УДК 510.62

С. И. МАЛЕНЧЕНКО

АЛГОРИТМ ПЕРЕЧИСЛЕНИЯ КЛИК ГРАФА

Кликой называют подмножество C вершин графа $G = (V, E$ такое что любые две вершины x, $y \in C \subseteq V$ смежны, т. е. имеет ребро x, $y \in E$ [1]. Учитывая, что в большинстве приложений и тересны только нерасширяемые клики, определение часто попо няется условием нерасширяемости, а именно: не существует кли C', такой что $C \subset C'$ [2]. Нам тоже будет удобно придерживать определения с этим дополнением.

Задача перечисления всех клик не решается иначе, как пер бором. Она является универсальной переборной задачей в смы ле Кука-Карпа [3, 4], т. е. принадлежит к такому классу зад (классу NP), что если хоть одна задача из этого класса буд решена алгоритмом, требующим полиномиального (от разме входных данных) числа операций, то и любая другая задача этого класса тоже будет решена алгоритмом полиномиальн сложности. Поскольку в класс NP входят сотни задач, известны своей трудностью (таких, как задача коммивояжера или зад о ранце), то большинство исследователей сходятся в том, ч поиски алгоритмов полиномиальной сложности для задач это класса — дело совершенно безнадежное.

Даже если бы клика в графе находилась одной элементарн операцией, то и тогда нашлись бы графы, перечисление кл в которых имеет экспоненциальную сложность. Известным пр мером такого рода является граф Муна-Мозе́ра M_n. Граф M имеет 3^n вершин, разбитых на n троек; вершины каждой тройк не связаны ребром, но каждая вершина любой тройки связан с остальными 3^n—3 вершинами. Ясно, что если взять по одно вершине из каждой тройки (а это можно сделать 3^n способ ми), то получится n-вершинная клика.

Однако бывают классы графов, обладающих специфическо структурой, и, эксплуатируя эту специфику, можно обойти тру ность, присущую общей задаче и вполне эффективно перечи лить все клики.

Такие задачи возникают в математической лингвистике п анализе классов эквивалентности языковых единиц (на разны уровнях — от морфологии до семантики). В таких задачах вхо ной материал должен был бы сам распадаться на классы экв валентности или на клики, если представлять анализируемы языковые единицы вершинами, а отношения — ребрами. Одн ко из-за нерегулярности естественных языков, неполноты ма риала и т. п. эти идеальные клики более или менее «подпо чены»: есть мнимые ребра или не хватает нужных. Мы рассмо рим случай, когда есть мнимые ребра, и задача состоит в то чтобы выдать все клики графа (односвязного, ибо в случа

многосвязного графа нужно решить эту задачу отдельно для каждой односвязной компоненты).

Алгоритм будет состоять из двух частей. На первом этапе будет существенно использоваться специфическая «почти кликовая» структура графа, что позволит очень просто выявить большинство клик. В этой части будет использоваться понятие «хорошей» вершины. Вторая часть алгоритма — лексикографический перебор подмножеств — будет применяться к графу, сильно уменьшенному после работы первой части алгоритма. Эта часть является, по существу, усовершенствованием алгоритма, анонсированного в [2] в качестве лучшего из известных. В нашей версии из этого алгоритма будет изгнана неудобная для программирования рекурсивность, а сложности формирования и хранения множеств N и D (перспективных и уже проанализированных вершин) совсем исчезнут, ибо эти множества превратятся во множества больших и меньших номеров по сравнению со старшим номером вершины рассматриваемого подмножества.

Для изложения первой части алгоритма введем некоторые обозначения и определения. Будем обозначать клику через C, а если надо дополнительно указать, что какие-то вершины принадлежат этой клике, то их номера (мы будем нумеровать вершины графа индексами $1, 2, \ldots, i, \ldots, n$) ставятся нижними индексами при C. Например клику, в которую (может быть, наряду с другими) входит вершина 1, можно обозначать C_1. Будем также обозначать множество вершин, смежных с i, через V_i. В силу рефлексивности отношения эквивалентности $i \in V_i$, т. е. мы будем рассматривать граф с петлями.

Вершина i называется *хорошей* тогда и только тогда, когда $V_i = C_i$. Иначе, т. е. если $V_i \supset C_i$, вершина называется *плохой*. Другими словами, вершина хорошая, если она принадлежит некоторой клике, и все ребра, инцидентные вершине, принадлежат этой клике, ребер «в сторону» нет. Поэтому, как только мы находим хорошую вершину, то мы находим целую клику; более того, затем хорошую вершину можно удалить из рассмотрения, не затронув при этом остальные клики. Поскольку мы рассматриваем графы такой структуры, что большинство вершин будет хорошими, то нужно отыскать простой способ нахождения хороших вершин. Опишем такой способ.

Будем задавать граф в виде матрицы смежности $A = \{a_{ij}\}_{i,j=1}^{n}$, где

$$a_{ij} = \begin{cases} 0, & \text{если нет ребра } (i, j), \\ 1, & \text{если есть ребро } (i, j). \end{cases}$$

Поскольку граф с петлями, то $a_{ii} = 1$.

По этой матрице строится другая матрица $T = \{t_{ij}\}_{i,j=1}^{n}$, по формуле $t_{ij} = a_{ij} |V_i \cap V_j|$ **(1)**. В частности, t_{ii} — это степень вершины (обычная степень плюс 1 за счет петли).

Матрица T удобна во многих отношениях. Во-первых, он делает ненужной матрицу A, ибо несет всю информацию, заклю ченную в A, поскольку $(t_{ij} > 0) \equiv (a_{ij} = 1)$, а $(t_{ij} = 0) \equiv (a_{ij} = 0$ Первая эквивалентность следует из того, что граф — с петлями поэтому если $a_{ij} = 1$, то $V_i \cap V_j \neq 0$, вторая эквивалентность три виальна. Во-вторых, матрица T вообще удобнее для исследовани клик, чем A. В частности, имеет место

Теорема. *Следующие утверждения эквивалентны:* (*а*) *вершин i — хорошая;* (*в*) *в i-й строке матрицы T имеется t_{ii} чисел, ра ных t_{ii}.*

Доказательство. 1) а → в. Поскольку вершина i имее степень t_{ii}, то в i-й строке матрицы T в силу (1) имеется ровн t_{ii} ненулевых элементов. Пусть j, $j \neq i$ — любой из таких эл ментов. Индекс j — это номер вершины, входящий в клику C_i с вершинами. Значит, вершина j смежна с собой, с вершиной с остальными $t_{ii} - 2$ вершинами клики и, если вершина j — пло хая, с еще какими-то вершинами, т. е. $V_j \supseteq C_i = V_i$. Отсюда $V_j \cap V_i$ $= V_i$ и $t_{ij} = a_{ij} |V_j \cap V_i| = 1 \cdot |V_i| = t_{ii}$. Всего таких j имеется $t_{ii} - 1$, а одно недостающее число доставляет диагональный эл мент t_{ii}.

2) в → а. Поскольку $(t_{ij} > 0) \equiv (a_{ij} = 1)$, то вершина i смежн с собой и еще с $t_{ii} - 1$ вершинами. Пусть j — любая из эти вершин. Так как, в силу (1), $|V_i \cap V_j| = |V_i|$, то $V_j \supseteq V_i$, значит вершина j входит в клику C_i. Вершина i тоже входит в эту кли ку, и ее степень такова, что ребер, не ведущих в эту клику, не Значит, i — хорошая вершина.

Теорема доказана.

Теорема дает алгоритм нахождения хороших вершин, ибо утверждение (в) легко проверяется.

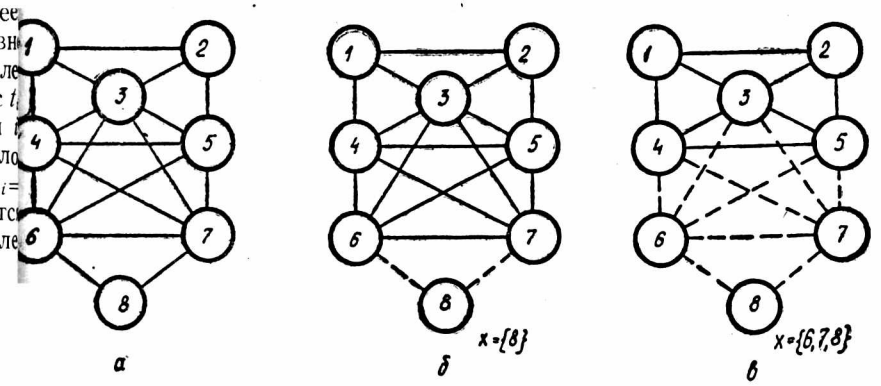
Таким образом, намечается первая часть алгоритма: нужно перебирать сверху вниз строки матрицы T и, найдя строку удовлетворяющую свойству (в), вывести выявленную клику удалить все обнаруженные хорошие вершины (запомнив их множестве X) и продолжать просмотр матрицы T, пока хороши вершины (а после удаления старых хороших вершин будут воз никать новые хорошие вершины) не исчезнут. При этом ил весь граф будет разобран, или останется его неразбираема часть, состоящая голько из плохих вершин.

Мы продемонстрируем эту простую идею на примере (рису нок). На рисунке, позиция *а*, показан исходный граф. В нем имеется только одна хорошая вершина — *8*. Когда эта вершин найдена, то находится клика {6, 7, 8}, вершина *8* удаляется (реб ра из нее на рисунке заменены на пунктирные), и она запоми нается в множестве X. Теперь появляются две новые хороши вершины *6* и *7*. По алгоритму обнаруживается хорошая вершин *6*, клика {3, 4, 5, 6, 7} и заодно еще одна хорошая вершина *7* Вершины *6* и *7* удаляются и запоминаются в X. Оставшийся граф из вершин $I = \{1, 2, 3, 4, 5\}$ не имеет хороших вершин и здес

нужно перейти ко второму этапу алгоритма — лексикографическому перебору всех подмножеств множества I.

На этом этапе можно справиться и с другим способом «запорченности» графа — нехваткой в клике малого числа пропущенных ребер. Для этого надо задать «допуски» — отождествить, например, числа t_{ii}—1 и t_{ii}. Но мы не будем далее останавливаться на этой ветви исследования.

Для ясности рассмотрим вторую часть алгоритма тоже на примере — неразобранной части графа (рисунок, позиция *в*), но более детально.



а *б* *в*

Для перебора всех подмножеств C множества I нам понадобятся три множества: уже сформированное множество X и два множества N и D, которые зависят от рассматриваемого подмножества. Пусть рассматривается подмножество C, $C \subseteq I$, $C = \{c_1, c_2, \ldots, c_k\}$, $c_1 < c_2 < \ldots < c_k$. В множество N включим те элементы из I, которые больше c_k, а в множество D — те элементы из $I \setminus C$, которые меньше c_k. Множество N нам понадобится для порождения очередного подмножества C, а множества X и D для того, чтобы узнать, расширяемой или нет является клика C.

Начнем с $C = \{1\}$. Это множество есть клика. Для него $D=0$, а $N = \{2, 3, 4, 5\}$. Добавим к C первый элемент из N. Теперь $C = \{1, 2\}$, $D = 0$, $N = \{3, 4, 5\}$. C — еще клика. Продолжим расширение C прежним образом: $C = \{1, 2, 3\}$, $D = 0$, $N = \{4, 5\}$. C все еще клика. Пойдем дальше: $C_{\text{нов}} = \{1, 2, 3, 4\}$. Это не клика. Тогда образуем $C_{\text{нов}}$ из $C_{\text{стар}}$ и следующего элемента N: $C_{\text{нов}} = \{1, 2, 3, 5\}$. Это не клика и N исчерпалось. Следовательно, $C = \{1, 2, 3\}$ — клика, не расширяемая в N. Осталось проверить, расширяется ли она в D или в X. Но $D = 0$, а ни одна вершина из X вместе с C не образует клики. Поэтому $C = \{1, 2, 3\}$ есть нерасширяемая клика и ее нужно вывести. (Теперь удобно заметить, что на первом этапе алгоритма перед выводом клики тоже нужно делать проверку нерасширяемости в текущем X).

Перейдем к следующему в лексикографическом порядке перспективному подмножеству. За подмножеством {1, 2, 3} лексикографически следует подмножества {1, 2, 3, 4}, {1, 2, 3, 4, 5}, {1, 2, 3, 5}, {1, 2, 4}, {1, 2, 5}, {1, 3} и т. д. Все подмножества, кроме последнего, рассматривать уже бессмысленно, ибо мы убедились, что подмножества {1, 2, 3} с любым индексом и N {4, 5} не образуют клики. Теперь нужно рассматривать подмножество {1, 3}, которое получается из {1, 2, 3} уменьшением мощности на единицу и заменой последнего элемента на следующий, т. е. 2 на 3.

Действуя и далее описанным образом, мы будем порождать подмножества согласно следующему протоколу: 1, 3 — расширяема в N, **1, 3, 4** — не расширяема в N; D, X — вывести; 1, 4 — не расширяема в N, расширяема в D — не выводить; 2 — расширяема в N; *2, 3* — расширяема в N; 2, 3, 4 — не клика; 2, 3, **5** — не расширяема в N, D, X — вывести; 2, 4 — не клика; 2, 5 — не расширяема в N, расширяема в D — не выводить; 3 — расширяема в N; 3, 4 — расширяема в N; 3, 4, 5 — не расширяема в N, расширяема в X — не выводить; 3, 5 — не расширяема в N, расширяема в D — не выводить; 4 — расширяема в N; 4, 5 — не расширяема в N, расширяема в D — не выводить; 5 — не расширяема в N, расширяема в D — не выводить; ___ — конец.

В результате работы второй части алгоритма выведены клики {1, 2, 3}, {1, 3, 4} и {2, 3, 5}.

Выше мы объяснили и продемонстрировали на примере работу алгоритма для перечисления клик графа. Говоря о программах, Дейкстра [5] справедливо утверждал, «что большинство из них появляется в виде, рассчитанном на механическое исполнение, но совершенно непригодном для человеческого восприятия». Это высказывание равно относится и к алгоритмам; поэтому мы не станем затемнять высказанные соображения их формальным техническим описанием.

Список литературы: 1. *Бе́рж К.* Теория графов и ее применения.— М.: Изд-во иностр. лит., 1962, с. 45—58. 2. *Рейнгольд Э., Нивергельт Ю., Део Н.* Комбинаторные алгоритмы. Теория и практика.— М.: Мир, 1980.— 123 с. 3. *Кук С. А.* Сложность процедур вывода теорем.— Кибернетический сборник, 1975, № 12, с. 5—15. 4. *Карп Р. М.* Сводимость комбинаторных проблем.— Кибернетический сборник, 1975, № 12, с. 16—38. 5. *Дейкстра Э.* Дисциплина программирования.— М.: Мир, 1978.—96 с.

Поступила в редколлегию 29.03.82.

4