систем органов исполнительной власти (bf-melnikov@yandex.ru).

Павел Павлович Стариков, Центр информационных технологий и систем органов исполнительной власти (pstarikov@inevm.ru).

Юлия Юрьевна Терентьева, Центр информационных технологий и систем органов исполнительной власти (terjul@mail.ru).

Статья получена 11 марта 2022 г.

[1] `https://ru.wikipedia.org/wiki/Рёберно_k-связный_граф` и др. Согласно приведённому там тексту (версия от 3 марта 2022 г.), «рёберно k-связный граф – граф, который остаётся связным после удаления не более чем $k - 1$ рёбер». По-видимому, здесь в очередной раз видна плохая калька с плохого английского языка, в этом русском варианте «не хватает артикля» – т.е. в данном случае нужно «удаления не более чем $k - 1$ *любых* рёбер». См. также английскую версию, `https://en.wikipedia.org/wiki/K-edge-connected_graph`, там текст *немного* более удачный.

[2] Исследование рёберной и вершинной связностей нужны для рассмотрения различных моделей атак на рассматриваемые сети связи. *Куда* потенциальный противник наносит удары: по вершине («городу») – или по ребру («каналу связи между двумя городами»)?

[3] См. процитированную выше ссылку, также в сноске.

дают и алгоритмы решения «нашей» задачи. Однако мы эти алгоритмы не рассматриваем: для реализации они достаточно сложны[4], а мы обычно ([7], [8], [9], [10] и мн. др.) рассматриваем такие задачи, в которых по требованию заказчика часто нужно *вносить в алгоритм некоторые модификации в процессе его выполнения* – а для сложных в исполнении алгоритмов это всегда очень проблематично.

Также отметим, что не всегда (не во всех конкретных алгоритмах) надо «гнаться» за их возможной линейностью, а также, обобщая, за возможным понижением степени полиномиальных алгоритмов. Одна из возможных причин этого (но при этом не единственная причина) – сложность реализации некоторых линейных алгоритмов, при том, что часто многие другие алгоритмы, предназначенные для решения тех же самых задач, дают вполне приемлемые временны́е результаты, но при этом существенно проще для реализации. Более того, нередко требуется «онлайновая» модификация некоторых алгоритмов[5] – а её, конечно же, проще и быстрее проводить для более простых алгоритмов и соответствующих им компьютерных программ. Всё это полностью видно на примере рассматриваемой здесь задачи проверки 2-кратной рёберной связности.

В-третьих – про эвристики в алгоритмах. Далее будет понятно, что и в рассматриваемых в настоящей статье задачах мы нередко используем такие вспомогательные алгоритмы, которые можно назвать эвристическими; по-видимому, вследствие этого эвристическими можно назвать и полные версии рассматриваемых алгоритмов. Среди эвристик «относящихся к задачам теории графов» и применявшихся ранее в наших проектах, упомянем описанные в [7], [10], [11], [12]. Однако для задач, рассматриваемых в настоящей статье, могут представлять интерес и эвристики, описанные в [13], [14], [15].

… Итак, мы «по частям» привели объяснение возможного альтернативного названия статьи, «Эвристические нелинейные алгоритмы проверки 2-кратной рёберной связности»; однако выбранное нами название – «менее обязывающее», просто потому, что, возможно, определённая нами 2-кратная рёберная связность графа всё-таки где-то когда-то исследовалась.

Приведём краткое содержание статьи по разделам. В разделах II – IV мы рассматриваем различные алгоритмы (точнее, группы алгоритмов), предназначенные для проверки 2-кратной рёберной связности. А именно, в разделе II мы приводим явные алгоритмы такой проверки – т. е. алгоритмы, в котором нужные пути строятся явным образом. В алгоритме, описываемом в разделе III, мы проверяем возможное наличие общего цикла для любых двух вершин заданного графа. А в разделе IV мы проверяем 2-кратную рёберную связность на основе: во-первых, отсутствия мостов, и, во-вторых, наличия для любой вершины графа хоть какого-нибудь включающего её цикла.

В разделе V приведена схема доказательства эквивалентности трёх алгоритмов проверки 2-кратной рёберной связности. Краткое описание вычислительных экспериментов приведено в разделе VI. Статья завершается заключением (раздел VII) – в котором специально выделены такие особенности рассматриваемой задачи, которые могут являться основой для последующего улучшения алгоритмов её решения.

Перед изложением основного материала статьи необходимо пояснить специфику используемых нами стандартных понятий теории графов. Конечно, главный термин – сам граф, обозначаемый стандартно, $G = (V, E)$; граф в нашем случае – неориентированный, не имеет петель и не является мультиграфом. В отличие от большинства наших предыдущих публикаций на тему графовых алгоритмов ([7], [8], [9], [10] и др.), в модели, рассматриваемой в настоящей статье, вершины графа *не имеют* привязки к координатной плоскости – поэтому, как и в наших менее связанных с данной тематикой «графовых» публикациях ([11], [12] и др.), можно сказать, что здесь мы рассматриваем «чистые» алгоритмы теории графов.

Приведём ещё несколько необходимых определений[6]. Путь – некоторая последовательность рёбер[7,8] вида

$$\big(\{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{n-1}, v_n\}\big). \qquad (1)$$

Если в (1) $v_0 = v_n$, то путь можно называть циклом.

Мост – это ребро, удаление которого увеличивает число компонент связности. Эквивалентное определение: некоторое ребро является мостом в том и только в том случае, если оно не содержится ни в одном цикле.

Рёберно-независимых и вершинно-независимые пути определяются естественным образом: в рёберно-независимом пути все рёбра в (1) различны (т. е. нет совпадающих пар вида $\{v_i, v_{i+1}\}$ и $\{v_j, v_{j+1}\}$), а в вершинно-независимом пути различны все вершины (т. е. в (1) нет совпадающих элементов вида v_i и v_j).

II. ЯВНЫЙ АЛГОРИТМ ПРОВЕРКИ 2-КРАТНОЙ РЁБЕРНОЙ СВЯЗНОСТИ

В этом разделе мы рассматриваем явный алгоритм проверки 2-кратной рёберной связности – т. е. алгоритм, в котором просто строятся явным образом нужные 2 пути. И можно заранее отметить, что этот алгоритм аналогичен применяемому в [10] алгоритму, включающему метод ветвей и границ (МВГ) – несмотря на то, что в алгоритмах настоящей статьи:

- во-первых, нет «привязки» к координатам вершин – и расстояния считаются не «геометрически», а обычным способом теории графов, [1] и мн. др.;
- во-вторых, в явном виде МВГ не используется.

Также заранее отметим следующее. Во всех алгоритмах мы при необходимости считаем, что предварительно был применён алгоритм Дейкстры. В частности, в алгоритме II.1, – что он был применён для обеих вершин,

[4] И, конечно, это тоже может представить тему отдельной публикации.

[5] Это обычно связано со вновь поступающими требованиями заказчиков программного продукта: заранее эти требования нередко не формулируются, а становятся понятными заказчикам только после получения первых конкретнов результатов работы программы.

[6] По Википедии и др. Отметим, что приведённые определения «адаптированы» под наши задачи.

[7] Иногда путь определяется как последовательность вершин – в нашей статье удобнее используемое нами определение.

[8] В рассматриваемых в настоящей статье задачах можно (и даже желательно) считать, что все рёбра приведённой последовательности различны; см. далее.

между которыми проверяется существование путей. В связи с этим также заранее отметим, что сложность первых двух предлагаемых в этом разделе алгоритмов $\sim N^3$ – поскольку в них именно алгоритм Дейкстры является «наиболее сложной составляющей».

Рассматриваемый первым (вспомогательный) алгоритм II.1. строит *один* путь между заданными вершинами – путём выбора точки «посередине». Как мы увидим далее, выбираемый этим алгоритмом путь фактически и является разделяющим элементом для МВГ – при том, что, как мы уже отмечали, в явном виде МВГ мы не используем.

Алгоритм II.1. *Построение одного псевдооптимального пути между двумя выбранными вершинами.*

Вход:

- неориентированный граф $G = (V, E)$
- и пара вершин $A, B \in V$.

Целевая функция: суммарная длина пути.

Выход:

- `true` в случае существования пути; при этом также выдаётся сам псевдооптимальный путь между вершинами A и B (как множество рёбер), и выбранная на шаге 4 алгоритма вершина графа [9].
- либо `false` при невозможности такого построения [10].

Обозначение результата: $\mathrm{WayII.1}(G, A, B)$.

Метод.

Шаг 1. Если $A = B$, то выход из алгоритма с ответом `true` и $\varnothing$ в качестве множества рёбер.

Шаг 2. Если $\{A, B\} \in E$, то выход из алгоритма с ответом `true` и $\{\{A, B\}\}$ в качестве множества рёбер.

Шаг 3. Построение векторов расстояний от A и B до всех вершин графа; обозначение этих векторов – $L_A(X)$ и $L_B(X)$ для всех $X \in V$; для построения этих векторов дважды применяется алгоритм Дейкстры. Если хотя бы в одном из этих векторов имеется значение ∞, то выход из алгоритма с ответом `false`.

Шаг 4. Выбор некоторой вершины $C \in V$ ($C \neq A$, $C \neq B$), такой что

$$L_A(C) + L_B(C) + \left|L_A(C) - L_B(C)\right| \to \min$$

(любой из таковых).

Шаг 5. Выход из алгоритма с ответом

$$\mathrm{WayII.1}(G \setminus B, A, C) \cup \mathrm{WayII.1}(G \setminus A, C, B)$$

(при этом ответ `true` выдаётся в случае обоих положительных результатов, иначе выдаётся `false`).

Конец описания алгоритма.

Приведём некоторые комментарии к описанному алгоритму. Для выбора точки «посередине» используется специально подобранная эвристика, которую, однако, мы подробно комментировать не будем: см. описание шага 4. Также ещё раз отметим, что сложность этого алгоритма $\sim N^3$ – поскольку в нём дважды используется алгоритм Дейкстры, а остальные шаги алгоритма имеют меньшую сложность.

[9] При её наличии, а также при необходимости её дальнейшего использования (в качестве «побочного эффекта»).

[10] Когда граф несвязан.

Далее рассмотрим применение алгоритма II.1 – алгоритмы II.2 и II.3. В первом из них мы используем т. н. табуированные вершины – которые можно считать *полным аналогом* табуированных разрешающих элементов упомянутого выше метода ветвей и границ [11]; о конкретных вариантах применения таких элементов непосредственно в «классическом варианте» МВГ [12] см., например, в [15], [16]. Однако, как несложно понять, во входные данные алгоритма II.1 подобные табуированные элементы включать незачем: это привело бы к его излишнему усложнению.

Алгоритм II.2. *Проверка существования двух рёберно-независимых путей между двумя выбранными вершинами (вспомогательный рекурсивный алгоритм).*

Вход:

- неориентированный граф $G = (V, E)$,
- пара вершин $A, B \in V$
- и множество табуированных вершин для первого пути $V_T \subseteq V$.

Выход: `true` в случае существования двух рёберно-независимых путей; иначе `false`.

Обозначение результата: $\mathrm{OtvII.2}(G, A, B, V_T)$.

Метод.

Шаг 1. Если

$$\neg\mathrm{WayII.1}(G \setminus V_T, A, B),$$

то выход из алгоритма с ответом `false`.

Шаг 2. Если для вершины C, зафиксированной на шаге 1 при вызове WayII.1, выполнено условие

$$\mathrm{WayII.1}((G \setminus V_T) \setminus C, A, B),$$

то выход из алгоритма с ответом `true`.

Шаг 3. Выход из алгоритма с ответом

$$\mathrm{OtvII.2}(G, A, B, V_T \cup \{C\}).$$

Конец описания алгоритма.

Теперь – его непосредственное применение.

Алгоритм II.3. *Проверка существования двух рёберно-независимых путей между двумя выбранными вершинами (нерекурсивный алгоритм-«вызывалка»).*

Вход:

- неориентированный граф $G = (V, E)$
- и пара вершин $A, B \in V$.

Выход: `true` в случае существования двух рёберно-независимых путей; иначе `false`.

Метод.

Вернуть в качестве ответа $\mathrm{OtvII.2}(G, A, B, \varnothing)$.

Конец описания алгоритма.

Алгоритм, отвечающий на основной вопрос настоящей статьи, опишем очень кратко – без каких-либо конкретных деталей.

Алгоритм II.4. *Проверка существования двух рёберно-независимых путей между любыми двумя вершинами.*

Вход: неориентированный граф $G = (V, E)$.

[11] Поэтому и сам алгоритм II.2 можно рассматривать как упрощённую («неявную») версию МВГ.

[12] В том числе – в незавершённом его варианте, truncated branch-and-bound method.

Выход: `true` в случае существования двух рёберно-независимых путей между любыми двумя вершинами; иначе `false`.

Метод.

Применить алгоритм II.3 для всех пар вершин и вернуть в качестве ответа `true` в случае всех ответов `true` этого алгоритма.

Конец описания алгоритма.

Понятно, что при применении чисто формальной оценки сложность алгоритма II.3 превышает N^3 (точнее, не менее N^4), поэтому – также чисто формально – сложность алгоритма II.3 не менее N^6. Казалось бы, такое значение совершенно неприемлемо (особенно в будущем, при рассмотрении задач с количеством вершин графа порядка 10 000) – но реальные вычислительные эксперименты (раздел VI) показывают практическую возможность применения алгоритмов настоящего раздела [13].

В заключение раздела отметим, что все приведённые здесь алгоритмы были описаны явным образом – но, конечно, в связи с наличием шага 4 алгоритма II.1 их нужно называть эвристическими.

III. ПРОВЕРКА 2-КРАТНОЙ РЁБЕРНОЙ СВЯЗНОСТИ НА ОСНОВЕ НАЛИЧИЯ ОБЩИХ ЦИКЛОВ

В алгоритме, описываемом в настоящем разделе, мы проверяем наличие общего цикла для любых двух вершин заданного графа. Алгоритм несложен – однако, аналогично алгоритму из раздела II, при построении чисто формальных оценок мы можем получить [14], что его сложность существенно превышает N^3.

Сто́ит заранее отметить такое «глубинное» отличие этого алгоритма от алгоритма из раздела II: в отличие от него, алгоритм, рассматриваемый здесь, не переносится на случай, когда 2-кратную рёберную связность мы будем определять не «на основе рёбер», а «на основе вершин» – в то время как алгоритм из предыдущего раздела при подобном переносе практически не изменится.

Как и в предыдущем разделе, мы приведём более одного алгоритма – однако их не нужно применять «последовательно»: описанные ниже алгоритмы III.1 и III.2 представляют собой альтернативные варианты.

Итак, здесь мы проверяем условие того, что *любые две вершины графа должны входить в какой-либо общий цикл*. В простом варианте мы строим бинарное отношение «входят в общий цикл», проставляя только положительные ответы для всех пар вершин цикла (т. е. «запоминая» эти вершины).

Приведём формальное описание алгоритма.

Алгоритм III.1. *Проверка наличия общего цикла для двух любых вершин (простой вариант).*

Вход: неориентированный граф $G = (V, E)$.

Выход: `true` в случае существования общего цикла для двух любых вершин; иначе `false`.

Метод.

Шаг 1. Создать 2-мерный массив для бинарного отношения «вершины входят в общий цикл»; первоначальное заполнение этого массива – все элементы равны `false` (т. е. ни одна пара «не входит»).

Шаг 2. Выбор какого-либо элемента массива со значением `false`. Пусть это – элемент, соответствующий вершинам графа i и j.

Шаг 3. Выполнить попытку построения цикла [15] между вершинами графа i и j. В случае невозможности такого построения – выход из алгоритма с ответом `false`.

Шаг 4. Для всех элементов цикла, отличных от вершин i и j (пусть очередная такая вершина – k), заполнение значениями `true` элементов массива, соответствующим i и k, а также j и k.

Шаг 5. Если все элементы массива равны `true` – то выход из алгоритма с ответом `true`.

Шаг 6. Переход на шаг 2.

Конец описания алгоритма.

Для эвристического варианта этого алгоритма мы *в процессе* построения на множестве вершин графа бинарного отношения «входят в общий цикл» [16] *одновременно* строим и транзитивное замыкание этого отношения [17]. Отметим, что наличие мостов (оно будет рассмотрено далее) здесь также является препятствием.

Понятно, что для большинства конкретных задач время работы приведённого алгоритма можно сильно уменьшить, если строить бинарное отношение «не до конца» [18]; неформально это можно объяснить следующим образом: строить транзитивное замыкание бинарного отношения существенно проще, чем искать новый цикл в графе. Поэтому для самого простого варианта подобного улучшения работы мы получаем следующий алгоритм.

Алгоритм III.2. *Проверка наличия общего цикла для двух любых вершин (эвристический вариант).*

Вход: неориентированный граф $G = (V, E)$.

Выход: `true` в случае существования общего цикла для двух любых вершин; иначе `false`.

Метод: он совпадает с методом алгоритма III.1 – с описанным далее дополнительным шагом 4′, выполняемым после шага 4.

[13] Разница с наилучшими результатами (других алгоритмов) для выбираемых вариантов проблемы даёт увеличение времени менее, чем на порядок (менее, чем в 10 раз) – см. подробнее ниже. При этом (также ниже) станет очевидно, что алгоритмы, описанные в настоящем разделе, в сравнении с последующими алгоритмами:

- во-первых, значительно больше зависят от конкретных входных данных;
- и, во-вторых, в значительно большей степени могут быть улучшены путём добавления различных эвристик.

Всё это действительно показывает возможность практического применения рассмотренных здесь алгоритмов.

[14] В настоящей статье мы делать этого не будем.

[15] Конкретные варианты алгоритма построения (для этого и следующего алгоритмов) мы здесь описывать не будем – отметим только два следующих обстоятельства. Во-первых, мы в первую очередь рассматриваем варианты этого алгоритма, являющиеся упрощением алгоритма II.2. Во-вторых, разные конкетные варианты алгоритмов мы предполагаем исследовать в последующих публикациях – немного об этом см. ниже в заключении.

[16] Понятно, в такой цикл, длина которого больше или равна 3.

[17] При этом, *как и обычно в подобных ситуациях*, подобное построение можно не выполнять до конца: транзитивное замыкание эвристически строится и по подмножеству строимого бинарного отношения. Точные описания *нескольких* возможных алгоритмов (а их действительно несколько: они различаются вспомогательными алгоритмами ответа на вопрос, надо ли продолжать строить исходное бинарное отношение или «транзитивно замыкать ту его часть, которая уже построена») и их сравнительные временны́е результаты мы предполагаем рассмотреть в будущих публикациях. Пока же среди таковых рассмотрим только алгоритм III.2 и отметим, что при описании этих алгоритмов мы будем использовать приёмы, применяемые в МВГ.

[18] Об одном из многих вариантов такого улучшения уже было кратко упомянуто в предыдущей сноске.

Шаг 4′. Имеющийся вариант бинарного отношения «вершины входят в общий цикл» преобразуется в транзитивное замыкание этого отношения.

Конец описания алгоритма.

IV. ПРОВЕРКА 2-КРАТНОЙ РЁБЕРНОЙ СВЯЗНОСТИ НАЛИЧИЕМ ЦИКЛОВ И ОТСУТСТВИЕМ МОСТОВ

В этом разделе мы рассматриваем самый простой (по-видимому) алгоритм настоящей статьи – по крайней мере, самый простой для исполнения (реализации). Кроме этого можно сказать, что описываемый далее алгоритм может с первого взгляда показаться не очень сильно отличающимся от алгоритмов, рассмотренных в разделе предыдущем. Однако, по-видимому, это всё-таки не так [19].

Итак, здесь мы проверяем 2-кратную рёберную связность на основе:

- во-первых, отсутствия мостов,
- и, во-вторых, наличия для любой вершины графа хотя бы одного проходящего через неё цикла.

Для первой из этих двух *целей* существуют различные варианты алгоритмов – см. классические варианты в [1], [6], [17] и др., а также описание сравнительно нового алгоритма в [18]; поэтому рассмотрим только (достаточно простой) вариант алгоритма для второй из них. Более того, всюду ниже этот алгоритм IV.1 будем также называть основным алгоритмом этого раздела, решающим задачу проверки 2-кратной рёберной связности.

Алгоритм IV.1. *Проверка наличия для любой вершины графа хотя бы одного проходящего через неё цикла (эвристический вариант).*

Вход: неориентированный граф $G = (V, E)$.

Выход: `true` в случае существования для любой вершины графа хотя бы одного проходящего через неё цикла; иначе `false`.

Шаг 1. Создать массив для предиката «вершина входит в хоть один цикл»; первоначальное заполнение этого массива – все элементы равны `false`.

Шаг 2. Выбор какого-либо элемента массива со значением `false`. Пусть это – элемент, соответствующий вершине графа i.

Шаг 3. Выполнить попытку построения цикла [20], проходящего через вершину i. В случае невозможности такого построения – выход из алгоритма с ответом `false`.

Шаг 4. Для всех элементов цикла заполнение значениями `true` соответствующих им элементов массива.

Шаг 5. Если все элементы массива равны `true` – то выход из алгоритма с ответом `true`.

Шаг 6. Переход на шаг 2.

Конец описания алгоритма.

[19] И поэтому в будущем мы предполагаем провести более подробное исследование подхода, обозначенного в настоящем разделе. Немного подробнее см. в заключении, раздел VII.

Кроме того, именно дальнейшее развитие подхода, рассмотренного в настоящем разделе, мы в первую очередь предполагаем применять при рассмотрении развития более общей задачи – т. е. для создания алгоритмов проверки наличия *трёх и более* рёберно-независимых путей между любой парой вершин заданного графа.

[20] См. сноску к алгоритму из предыдущего раздела.

V. ОБ ЭКВИВАЛЕНТНОСТИ ТРЁХ АЛГОРИТМОВ

Приведём только *схему доказательства* эквивалентности трёх алгоритмов проверки 2-кратной рёберной связности – само полное доказательство несложно. Будем в этом разделе обозначать алгоритмы следующим образом:

(2) явный алгоритм, раздел II;
(3) алгоритм проверки наличия общего цикла, раздел III;
(4) алгоритм проверки отсутствия мостов и наличия для любой вершины графа хоть какого-нибудь включающего её цикла, раздел IV.

Эквивалентность (2)$\Leftrightarrow$(3) очевидна, она просто следует из определений теории графов.

Эквивалентность (3)$\Leftrightarrow$(4) доказывается следующим образом. Мы для рассматриваемого графа строим бинарное отношение «вершины входят в общий цикл» – являющееся отношением эквивалентности. Если при этом *не все* вершины попали в один класс эквивалентности – то это означает, что либо граф не является связанным, либо содержим мосты; обе эти возможности противоречат сделанным предположениям.

VI. КРАТКОЕ ОПИСАНИЕ ВЫЧИСЛИТЕЛЬНЫХ ЭКСПЕРИМЕНТОВ

Повторим ещё раз, что в статье рассматриваются *очень предварительные* версии алгоритмов для задач, которые мы предполагаем рассматривать и в дальнейшем, причём для значительно бóльших размерностей. Поэтому к результатам вычислительных экспериментов не стóит относиться как к чему-то окончательно определённому. Кроме того, необходимо отметить, что несмотря на уже отмеченную «непривязанность» вершин графа к координатам, задача в будущем предполагается быть встроенной в комплекс программ, в котором все вершины с системой координат связаны – и всё это нашло отражение в генерации входных данных.

Очень важно отметить, что каждая из упомянутых далее нескольких констант «подвергалась нормальному распределению» – т. е. изменению по нормальному распределению с указанным математическим ожиданием, равным указанному значению этой константы, и очень малой дисперсией. Поэтому, например, значение $N_1 = 40$ не означает, вообще говоря, что в сгенерированном графе окажутся ровно 40 вершин – но, конечно, случайная величина, равная числу вершин графа до применения «точек концентрации» (см. ниже) действительно равна 40.

Само проведение вычислительных экспериментов (в частности, генерация входных данных для получения приемлемых результатов) производилось следующим образом. Задавалась константа N_1 (порядка 40 – в будущем мы предполагаем рассматривать значительно бóльшие размерности), и в единичный квадрат согласно равномерному закону распределения помещались N_1 точек. Среди них случайно выбирались N_2 (обычно 3) «точки концентрации» – и в единичные квадраты с центрами в каждой из этих точек и длиной стороны 0.1 добавлялись по N_3 (обычно порядка 6) новых случайных точек [21]; понятно, что всё это увеличивало размерность задачи.

[21] Конечно, мы специальным образом обрабатывали возможный выход точки за границы «большого» единичного квадрата.

Для такой полученной размерности случайно генерировался граф – методом, аналогичным [11] и др.; отметим также, что в [10] мы начали рассматривать и другие подходы к подобной случайной генерации – однако в настоящей статье мы их не применяли.

К полученным точкам мы случайным образом добавляли рёбра. Конкретно, к первоначально пустому множеству рёбер мы для каждой точки добавляли число рёбер, равномерно распределённое от 1 до N_4 (обычно порядка $N_1/10$; см. также замечание в следующем абзаце) – причём добавляемые рёбра выбирались по следующему правилу. Ближайшая точка соединялась ребром с рассматриваемой, если значение сгенерированной случайной величины не превышало значения R_1 (обычно порядка 0.1); в противном случае рассматривалась вторая по близости точка – и так далее. Конечно, при этом рёбра нужно было заранее упорядочить по длине – но, во-первых, сложность соответствующего алгоритма приемлема (она порядка $N^2 \cdot \ln(N)$ для общего числа N вершин), и, во-вторых, нам важна скорость работы основных алгоритмов обработки данных – а не алгоритма их генерации.

Приведём ещё одно замечание про константу N_4. Из сказанного выше легко заключить, что большее значение этой константы даёт, вообще говоря, большее число рёбер в сгенерированном графе. Именно поэтому описанные далее результаты вычислительных экспериментов мы – для выбранного значения N_1 – приводим для двух элементов: сами алгоритмы (это будут столбцы) и значения константы N_4 (это будут строки).

Таким образом, как мы уже отметили, описанный здесь алгоритм случайной генерации графов (повторим, что со значительно бóльшими значениями констант – прежде всего N_1) мы применяем для проверки алгоритмов и в задачах, «привязанных» к координатам, – конкретные ссылки на статьи уже были приведены. А в нашей ситуации от координат мы отказываемся – *начиная с этого момента*[22]; теперь мы рассматриваем сгенерированные таким образом обычные графы. Сами результаты вычислительных экспериментов мы описываем очень кратко: мы предполагаем, что об этом будет сделана особая публикация, которая включит, среди прочего, фрагменты компьютерных программ.

Вычислительные эксперименты проводились по следующей методике. Выбирались значения

$$N_1 = 40, \quad N_2 = 3, \quad N_3 = 6, \quad R_1 = 0.1;$$

про величины дисперсий для получения значений соответствующих случайных величин подробно писать не будем[23]. Для значений, соответствующих рассматриваемой строке (напомним, что в ней находится значение константы N_4), генерировался граф – методом, кратко описанным выше. Именно этот граф подавался на вход каждому из алгоритмов, описанных в столбцах. (И, конечно, для каждого сгенерированного графа *результаты* работы всех алгоритмов оказывались одинаковы – нас в оставшейся части раздела интересует *время* их работы.)

В первую очередь сгенерированный граф подавался на вход алгоритму IV.1; время работы этого алгоритма *для рассматриваемого графа* принималось равным 1 (т. е. мы считали, что это – ровно одна временна́я единица; специально повторим, что она «объявлялась таковой» именно для рассматриваемого графа).

Далее считалось время для оставшихся алгоритмов, отвечающих на основной вопрос статьи – алгоритмов II.4, III.1 и III.2. Значения времени[24] запоминались – и впоследствии усреднялись для 30 генераций (для каждого значения константы N_4). При этом усреднение проводилось специальным вспомогательным алгоритмом: отбрасывались 5 наибольших и 5 наименьших значений, для остальных значений считалось среднее арифметическое. В окончательных результатах мы использовали 3 значащие десятичные цифры.

(Таким образом, нельзя сказать, что эти средние арифметические имеют размерность времени; здесь точнее говорить, что усредняются некоторые *безразмерные* величины, имеющие смысл отношений.)

Результаты вычислений приведены в следующей таблице I.

Таблица I
Относительное время работы алгоритмов

N_4 \ **Алгоритм**	**II.4**	**III.1**	**III.2**
2	2.12	1.34	1.31
4	2.01	1.12	1.12
8	1.84	1.05	1.04

Приведённые результаты, конечно, *очень предварительные* – хотя бы просто потому, что при реализации алгоритмов мы пользовались *нашей интерпретацией* эвристик. Поэтому реализацию каждого алгоритма, скорее всего, можно улучшить – и заранее неясно, для какого именно алгоритма подобное улучшение даст самый лучший результат.

VII. ЗАКЛЮЧЕНИЕ

Окончательного (и даже промежуточного) вывода – пока просто нет; мы привели описания возможных практических алгоритмов решения рассматриваемй задачи и кратко описали направления дальнейших исследований. Пока также непонятно, будут ли обобщено относительное время работы рассмотренных алгоритмов на существенно бóльшие размерности – что и является одной из наших окончательных целей. Также повторим, что *для каждого* из реализованных алгоритмов возможно его «программистское» улучшение – что может привести к изменению их относительных временны́х результатов.

И, как уже было отмечено, рассмотренные в статье алгоритмы проверки независимости путей между двумя вершинами графа (а также построения независимых путей) имеют высокую практическую значимость в теории связи. В качестве дальнейшего развития данной задачи мы предполагаем реализовать распараллеливание рассмотренных нами алгоритмов – прежде всего алгоритмов *последовательного выбора* рёбер (причём также и для алгоритмов проверки, и для алгоритмов построения).

[22] Однако, конечно, «первоначальная привязка» к координатам нужна – для получения «естественных» входных данных.

[23] Кратко. Каждое из этих значений для своего распределения рассматривалось как математическое ожидание M. Значение σ обычно выбиралось равным M/5. После вычисления производились специальные «запрещения» очень малых (в том числе отрицательных) и очень больших значений.

[24] Ещё раз повторим: эти значения считались в своих единицах для каждого рассматриваемого графа.

Ускорение работы представленных алгоритмов позволит использовать их в онлайн-режиме – с целью получения оперативных данных в процессе эксплуатации и управления сетью связи.

Среди других направлений ближайшей работы отметим следующие (уже упомянутые в основном тексте статьи):

- существенное увеличение размерности рассматриваемых частных случаев задачи – и, соответственно, адаптация алгоритмов и реализация соответствующих компьютерных программ, которые подходят для этих размерностей;
- замену в рассматриваемых задачах рёберной независимости на вершинную независимость;
- обобщение задачи – для k рёберно-независимых путей (вершинно-независимых путей) при $k > 2$.

Список литературы

[1] Харари Ф. *Теория графов* // М.: Мир, 1973. – 301 с.

[2] Дистель Р. *Теория графов* // Новосибирск: Издательство института математики, 2002. – 336 с.

[3] Gera R., Hedetniemi S., Larson C. (Eds.) Graph Theory. Favorite Conjectures and Open Problems – 1. Berlin: Springer, 2016 – 291 p.

[4] Карпов Д. *Теория графов* // СПб.: Изд-во Санкт-Петербургского отделение Математического института им. В. А. Стеклова РАН, 2017. – 482 с.

[5] Gera R., Hedetniemi S., Larson C. (Eds.) Graph Theory. Favorite Conjectures and Open Problems – 2. Berlin: Springer, 2018. – 281 p.

[6] Нидхем М., Ходлер Э. Графовые алгоритмы. Практическая реализация на платформах Apache Spark и Neo4j. М.: ДМК Пресс, 2020. – 258 p.

[7] Булынин А., Мельников Б., Мещанин В., Терентьева Ю. *Оптимизационные задачи, возникающие при проектировании сетей связи высокой размерности, и некоторые эвристические методы их решения* // Информатизация и связь. – 2020. – № 1. – С. 34–40.

[8] Melnikov B., Terentyeva Y. *Building communication networks: on the application of the Kruskal's algorithm in the problems of large dimensions* // В сборнике: IOP Conference Series: Materials Science and Engineering. Krasnoyarsk Science and Technology City Hall. – Krasnoyarsk, Russian Federation. – 2021. – С. 12089.

[9] Мельников Б., Терентьева Ю. *Построение оптимального остовного дерева как инструмент для обеспечения устойчивости сети связи* // Известия высших учебных заведений. Поволжский регион. Технические науки. – 2021. – № 1 (57). – С. 36–45.

[10] Стариков П. *О множестве независимых путей в графе* // Информатизация и связь. – 2021. – № 6. – С. 176–181.

[11] Мельников Б., Сайфуллина Е. *Применение мультиэвристического подхода для случайной генерации графа с заданным вектором степеней* // Известия высших учебных заведений. Поволжский регион. Физико-математические науки. – 2013. – № 3 (27). – С. 70–83.

[12] Мельников Б., Сайфуллина Е. , Терентьева Ю., Чурикова Н. *Применение алгоритмов генерации случайных графов для исследования надёжности сетей связи* // Информатизация и связь. – 2018. – № 1. – С. 71–80.

[13] Мельников Б., Радионов А. *О выборе стратегии в недетерминированных антагонистических играх* // Программирование (Известия РАН). – 1998. – № 5. – С. 55–68.

[14] Мельников Б., Романов Н. *Еще раз об эвристиках для задачи коммивояжера* // Теоретические проблемы информатики и ее приложений. – 2001. – № 4. – С. 81–95.

[15] Мельников Б., Мельникова Е. *Кластеризация ситуаций в алгоритмах реального времени для задач дискретной оптимизации* // Системы управления и информационные технологии. – 2007. – № 2 (28). С. 16–20.

[16] Мельников Б., Эйрих С. *Подход к комбинированию незавершенного метода ветвей и границ и алгоритма имитационной нормализации* // Вестник Воронежского государственного университета. Серия: Системный анализ и информационные технологии. – 2010. – № 1. С. 35–38.

[17] Tarjan R. *A note on finding the bridges of a graph* // Information Processing Letters. – 1974. – Vol. 2. No. 6. – P. 160–161.

[18] Westbrook J., Tarjan R. *Maintaining bridge-connected and biconnected components on-line* // Algorithmica. – 1992. – Vol. 7. No. 5-6. – P. 433–464.

Борис Феликсович МЕЛЬНИКОВ,
главный научный сотрудник Центра информационных технологий и систем органов исполнительной власти
(https://citis.ru/),
email: bf-melnikov@yandex.ru,
mathnet.ru: personid=27967,
elibrary.ru: authorid=15715,
scopus.com: authorId=55954040300,
ORCID: orcidID=0000-0002-6765-6800.

Павел Павлович СТАРИКОВ,
директор Центра информационных технологий и систем органов исполнительной власти
(https://citis.ru/),
email: pstarikov@inevm.ru,
elibrary.ru: authorid=1099967.

Юлия Юрьевна ТЕРЕНТЬЕВА,
ведущий научный сотрудник Центра информационных технологий и систем органов исполнительной власти
(https://citis.ru/),
email: terjul@mail.ru,
elibrary.ru: authorid=1577,
scopus.com: authorId=57216810316,
ORCID: orcidID=0000-0002-2418-003X.

About one problem of analysis of the topology of communication networks

Boris Melnikov, Pavel Starikov, Yulia Terentyeva

***Abstract*—The paper discusses heuristic nonlinear algorithms for checking 2-multiple edge connectivity. This is the term we introduce, a graph has 2-multiple edge connectivity in the case when there are at least 2 edge-independent paths between any two of its vertices; it is important to note that this concept does not coincide with the concept of edge 2-connectivity of a graph, which is more studied in the literature.**

For the last problem, polynomial algorithms have been previously described in the literature, and their small modifications also give algorithms for solving "our" problem. However, we do not consider them: they are quite complex to implement, and we need algorithms in which, at the request of the customer, it is often necessary to make some modifications during its execution.

In the problems considered in this paper, we use such auxiliary algorithms that can be called heuristic; as a result, the full versions of the algorithms under consideration can also be called heuristic. In the paper we consider 3 different groups of algorithms designed to test 2-multiple edge connectivity.

Namely, first we give explicit algorithms for such verification, i.e. algorithms in which the necessary paths are constructed explicitly. Next, in the following algorithm, we check the possible existence of a common cycle for any two vertices of a given graph. The third algorithm is that we check 2-multiple edge connectivity based, firstly, on the absence of bridges and, secondly, on the presence of at least some cycle involving it for any vertex of the graph.

We present a scheme for proving the equivalence of these three algorithms, and after that, a brief description of computational experiments.

***Keywords*—communication networks, graphs, edge connectivity, heuristic algorithms.**

References

[1] Harary F. *Graph theory* // Massachusetts: Addison-Wesley Publ., 1969. – 274 p.

[2] Diestel R. *Graph theory* // Heidelberg: Springer-Verlag, 1997. – 358 p.

[3] Gera R., Hedetniemi S., Larson C. (Eds.) Graph Theory. Favorite Conjectures and Open Problems – 1. Berlin: Springer, 2016 – 291 p.

[4] Karpov D. *Graph theory* // Saint Petersburg: Publishing House of the St. Petersburg Branch Mathematical Institute named after V. A. Steklov of Russian Academy of Sciences, 2017. – 482 p. (in Russian)

[5] Gera R., Hedetniemi S., Larson C. (Eds.) Graph Theory. Favorite Conjectures and Open Problems – 2. Berlin: Springer, 2018. – 281 p.

[6] Needham M., Hodler E. Graph Algorithms. Practical Examples in Apache Spark and Neo4j. Berlin: O'Reilly Media, Inc., 2019 – 300 p.

[7] Bulynin A., Melnikov B., Meshchanin V., Terentyeva Yu. *Optimization problems arising in the design of high-dimensional communication networks and some heuristic methods for their solution* // Informatization and communication. – 2020. – No. 1. – P. 34–40. (in Russian)

[8] Melnikov B., Terentyeva Yu. *Building communication networks: on the application of the Kruskal's algorithm in the problems of large dimensions* // In: IOP Conference Series: Materials Science and Engineering. Krasnoyarsk Science and Technology City Hall. – Krasnoyarsk, Russian Federation. – 2021. – P. 12089.

[9] Melnikov B., Terentyeva Yu. *Building an optimal spanning tree as a tool to ensure the stability of the communication network* // News of higher educational institutions. Volga region. Technical sciences. – 2021. – No. 1 (57). – P. 36–45. (in Russian)

[10] Starikov P. *About the set of independent paths in a graph* // Informatization and communication. – 2021. – No. 6. – P. 176–181. (in Russian)

[11] Melnikov B., Sayfullina E. *Application of a multiheuristic approach for random generation of a graph with a given degree vector* // News of higher educational institutions. Volga region. Physical and mathematical sciences. – 2013. – No. 3 (27). – P. 70–83. (in Russian)

[12] Melnikov B., Sayfullina E. , Terentyeva Yu., Churikova N. *Application of random graph generation algorithms to study the reliability of communication networks* // Informatization and communication. – 2018. – No. 1. – P. 71–80. (in Russian)

[13] Melnikov B., Radionov A. *On the choice of strategy in non-deterministic antagonistic games* // Programming (Russian Academy of Sciences Ed.). – 1998. – No. 5. – P. 55–68. (in Russian)

[14] Melnikov B., Romanov N. *Once again about heuristics for the traveling salesman problem* // Theoretical problems of computer science and its applications. – 2001. – No. 4. – P. 81–95. (in Russian)

[15] Melnikov B., Melnikova E. *Clustering of situations in real-time algorithms for discrete optimization problems* // Control systems and information technologies. – 2007. – No. 2 (28). P. 16–20. (in Russian)

[16] Melnikov B., Eirih S. *An approach to combining an incomplete branch and boundary method and a simulation normalization algorithm* // Bulletin of the Voronezh State University. Series: System Analysis and Information Technology. – 2010. – No. 1. P. 35–38. (in Russian)

[17] Tarjan R. *A note on finding the bridges of a graph* // Information Processing Letters. – 1974. – Vol. 2. No. 6. – P. 160–161.

[18] Westbrook J., Tarjan R. *Maintaining bridge-connected and biconnected components on-line* // Algorithmica. – 1992. – Vol. 7. No. 5-6. – P. 433–464.

Boris MELNIKOV,
Main Researcher of The Center for Information Technologies and Systems for Executive Authorities (https://citis.ru/),
email: bf-melnikov@yandex.ru,
mathnet.ru: personid=27967,
elibrary.ru: authorid=15715,
scopus.com: authorId=55954040300,
ORCID: orcidID=0000-0002-6765-6800.

Pavel STARIKOV,
Director of The Center for Information Technologies and Systems for Executive Authorities (https://citis.ru/),
email: pstarikov@inevm.ru,
elibrary.ru: authorid=1099967.

Yulia TERENTYEVA,
Leading Researcher of The Center for Information Technologies and Systems for Executive Authorities (https://citis.ru/),
email: terjul@mail.ru,
elibrary.ru: authorid=1577,
scopus.com: authorId=57216810316,
ORCID: orcidID=0000-0002-2418-003X.