Теоретические основы информатики

Theoretical foundations of computer science

УДК 519.17;519.85;004.02

УЛУЧШЕННЫЕ ВЕРХНИЕ ОЦЕНКИ В ЗАДАЧЕ ОПТИМАЛЬНОГО РАЗБИЕНИЯ ГРАФА НА КЛИКИ

А. Б. БЕЛЫЙ [1)], ***С. Л. СОБОЛЕВСКИЙ*** [2), 3), 4)], ***А. Н. КУРБАЦКИЙ*** [5)], ***К. РАТТИ*** [4)]

[1)]*СМАРТ-центр, проезд Криэйт, 1, 138602, г. Сингапур, Сингапур*
[2)]*Национальный исследовательский университет ИТМО, пр. Кронверкский, 49, 197101, г. Санкт-Петербург, Россия*
[3)]*Нью-Йоркский университет, ул. Джэй, 370, 11201, г. Нью-Йорк, США*
[4)]*Массачусетский технологический институт, пр. Массачусетс, 77, 02139, г. Кембридж, США*
[5)]*Белорусский государственный университет, пр. Независимости, 4, 220030, г. Минск, Беларусь*

Рассматривается задача нахождения разбиения полного взвешенного графа на клики так, что сумма весов ребер между вершинами, принадлежащими одной клике, максимальна. Данная задача, известная как задача разбиения графа на клики (clique partitioning problem), возникает во многих приложениях и представляет собой вариант

Образец цитирования:
Белый АБ, Соболевский СЛ, Курбацкий АН, Ратти К. Улучшенные верхние оценки в задаче оптимального разбиения графа на клики. *Журнал Белорусского государственного университета. Математика. Информатика.* 2019;3:93–104.
https://doi.org/10.33581/2520-6508-2019-3-93-104

For citation:
Belyi AB, Sobolevsky SL, Kurbatski AN, Ratti C. Improved upper bounds in clique partitioning problem. *Journal of the Belarusian State University. Mathematics and Informatics.* 2019; 3:93–104. Russian.
https://doi.org/10.33581/2520-6508-2019-3-93-104

Авторы:
Александр Борисович Белый – инженер-программист.
Станислав Леонидович Соболевский – доктор физико-математических наук; профессор Института дизайна и урбанистики[2)], ассоциированный профессор практики Центра исследований и развития городов[3)], исследователь[4)].
Александр Николаевич Курбацкий – доктор технических наук, профессор; заведующий кафедрой технологий программирования факультета прикладной математики и информатики.
Карло Ратти – кандидат наук; профессор практики кафедры урбанистических исследований и планирования, директор лаборатории MIT Senseable City Lab.

Authors:
Alexander B. Belyi, software engineer.
alex.bely@smart.mit.edu
http://orcid.org/0000-0001-5650-3182
Stanislav L. Sobolevsky, doctor of science (physics and mathematics); professor at the Institute Design and Urban Science[b], associate professor of practice at the Center for Urban Science and Progress[c], researcher[d].
sobolevsky@nyu.edu
Alexander N. Kurbatski, doctor of science (engineering), full professor; head of the department of software engineering, faculty of applied mathematics and computer science.
kurb@unibel.by
Carlo Ratti, PhD; professor of the practice at the department of urban studies and planning, director of MIT Senseable City Lab.
ratti@mit.edu

классической задачи кластеризации. Она, как и многие другие задачи комбинаторной оптимизации, является NP-трудной, поэтому нахождение ее точного решения зачастую оказывается трудоемким. В данной работе предлагается новый метод построения верхней оценки для функции качества разбиения и показывается, как полученная оценка применяется в методе ветвей и границ при нахождении точного решения. Предлагаемый подход накладывает ограничения на максимально возможное качество разбиения. Новизна метода заключается в возможности использования треугольников, пересекающихся по ребрам, что позволяет находить гораздо более точные оценки, чем при рассмотрении только непересекающихся подграфов. Помимо построения начальной оценки в статье описывается способ ее пересчета при фиксировании ребер на каждом шаге метода ветвей и границ. Приводятся результаты тестирования предлагаемого алгоритма на сгенерированных наборах случайных графов. Показывается, что версия, использующая новые оценки, работает в несколько раз быстрее ранее известных методов.

Ключевые слова: разбиение графа на клики; точное решение; метод ветвей и границ; верхние оценки.

Благодарность. Исследование выполнено при поддержке Национального исследовательского фонда (офис премьер-министра Сингапура) в рамках программы CREATE, Singapore-MIT Alliance for Research and Technology (SMART) Future Urban Mobility (FM) IRG.

IMPROVED UPPER BOUNDS IN CLIQUE PARTITIONING PROBLEM

A. B. BELYI [a], ***S. L. SOBOLEVSKY*** [b, c, d], ***A. N. KURBATSKI*** [e], ***C. RATTI*** [d]

[a] *SMART Centre, 1 Create Way, Singapore 138602, Singapore*
[b] *ITMO University, 49 Kronverksky Avenue, Saint Petersburg 197101, Russia*
[c] *New York University, 370 Jay Street, New York 11201, USA*
[d] *Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge 02139, USA*
[e] *Belarusian State University, 4 Niezaliežnasci Avenue, Minsk 220030, Belarus*

Corresponding author: A. B. Belyi (alex.bely@smart.mit.edu)

In this work, a problem of partitioning a complete weighted graph into cliques in such a way that sum of edge weights between vertices belonging to the same clique is maximal is considered. This problem is known as a clique partitioning problem. It arises in many applications and is a varian of classical clustering problem. However, since the problem, as well as many other combinatorial optimization problems, is NP-hard, finding its exact solution often appears hard. In this work, a new method for constructing upper bounds of partition quality function values is proposed, and it is shown how to use these upper bounds in branch and bound technique for finding an exact solution. Proposed method is based on the usage of triangles constraining maximal possible quality of partition. Novelty of the method lies in possibility of using triangles overlapping by edges, which allows to find much tighter bounds than when using only non-overlapping subgraphs. Apart from constructing initial estimate, a method of its recalculation, when fixing edges on each step of branch and bound method, is described. Test results of proposed algorithm on generated sets of random graphs are provided. It is shown, that version that uses new bounds works several times faster than previously known methods.

Keywords: clique partitioning; branch and bound method; exact solution; upper bounds.

Acknowledgements. This research is supported by the National Research Foundation (prime minister's office, Singapore), under its CREATE programme, Singapore-MIT Alliance for Research and Technology (SMART) Future Urban Mobility (FM) IRG.

Введение

Задача оптимального разбиения полного взвешенного графа на клики возникает во многих приложениях, особенно часто там, где нужно кластеризовать объекты, учитывая только отношения между ними [1]. Ярким примером целого класса таких задач может служить поиск сообществ в комплексных сетях [2–5], который сводится к задаче оптимального разбиения на клики, если осуществлять поиск путем максимизации целевой функции, такой как, например, модулярность [6; 7] или длина кода [8]. Практическая значимость задачи в последнее время вызывает все больший интерес ученых. Однако, поскольку она является NP-трудной [9], большинство исследований сконцентрированы на предложении эвристик, способных относительно быстро находить решения, близкие к оптимальным [10–15]. В то же время алгоритмы, приводящие к точному решению, предлагаются редко и в основном представляют собой вариации метода ветвей и границ [11; 16; 17]. Один из наиболее новых и эффективных

алгоритмов был представлен в работе [17], основной вклад авторов которой заключается в получении нового метода оценки верхней границы значений функции качества разбиения.

В настоящей статье предлагается улучшенный метод, позволяющий находить более эффективные оценки. Его использование приводит к существенному увеличению скорости схождения метода ветвей и границ. Показывается, что новый метод работает в несколько раз быстрее ранее известных подходов.

Задача разбиения графа на клики может быть сформулирована следующим образом [1]. Пусть задан взвешенный полный граф $G = (V, E)$, в котором веса ребер есть вещественные числа как положительные, так и отрицательные, V – множество вершин, $E = \left\{ (i, j, e_{ij}) \middle| i, j \in V, e_{ij} \in \mathbb{R} \right\}$ – множество ребер с весами. Если изначально граф не полный, то отсутствующие ребра могут быть представлены ребрами с весом 0. Далее ребра с положительным либо отрицательным весом будем называть положительными либо отрицательными соответственно. На данном графе вводится функция качества разбиения $Q(C)$, которая для каждого разбиения C множества вершин на кластеры равна сумме весов ребер, соединяющих вершины из одного кластера:

$$Q(C) = \sum_{C_i = C_j} e_{ij},$$

где C_i – номер кластера, в который попадает вершина i; e_{ij} – вес ребра (i, j).

Требуется найти такое разбиение множества вершин V на кластеры, при котором значение Q максимально.

Данная задача часто возникает на практике, особенно там, где нужно разделить объекты на заранее неизвестное число групп. Тогда объектам сопоставляются вершины графа, а некоторой мере схожести объектов соответствуют веса ребер. Примеры таких задач можно найти в биологии [1], в сферах планирования [18] и групповой технологии [16; 19]. При исследовании комплексных сетей важным аспектом является обнаружение в них структуры сообществ [2–5]. Некоторые наиболее популярные подходы основываются на сведении задачи поиска сообществ к задаче разбиения на клики путем замены изначальной сети графом, веса ребер которого задаются специальной функцией, такой как модулярность [6; 7] или длина кода [8]. Таким образом, нахождение точного решения задачи разбиения на клики может быть применено при определении оптимальной с точки зрения конкретной функции качества структуры сообществ [20].

Построение верхней оценки

Очевидной тривиальной верхней оценкой значений функции Q является сумма всех положительных весов ребер:

$$Q_{\text{trivial\_max}} = \sum_{e_{ij} > 0} e_{ij}.$$

Однако на практике данная оценка обычно очень далека от реально достижимого максимума. Для получения более точной оценки можно учесть следующее наблюдение. Рассмотрим тройку вершин (a, b, c), соединенных ребрами с весами e_{ab}, e_{bc}, e_{ac} такими, что $e_{ab} > 0$, $e_{ac} > 0$, $e_{bc} < 0$. Несложно заметить, что при любом разбиении вершин на кластеры либо хотя бы одно положительное ребро не попадет в сумму (т. е. или вершины a и b, или вершины a и c окажутся в разных кластерах), либо отрицательное ребро будет включено в сумму (т. е. вершины b и c окажутся в одном кластере). Такую упорядоченную тройку вершин (a, b, c) с весами ребер $e_{ab} > 0$, $e_{ac} > 0$, $e_{bc} < 0$ будем называть штрафующим треугольником, а величину $p_{abc} = \min(e_{ab}, e_{ac}, -e_{bc})$ – штрафом, поскольку из рассуждений выше следует, что

$$Q(C) \le Q_{\text{trivial\_max}} - p_{abc}$$

для любого разбиения C, или, что то же самое,

$$Q^* \le Q_{\text{trivial\_max}} - p_{abc},$$

где Q^* – максимально возможное значение $Q(C)$, достигаемое при оптимальном C.

Если штрафующие треугольники не пересекаются по ребрам, то каждый из них накладывает свой штраф на максимально возможное значение Q. То есть если S – множество непересекающихся по ребрам штрафующих треугольников, то

$$Q^* \le Q_{\text{trivial\_max}} - \sum_{\{a,b,c\} \in S} p_{abc}.$$

Обозначив $P_G = Q_{\text{trivial\_max}} - Q^*$ минимальный штраф разбиения графа G, предыдущее утверждение можно записать как $\sum_{\{a,b,c\} \in S} p_{abc} \le P_G$. Данное наблюдение было доказано и использовано в работе [17]. Ниже показывается, как, применяя идею из [5], построить более эффективную оценку, учитывая штрафующие треугольники, пересекающиеся по ребрам.

Теорема. *Пусть* (a, b, c) *– штрафующий треугольник в графе* $G = (V, E)$. *Построим новый граф* $G' = (V, E')$ *на вершинах V, вычтя из положительных весов и добавив к отрицательному весу ребер* e_{ab}, e_{ac} *и* e_{bc} *штраф* p_{abc}, *т. е.* $E' = E \setminus \{(a, b, e_{ab}), (a, c, e_{ac}), (b, c, e_{bc})\} \cup \{(a, b, e_{ab} - p_{abc}), (a, c, e_{ac} - p_{abc}), (b, c, e_{bc} + p_{abc})\}$. *Тогда* $Q^* \le Q_{\text{trivial\_max}} - p_{abc} - P_{G'}$ *или* $P_{G'} + p_{abc} \le P_G$.

Доказательство. По определению $P_{G'} = Q'_{\text{trivial\_max}} - Q'^*$, где $Q'_{\text{trivial\_max}}$ – сумма положительных ребер в графе G'; Q'^* – максимальное значение Q при оптимальном разбиении графа G'. Тогда $Q_{\text{trivial\_max}} - p_{abc} - P_{G'} = Q_{\text{trivial\_max}} - p_{abc} - Q'_{\text{trivial\_max}} + Q'^* = Q_{\text{trivial\_max}} - Q'_{\text{trivial\_max}} - p_{abc} + Q'^* = p_{abc} + Q'^* =$ (поскольку $Q_{\text{trivial\_max}}$ равно сумме положительных ребер в графе G, а $Q'_{\text{trivial\_max}}$ – сумме положительных ребер в G' и G и G' отличаются весами только двух положительных ребер) $= e_{ab} + e_{ac} - (e_{ab} - p_{abc}) - (e_{ac} - p_{abc}) - p_{abc} + Q'^* = p_{abc} + Q'^*$. Остается показать, что $Q^* - Q'^* \le p_{abc}$.

Пусть C – оптимальное разбиение, при котором достигается $Q^* = Q(C)$. Рассмотрим то же самое разбиение графа G' и положим для него $Q = Q'(C)$. По определению $Q'(C) \le Q'^*$, т. е. достаточно показать, что $Q(C) - Q'(C) \le p_{abc}$. Поскольку все ребра в G и G', кроме трех, имеют одинаковые веса, то разность в левой части неравенства зависит только от того, какие из весов ребер e_{ab}, e_{ac} и e_{bc} включены в суммы. Возможны три случая:

1) все три вершины a, b и c принадлежат разным кластерам. Тогда ни одно ребро не включено в суммы и выполняется

$$Q(C) - Q'(C) = 0 \le p_{abc};$$

2) вершины a, b и c принадлежат одному кластеру. Тогда все три ребра включены в суммы и выполняется

$$Q(C) - Q'(C) = e_{ab} + e_{ac} + e_{bc} - (e_{ab} - p_{abc}) - (e_{ac} - p_{abc}) - (e_{bc} + p_{abc}) = p_{abc};$$

3) две вершины из a, b и c принадлежат одному кластеру. Тогда ровно одно ребро включено в суммы и либо $Q(C) - Q'(C) = e_{ab} - (e_{ab} - p_{abc}) = p_{abc}$, либо $Q(C) - Q'(C) = e_{ac} - (e_{ac} - p_{abc}) = p_{abc}$, либо $Q(C) - Q'(C) = e_{bc} - (e_{bc} + p_{abc}) = -p_{abc}$. Следовательно, в любом случае утверждение теоремы выполнено и она доказана.

Таким образом, чтобы построить оценку $Q_{\max} \ge Q^*$, можно последовательно находить штрафующие треугольники и вычитать их штраф из весов ребер, пока в графе не останется штрафующих треугольников. Сумма полученных штрафов $\sum p_{abc}$ будет оценкой снизу для P_G, и тогда $\sum p_{abc} \le P_G = Q_{\text{trivial\_max}} - Q^*$, $Q^* \le Q_{\text{trivial\_max}} - \sum p_{abc} = Q_{\max}$.

Нахождение точного решения

Применяя полученную оценку, для определения точного решения можно использовать метод ветвей и границ. На первом шаге найдем какое-нибудь разбиение, желательно со значением Q, близким к максимуму. Эта величина будет начальной нижней оценкой $Q_{\min}$ достижимого значения Q. Далее построим набор штрафующих треугольников и начальную оценку $Q_{\max}$. Затем на каждой итерации будем рассматривать очередное ребро и два возможных разбиения: разбиение, в котором данное ребро зафиксировано как внутрикластерное, т. е. концы ребра попадают в один кластер, и разбиение, в котором данное ребро зафиксировано как межкластерное, т. е. концы ребра лежат в разных кластерах. Для каждого разбиения будем пересчитывать оценку $Q_{\max}$ с учетом нового ограничения. Если на очередном шаге оценка $Q_{\max}$ меньше уже достигнутого значения $Q_{\min}$, то при текущих ограничениях получить решение, лучшее уже найденного, невозможно, и данная ветвь отсекается. Иначе рекурсивно рассматривается очередное ребро. Ниже опишем каждый шаг более детально.

Начальное разбиение и оценку $Q_{\min}$ можно найти с помощью какой-либо из множества эвристик, описанных в литературе. В данной работе используется алгоритм Combo [15], как один из наиболее точных и быстрых методов со свободно доступным исходным кодом, позволяющим достичь наилучших (по сравнению с другими подходами) значений целевой функции.

Для построения начального набора штрафующих треугольников применяется жадный алгоритм. На каждой итерации выбирается треугольник с наименьшим (наибольшим по модулю) весом отрицательного ребра. Веса ребер соответствующего треугольника корректируются с учетом полученного штрафа, и данный процесс повторяется, пока в графе есть штрафующие треугольники. Описанный процесс может быть представлен в виде нижеприведенного алгоритма 1. Далее при описании всех алгоритмов подразумевается, что они имеют доступ к графу $G = (V, E)$.

Алгоритм 1. Построение набора штрафующих треугольников *S*

Выход: набор штрафующих треугольников *S*.

1. Изначально набор *S* пуст.
2. Найти все отрицательные ребра.
3. Упорядочить их по возрастанию веса.
4. Для каждого ребра (b, c, e_{bc}) из полученного списка:
   для каждой вершины *a*, неинцидентной ребру:
   если $e_{ab} > 0$ и $e_{ac} > 0$:
   добавить треугольник (a, b, c) в *S*;
   // обновить веса ребер треугольника (a, b, c) с учетом штрафа:
   $p_{abc} = \min(e_{ab}, e_{ac}, -e_{bc})$;
   $e_{ab} = e_{ab} - p_{abc}$;
   $e_{ac} = e_{ac} - p_{abc}$;
   $e_{bc} = e_{bc} + p_{abc}$.
5. Вернуть набор *S*.

Отрицательные ребра можно найти путем простого перебора всех ребер за $O(n^2)$ итераций, где $n = |V|$ – число вершин. На шаге 4 алгоритма 1 рассматриваются некоторые тройки вершин ровно один раз. Так как всего троек вершин порядка n^3, то алгоритм 1 выполняется за время $O(n^3)$.

На каждой итерации метода ветвей и границ очередное незафиксированное ребро объявляется либо внутрикластерным, либо межкластерным. В каждом случае нужно обновить статус остальных ребер, чтобы выполнялось условие транзитивности, т. е. если ребра (a, b) и (b, c) внутрикластерные, то таковым должно быть и ребро (a, c), и если ребро (a, b) внутрикластерное, а ребро (b, c) межкластерное, то ребро (a, c) также должно быть межкластерным. Это может быть сделано с помощью следующего алгоритма 2.

Алгоритм 2. Обновление зафиксированных ребер

Вход: зафиксированное ребро (a, b).

Выход: набор зафиксированных ребер.

1. Найти вершины, принадлежащие кластеру вершины *a* (множество $A = \{i \mid C_i = C_a\}$), кластеру вершины *b* (множество $B = \{i \mid C_i = C_b\}$); вершины, про которые уже известно, что они принадлежат кластеру, отличному от кластера вершины *a* (множество $X = \{i \mid C_i \neq C_a\}$); вершины, про которые уже известно, что они принадлежат кластеру, отличному от кластера вершины *b* (множество $Y = \{i \mid C_i \neq C_b\}$).
2. Если ребро (a, b) зафиксировано как внутрикластерное, то таковыми же объявить все ребра, соединяющие вершины множеств *A* и *B*, а все ребра, соединяющие *A* и *Y*, а также *B* и *X*, объявить межкластерными;
   иначе все ребра, соединяющие *A* и *B*, объявить межкластерными.
3. Возвратить все вновь зафиксированные ребра.

Первый шаг может быть выполнен путем рассмотрения всех вершин графа, т. е. за $O(n)$ итераций. На втором шаге обновляется статус некоторых ребер, которых всего не более n^2. Таким образом, сложность алгоритма 2 есть $O(n^2)$.

После обновления статуса ребер нужно обновить текущее значение оценки сверху $Q_{\max}$. Пусть (a, b, c) – штрафующий треугольник. Если на текущем шаге положительное ребро фиксируется как

межкластерное или внутрикластерным объявляется отрицательное ребро, то абсолютное значение его веса добавляется к штрафу и все штрафующие треугольники, включающие данное ребро, перестают учитываться. Если же внутрикластерным объявляется положительное ребро или межкластерным объявляется отрицательное, то штраф треугольника обновляется: если (a, b) фиксируется внутрикластерным, то $p_{abc} = \min(e_{ac}, -e_{bc})$, если (a, c) фиксируется внутрикластерным, то $p_{abc} = \min(e_{ab}, -e_{bc})$, если (b, c) фиксируется межкластерным, то $p_{abc} = \min(e_{ab}, e_{ac})$. Таким образом, для обновления множества штрафующих треугольников и пересчета $Q_{\max}$ можно использовать алгоритм 3.

Алгоритм 3. Обновление $Q_{\max}$ и S

Вход: набор штрафующих треугольников S.

Выход: обновленный S и новая $Q_{\max}$.

1. $Q_{\max} = Q_{\text{trivial\_max}}$.
2. Для каждого зафиксированного ребра (a, b):

   если $e_{ab} > 0$ и (a, b) является межкластерным, то

   $Q_{\max} = Q_{\max} - e_{ab}$;

   если $e_{ab} < 0$ и (a, b) является внутрикластерным, то

   $Q_{\max} = Q_{\max} + e_{ab}$.
3. Для каждого треугольника (a, b, c) из S:

   если межкластерным является (a, b) или (a, c) либо (b, c) есть внутрикластерное, то

   исключить (a, b, c) из S;

   $p_{abc} = 0$;

   иначе, если (a, b) является внутрикластерным, то

   $p_{abc} = \min(e_{ac}, -e_{bc})$;

   $e_{ac} = e_{ac} - p_{abc}$;

   $e_{bc} = e_{bc} + p_{abc}$;

   иначе, если (a, c) является внутрикластерным, то

   $p_{abc} = \min(e_{ab}, -e_{bc})$;

   $e_{ab} = e_{ab} - p_{abc}$;

   $e_{bc} = e_{bc} + p_{abc}$;

   иначе, если (b, c) является межкластерным, то

   $p_{abc} = \min(e_{ab}, e_{ac})$;

   $e_{ab} = e_{ab} - p_{abc}$;

   $e_{ac} = e_{ac} - p_{abc}$;

   иначе:

   $p_{abc} = \min(e_{ab}, e_{ac}, -e_{bc})$;

   $e_{ab} = e_{ab} - p_{abc}$;

   $e_{ac} = e_{ac} - p_{abc}$;

   $e_{bc} = e_{bc} + p_{abc}$;

   $Q_{\max} = Q_{\max} - p_{abc}$.
4. Пока в графе есть штрафующие треугольники (a, b, c):

   $p_{abc} = \min(e_{ab}, e_{ac}, -e_{bc})$;

   $e_{ab} = e_{ab} - p_{abc}$;

   $e_{ac} = e_{ac} - p_{abc}$;

   $e_{bc} = e_{bc} + p_{abc}$;

   $Q_{\max} = Q_{\max} - p_{abc}$.
5. Возвратить обновленный S и $Q_{\max}$.

Добавление в S каждого штрафующего треугольника обнуляет как минимум одно ребро в графе. Таким образом, штрафующих треугольников не более n^2, и шаги 2 и 3 выполняются за $O(n^2)$ итераций. Шаг 4 может быть выполнен рассмотрением всех треугольников, т. е. за $O(n^3)$ итераций. Таким образом, сложность алгоритма 3 есть $O(n^3)$.

В методе ветвей и границ важным аспектом является очередность, в которой происходит ветвление. В работе [17] показано, что выбирать очередное ребро нужно таким образом, чтобы при объявлении его межкластерным значение $Q_{\max}$ максимально уменьшалось. В нашем случае получение точных значений изменения $Q_{\max}$ на каждом шаге требует вызова алгоритма 3, что слишком трудоемко. Чтобы найти оценки изменения $Q_{\max}$, для каждого положительного ребра рассмотрим, как $Q_{\max}$ меняется при объявлении межкластерным только этого ребра. И в дальнейшем ребра рассматриваются в порядке убывания величины вычисленного изменения.

Алгоритм 4. Упорядочивание ребер

Вход: набор штрафующих треугольников S, дающий оценку $Q_{\max}$.

Выход: упорядоченный список ребер.

1. Для каждого положительного ребра (a, b):
   объявить ребро (a, b) межкластерным;
   вычислить новую $Q'_{\max}$ с помощью алгоритма 3;
   ребру (a, b) поставить в соответствие значение $Q_{\max} - Q'_{\max}$.
2. Отсортировать ребра в порядке убывания соответствующих им значений.
3. Вернуть упорядоченный список.

Алгоритм 4 будет вызываться только один раз в начале работы метода ветвей и границ. Так как положительных ребер может быть порядка n^2 и производится их сортировка, то алгоритм 4 выполняется за время $O\left(n^2 \log(n)\right)$.

Далее представлен рекурсивный алгоритм обхода в глубину дерева поиска метода ветвей и границ.

Алгоритм 5. Рекурсивный обход в глубину дерева поиска

Вход: список ребер для рассмотрения, индекс текущего ребра, набор штрафующих треугольников S, значение $Q_{\min}$.

Выход: значение $Q_{\min}$ и сохраненное оптимальное разбиение.

1. Если индекс выходит за рамки списка ребер:
   // значит, все положительные ребра зафиксированы
   // и полученное разбиение является новым наилучшим разбиением.
   Незафиксированные отрицательные ребра объявить межкластерными.
   Запомнить разбиение.
   Обновить $Q_{\min} = Q$.
   Возвратить $Q_{\min}$.
2. Выбрать из списка ребро с заданным индексом.
3. Пока выбранное ребро зафиксировано:
   выбирать следующее ребро и увеличивать индекс.
4. Повторить для {выбранное ребро объявляется внутрикластерным, выбранное ребро объявляется межкластерным}:
   с помощью алгоритма 2 найти и зафиксировать остальные ребра, которые нужно зафиксировать;
   с помощью алгоритма 3 найти новый набор штрафующих треугольников S' и новое значение $Q_{\max}$.
   Если $Q_{\max} > Q_{\min}$, то
      обновить значение $Q_{\min}$ значением, полученным рекурсивным вызовом алгоритма 5 со следующим индексом, с набором S' и текущим значением $Q_{\min}$.
   Возвратить в исходное состояние ребра, зафиксированные в начале шага 4.
5. Возвратить $Q_{\min}$.

Окончательный алгоритм нахождения оптимального разбиения графа на клики может быть представлен следующим алгоритмом 6.

Алгоритм 6. Нахождение точного решения задачи разбиения графа на клики

1. Используя алгоритм Combo, построить начальное разбиение и оценку $Q_{\min}$.
2. По алгоритму 1 найти начальный набор штрафующих треугольников S и оценку $Q_{\max}$.
3. С помощью алгоритма 4 упорядочить ребра.
4. Вызвать алгоритм 5 со списком, полученным на шаге 3, индексом 1, набором штрафующих треугольников S и значением $Q_{\min}$.
5. Возвратить оптимальное разбиение и значение $Q_{\min}$.

Данный алгоритм рассматривает все возможные комбинации, запоминает оптимальное разбиение и наибольшее достижимое значение $Q_{\min}$.

Вычислительный эксперимент и анализ результатов

Предлагаемый алгоритм реализован на языке C++. Для его тестирования использовались наборы искусственных графов, предложенные в [17]. Первый набор состоит из графов на n вершинах, веса ребер которых выбирались случайным образом из равномерного распределения на отрезке $[-q, q]$. Для возможности сравнения с результатами из работы [17] применялась аналогичная процедура: для каждого n от 10 до 20 и каждого q из набора $\{1, 2, 3, 5, 10, 50, 100\}$ генерировалось 5 случайных графов.

В табл. 1 приведены результаты работы предлагаемого алгоритма, а также результаты из [17] для сравнения. Каждое значение в табл. 1 равно сумме соответствующих значений для 35 случайных графов. *Nodes* обозначает количество рассмотренных вершин в дереве поиска решения, t – время выполнения в секундах, $Q_{\text{init\_min}}$ и $Q_{\text{init\_max}}$ – начальные оценки $Q_{\min}$ и $Q_{\max}$, полученные на шагах 1 и 2 алгоритма 6 соответственно. Значения $Q_{\text{trivial\_max}}$, $Q_{\text{init\_max}}$, $Q_{\text{init\_min}}$ нормализованы относительно оптимального решения Q^*.

Таблица 1

Результаты на первом наборе случайных графов

Table 1

Results for the first set of random graphs

n	$Q^a_{\text{trivial\_max}}$	$Q^a_{\text{init\_max}}$	$Q^a_{\text{init\_min}}$	$Q^b_{\text{trivial\_max}}$	$Q^b_{\text{init\_max}}$	$Q^b_{\text{init\_min}}$	$Nodes^a$	$Nodes^b$	t^a	t^b
10	1,720	**1,099**	**1,000**	1,764	1,226	0,994	**362**	1205	**0,05**	0,05
11	1,819	**1,119**	**0,998**	1,831	1,272	0,988	**877**	4236	**0,10**	0,13
12	1,758	**1,120**	**0,994**	1,932	1,305	0,993	**1401**	7577	**0,16**	0,18
13	2,014	**1,191**	**0,990**	1,867	1,287	0,986	**3723**	20 005	**0,42**	0,47
14	2,006	**1,199**	**0,991**	1,971	1,355	0,983	**6590**	50 101	**0,77**	1,28
15	2,065	**1,223**	0,995	2,071	1,367	**0,996**	**10 776**	185 336	**1,58**	5,26
16	2,119	**1,261**	0,991	2,043	1,341	**0,999**	**43 150**	499 569	**6,35**	16,3
17	2,154	**1,229**	0,994	2,189	1,419	**0,997**	**63 916**	4 186 427	**10,26**	155
18	2,226	**1,305**	0,988	2,230	1,433	**0,993**	**199 302**	9 811 533	**39,84**	466
19	2,183	**1,264**	0,986	2,236	1,439	**0,994**	**479 192**	37 572 347	**101,08**	1849
20	2,313	**1,314**	**0,989**	2,251	1,440	0,988	**918 177**	185 321 420	**259,61**	11 299

Примечание. Здесь и в табл. 2–4 обозначены результаты: ^{a} – предлагаемого алгоритма; ^{b} – алгоритма из [17]. Полужирным шрифтом выделены лучшие результаты.

Графы второго набора получены в итоге следующей процедуры. Для каждого графа из n вершин фиксировался параметр p. Затем для каждой вершины строился бинарный вектор длины p, у которого значения 0 или 1 выбирались равновероятно. Вес ребра между вершинами i и j полагался равным p минус удвоенное количество позиций, в которых векторы i и j отличаются. Как и в первом наборе, в целях удобного сравнения с результатами [17] для каждого значения n от 10 до 24 и p из набора $\{1, 2, 3, 5, 10, 50, 100\}$ генерировалось 5 случайных графов (табл. 2). Каждое значение в табл. 2 равно сумме соответствующих значений для 35 реализаций случайных графов.

Таблица 2

Результаты на втором наборе случайных графов

Table 2

Results for the second set of random graphs

n	$Q^a_{\text{trivial\_max}}$	$Q^a_{\text{init\_max}}$	$Q^a_{\text{init\_min}}$	$Q^b_{\text{trivial\_max}}$	$Q^b_{\text{init\_max}}$	$Q^b_{\text{init\_min}}$	$Nodes^a$	$Nodes^b$	t^a	t^b
10	1,370	**1,057**	**0,998**	1,387	1,122	0,985	**167**	488	**0,03**	0,04
11	1,427	**1,089**	**0,998**	1,476	1,177	0,995	**376**	962	**0,05**	0,05

Окончание табл. 2
Ending table 2

n	$Q^a_{\text{trivial\_max}}$	$Q^a_{\text{init\_max}}$	$Q^a_{\text{init\_min}}$	$Q^b_{\text{trivial\_max}}$	$Q^b_{\text{init\_max}}$	$Q^b_{\text{init\_min}}$	$Nodes^a$	$Nodes^b$	t^a	t^b
12	1,466	**1,064**	0,994	1,421	1,149	**1,000**	**388**	972	**0,07**	0,05
13	1,480	**1,092**	0,992	1,437	1,144	**0,997**	**791**	2178	**0,11**	0,08
14	1,522	**1,102**	**0,994**	1,516	1,173	0,991	**1597**	6158	**0,25**	0,19
15	1,523	**1,116**	0,993	1,546	1,178	**0,995**	**2134**	7819	**0,38**	0,22
16	1,560	**1,126**	0,988	1,541	1,181	**0,992**	**4215**	21 752	**0,78**	0,71
17	1,560	**1,113**	**0,996**	1,569	1,188	0,988	**5277**	138 305	**1,08**	5,08
18	1,553	**1,115**	**0,996**	1,575	1,195	0,992	**8532**	160 195	**2,17**	6,52
19	1,617	**1,148**	**0,997**	1,592	1,214	0,987	**34 968**	1 389 759	**11,14**	66,4
20	1,602	**1,131**	**0,991**	1,630	1,228	0,986	**38 270**	2 598 775	**12,56**	136
21	1,640	**1,148**	**0,997**	1,631	1,229	0,983	**38 993**	11 977 231	**14,87**	741
22	1,708	**1,175**	**0,994**	1,639	1,232	0,990	**160 856**	14 413 288	**71,73**	962
23	1,691	**1,172**	**0,993**	1,632	1,218	0,992	**295 452**	25 313 750	**134,63**	1805
24	1,724	**1,192**	**0,995**	1,728	1,269	0,984	**1 615 930**	778 958 420	**782,56**	67 034

Третий набор состоит из графов, первый шаг построения которых совпадал с процедурой генерации графов первого набора, но на втором шаге вес каждого ребра мог быть обнулен с заданной вероятностью. Было сгенерировано два поднабора, в первом вероятность обнуления веса ребра составляла 40 %, во втором – 80 % (табл. 3 и 4).

Таблица 3

Результаты на третьем наборе случайных графов с вероятностью обнуления веса ребра 40 %

Table 3

Results for the third set of random graphs with 40 % probability of changing an edge weight to zero

n	$Q^a_{\text{trivial\_max}}$	$Q^a_{\text{init\_max}}$	$Q^a_{\text{init\_min}}$	$Q^b_{\text{trivial\_max}}$	$Q^b_{\text{init\_max}}$	$Q^b_{\text{init\_min}}$	$Nodes^a$	$Nodes^b$	t^a	t^b
10	1,327	**1,076**	**0,998**	1,468	1,153	0,987	**254**	388	0,03	**0,01**
11	1,374	**1,085**	**0,988**	1,494	1,173	0,985	**451**	810	0,04	**0,01**
12	1,489	**1,117**	**0,996**	1,498	1,167	0,983	**596**	2442	0,07	**0,04**
13	1,517	**1,142**	**0,990**	1,513	1,192	0,988	**1269**	5128	0,12	**0,09**
14	1,566	**1,158**	**0,993**	1,492	1,184	0,990	**2422**	4836	0,21	**0,08**
15	1,638	**1,182**	**0,994**	1,616	1,243	0,983	**3783**	25 647	**0,41**	0,54
16	1,703	**1,180**	**0,988**	1,696	1,267	0,986	**10 789**	54 728	**1,21**	1,38
17	1,699	**1,181**	**0,990**	1,750	1,307	0,975	**13 918**	140 765	**1,79**	3,93
18	1,736	**1,201**	**0,988**	1,699	1,263	0,974	**46 884**	382 507	**7,00**	11,59
19	1,736	**1,199**	**0,988**	1,800	1,315	0,984	**56 757**	1 469 527	**8,85**	55,69
20	1,864	**1,231**	**0,986**	1,850	1,326	0,985	**122 830**	3 195 924	**25,30**	114

Таблица 4

Результаты на третьем наборе случайных графов с вероятностью обнуления веса ребра 80 %

Table 4

Results for the third set of random graphs with 80 % probability of changing an edge weight to zero

n	$Q^a_{\text{trivial\_max}}$	$Q^a_{\text{init\_max}}$	$Q^a_{\text{init\_min}}$	$Q^b_{\text{trivial\_max}}$	$Q^b_{\text{init\_max}}$	$Q^b_{\text{init\_min}}$	$Nodes^a$	$Nodes^b$	t^a	t^b
10	1,064	1,040	**1,000**	1,060	**1,037**	0,992	51	**30**	0,01	**0**
11	1,061	1,013	0,987	1,067	1,013	**1,000**	47	**14**	0,01	**0**
12	1,079	1,015	0,984	1,050	**1,006**	**0,995**	82	**55**	0,01	**0**
13	1,093	1,043	**0,990**	1,088	**1,023**	0,977	**149**	167	0,01	0,01
14	1,074	1,029	0,977	1,048	**1,019**	**0,999**	176	**71**	0,02	**0**
15	1,101	**1,042**	0,974	1,110	1,059	**0,982**	**228**	472	0,02	**0,01**
16	1,133	**1,041**	0,978	1,135	1,062	**0,979**	**243**	720	0,02	**0,01**
17	1,174	**1,055**	**0,996**	1,114	1,080	0,989	**642**	789	0,04	**0,02**
18	1,159	**1,078**	0,981	1,143	1,082	**0,985**	**650**	979	0,05	**0,02**
19	1,148	**1,068**	**0,987**	1,195	1,108	0,985	**891**	2601	0,08	**0,06**
20	1,164	1,098	0,982	1,147	**1,072**	**0,986**	**2366**	2423	0,20	**0,05**

Тестирование выполнялось на компьютере с оперативной памятью 8 Гб и процессором Intel Core i7 частотой 2,3 ГГц под управлением MacOS 10.13.6. Как видно из табл. 1–3, предложенный алгоритм работает быстрее, чем алгоритм из [17], который, в свою очередь, как показали его авторы, выполняется быстрее, чем другие известные алгоритмы. Для графов до 20 вершин, в которых большая часть ребер имеет вес 0, описанный алгоритм работает медленнее, чем алгоритм из [17], однако оба алгоритма осуществляются за доли секунды, и с ростом размера графа предлагаемый алгоритм рассматривает меньшее число вариантов (см. табл. 4). Отметим, что представленные для сравнения результаты из [17] получены на компьютере с частотой процессора 1,86 ГГц, в то время как тестирование предлагаемого алгоритма проводилось на несколько более быстром компьютере. Кроме того, данные сравниваются на разных реализациях случайных графов, сгенерированных по одним и тем же правилам. Однако увеличение скорости работы явно не пропорционально увеличению производительности компьютера и в большей степени вызвано улучшением эффективности алгоритма. Убедиться в этом можно, например учитывая тот факт, что пересчет значения $Q_{\max}$ на каждой итерации в обоих алгоритмах выполняется за время $O\left(n^3\right)$, при этом по предлагаемому алгоритму рассматривается гораздо меньше вершин дерева поиска, чем по алгоритму из [17].

Заключение

Предлагаемый алгоритм может быть применен в целях более быстрого нахождения точного решения задачи оптимального разбиения графа на клики. Как и в ранее известном методе, рассмотрение каждой вершины дерева поиска метода ветвей и границ выполняется за время $O\left(n^3\right)$, однако константа в сложности пересчета в описанном алгоритме оказывается несколько выше, что иногда ведет к незначительному проигрыванию по времени. Но данный эффект заметен только на очень простых графах, с которыми оба алгоритма справляются за доли секунды. Для более сложных графов ускорение за счет предложенных в данной работе более точных верхних оценок качества разбиения оказывается существенным. По сравнению с ранее известным методом описанный алгоритм позволяет получить ускорение по времени более чем в 85 раз на некоторых типах графов (с 18,5 ч до 13 мин). При этом сокращение числа рассматриваемых вершин дерева поиска метода ветвей и границ может быть более чем в 480 раз.

Библиографические ссылки

1. Grötschel M, Wakabayashi Y. A cutting plane algorithm for a clustering problem. *Mathematical Programming. Series B*. 1989; 45(1–3):59–96. DOI: 10.1007/BF01589097.

2. Fortunato S. Community detection in graphs. *Physics reports*. 2010;486(3–5):75–174. DOI: 10.1016/j.physrep.2009.11.002.

3. Belyi A, Bojic I, Sobolevsky S, Sitko I, Hawelka B, Rudikova L, et al. Global multi-layer network of human mobility. *International Journal of Geographical Information Science*. 2017;31(7):1381–1402. DOI: 10.1080/13658816.2017.1301455.

4. Belyi A, Bojic I, Sobolevsky S, Rudikova L, Kurbatski A, Ratti C. Community structure of the world revealed by Flickr data. В: *Технологии информатизации и управления. ТИМ-2016. Материалы III Международной научно-практической конференции; 14–15 апреля 2016 г.; Гродно, Беларусь*. Гродно: ГрГУ; 2016. с. 1–9.

5. Sobolevsky S, Belyi A, Ratti C. Optimality of community structure in complex networks. arXiv:1712.05110 [Preprint]. 2017 [cited 2019 March 22]: [17 p.]. Available from: https://arxiv.org/abs/1712.05110.

6. Newman ME, Girvan M. Finding and evaluating community structure in networks. *Physical Review E*. 2004;69(2):026113. DOI: 10.1103/PhysRevE.69.026113.

7. Newman ME. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences of the United States of America*. 2006;103(23):8577–8582. DOI: 10.1073/pnas.0601602103.

8. Rosvall M, Bergstrom CT. Maps of random walks on complex networks reveal community structure. *Sciences of the United States of America*. 2008;105(4):1118–1123. DOI: 10.1073/pnas.0706851105.

9. Wakabayashi Y. *Aggregation of binary relations: algorithmic and polyhedral investigations*. Augsburg: University of Augsburg; 1986. 191 p.

10. De Amorim SG, Barthélemy JP, Ribeiro CC. Clustering and clique partitioning: simulated annealing and tabu search approaches. *Journal of Classification*. 1992;9(1):17–41. DOI: 10.1007/BF02618466.

11. Dorndorf U, Pesch E. Fast clustering algorithms. *ORSA Journal on Computing*. 1994;6(2):141–153. DOI: 10.1287/ijoc.6.2.141.

12. Charon I, Hudry O. Noising methods for a clique partitioning problem. *Discrete Applied Mathematics*. 2006;154(5):754–769. DOI: 10.1016/j.dam.2005.05.029.

13. Zhou Y, Hao JK, Goëffon A. A three-phased local search approach for the clique partitioning problem. *Journal of Combinatorial Optimization*. 2016;32(2):469–491. DOI: 10.1007/s10878-015-9964-9.

14. Brimberg J, Janićijević S, Mladenović N, Urošević D. Solving the clique partitioning problem as a maximally diverse grouping problem. *Optimization Letters*. 2017;11(6):1123–1135. DOI: 10.1007/s11590-015-0869-4.

15. Sobolevsky S, Campari R, Belyi A, Ratti C. General optimization technique for high-quality community detection in complex networks. *Physical Review E*. 2014;90(1):012811. DOI: 10.1103/PhysRevE.90.012811.

16. Oosten M, Rutten JHGC, Spieksma FCR. The clique partitioning problem: facets and patching facets. *Networks: An International Journal*. 2001;38(4):209–226. DOI: 10.1002/net.10004.

17. Jaehn F, Pesch E. New bounds and constraint propagation techniques for the clique partitioning problem. *Discrete Applied Mathematics*. 2013;161(13–14):2025–2037. DOI: 10.1016/j.dam.2013.02.011.

18. Dorndorf U, Jaehn F, Pesch E. Modelling robust flight-gate scheduling as a clique partitioning problem. *Transportation Science*. 2008;42(3):292–301. DOI: 10.1287/trsc.1070.0211.

19. Wang H, Alidaee B, Glover F, Kochenberger G. Solving group technology problems via clique partitioning. *International Journal of Flexible Manufacturing Systems*. 2006;18(2):77–97. DOI: 10.1007/s10696-006-9011-3.

20. Aloise D, Cafieri S, Caporossi G, Hansen P, Perron S, Liberti L. Column generation algorithms for exact modularity maximization in networks. *Physical Review E*. 2010;82(4):046112. DOI: 10.1103/PhysRevE.82.046112.

References

1. Grötschel M, Wakabayashi Y. A cutting plane algorithm for a clustering problem. *Mathematical Programming. Series B*. 1989; 45(1–3):59–96. DOI: 10.1007/BF01589097.

2. Fortunato S. Community detection in graphs. *Physics reports*. 2010;486(3–5):75–174. DOI: 10.1016/j.physrep.2009.11.002.

3. Belyi A, Bojic I, Sobolevsky S, Sitko I, Hawelka B, Rudikova L, et al. Global multi-layer network of human mobility. *International Journal of Geographical Information Science*. 2017;31(7):1381–1402. DOI: 10.1080/13658816.2017.1301455.

4. Belyi A, Bojic I, Sobolevsky S, Rudikova L, Kurbatski A, Ratti C. Community structure of the world revealed by Flickr data. In: *Tekhnologii informatizatsii i upravleniya. TIM-2016. Materialy III Mezhdunarodnoi nauchno-prakticheskoi konferentsii; 14–15 aprelya 2016 g.; Grodno, Belarus'* [Technologies of Information and Management TIM-2016. Materials of the 3rd International science and training conference; 2016 April 14–15; Grodno, Belarus]. Grodno: Yanka Kupala State University of Grodno; 2016. p. 1–9.

5. Sobolevsky S, Belyi A, Ratti C. Optimality of community structure in complex networks. arXiv:1712.05110 [Preprint]. 2017 [cited 2019 March 22]: [17 p.]. Available from: https://arxiv.org/abs/1712.05110.

6. Newman ME, Girvan M. Finding and evaluating community structure in networks. *Physical Review E*. 2004;69(2):026113. DOI: 10.1103/PhysRevE.69.026113.

7. Newman ME. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences of the United States of America*. 2006;103(23):8577–8582. DOI: 10.1073/pnas.0601602103.

8. Rosvall M, Bergstrom CT. Maps of random walks on complex networks reveal community structure. *Sciences of the United States of America*. 2008;105(4):1118–1123. DOI: 10.1073/pnas.0706851105.

9. Wakabayashi Y. *Aggregation of binary relations: algorithmic and polyhedral investigations*. Augsburg: University of Augsburg; 1986. 191 p.

10. De Amorim SG, Barthélemy JP, Ribeiro CC. Clustering and clique partitioning: simulated annealing and tabu search approaches. *Journal of Classification*. 1992;9(1):17–41. DOI: 10.1007/BF02618466.

11. Dorndorf U, Pesch E. Fast clustering algorithms. *ORSA Journal on Computing*. 1994;6(2):141–153. DOI: 10.1287/ijoc.6.2.141.

12. Charon I, Hudry O. Noising methods for a clique partitioning problem. *Discrete Applied Mathematics*. 2006;154(5):754–769. DOI: 10.1016/j.dam.2005.05.029.

13. Zhou Y, Hao JK, Goëffon A. A three-phased local search approach for the clique partitioning problem. *Journal of Combinatorial Optimization.* 2016;32(2):469–491. DOI: 10.1007/s10878-015-9964-9.

14. Brimberg J, Janićijević S, Mladenović N, Urošević D. Solving the clique partitioning problem as a maximally diverse grouping problem. *Optimization Letters.* 2017;11(6):1123–1135. DOI: 10.1007/s11590-015-0869-4.

15. Sobolevsky S, Campari R, Belyi A, Ratti C. General optimization technique for high-quality community detection in complex networks. *Physical Review E.* 2014;90(1):012811. DOI: 10.1103/PhysRevE.90.012811.

16. Oosten M, Rutten JHGC, Spieksma FCR. The clique partitioning problem: facets and patching facets. *Networks: An International Journal.* 2001;38(4):209–226. DOI: 10.1002/net.10004.

17. Jaehn F, Pesch E. New bounds and constraint propagation techniques for the clique partitioning problem. *Discrete Applied Mathematics.* 2013;161(13–14):2025–2037. DOI: 10.1016/j.dam.2013.02.011.

18. Dorndorf U, Jaehn F, Pesch E. Modelling robust flight-gate scheduling as a clique partitioning problem. *Transportation Science.* 2008;42(3):292–301. DOI: 10.1287/trsc.1070.0211.

19. Wang H, Alidaee B, Glover F, Kochenberger G. Solving group technology problems via clique partitioning. *International Journal of Flexible Manufacturing Systems*. 2006;18(2):77–97. DOI: 10.1007/s10696-006-9011-3.

20. Aloise D, Cafieri S, Caporossi G, Hansen P, Perron S, Liberti L. Column generation algorithms for exact modularity maximization in networks. *Physical Review E.* 2010;82(4):046112. DOI: 10.1103/PhysRevE.82.046112.

Статья поступила в редколлегию 26.08.2019.
Received by editorial board 26.08.2019.