УДК 681.3.06+519.6

ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ ПОИСКА НЕЗАВИСИМЫХ МНОЖЕСТВ НА ГРАФАХ

Ю.В. Ладыженский, В.А. Куркчи
Кафедра ПМиИ, ДонНТУ

Abstract

Ladyzhensky Y.V., Kourktchi V.A. Parallel algorithms for graph independent set problem. *Two parallel heuristic algorithms for a maximum independent set problem are given. Goldberg-Spenser's algorithm is modified. An algorithm based on greed heuristic and limited search is considered. Testing results for both algorithms are given and conclusions about precision properties are made.*

Введение

Задача о наибольшем независимом множестве является классической задачей теории графов. Эта задача часто возникает при параллельных вычислениях, проектировании и моделировании для минимизации обмена между процессорами и/или максимизации возможного числа процессоров [1]. Также она возникает при диагностике распределенных многопроцессорных систем [2].

Анализ последних исследований [3] показывает, что существует ряд приближенных параллельных алгоритмов решения данной задачи. Среди наиболее популярных подходов отметим моделирование отжига, нейросети и генетические алгоритмы. Однако требуется разработка принципиально новых алгоритмов, так как ни один из существующих не способен обеспечивать высокую точность наравне с высоким быстродействием. Отсутствуют единый метод оценки качества алгоритмов и методы доказательства возможности получения допустимого решения за конечное время. В [4] говорится о невозможности сравнения приближенных алгоритмов, так как для различных приложений задачи скорость и точность имеют различную важность.

Цель проводимых далее исследований состоит в получении параллельного алгоритма, позволяющего получить как можно более точный результат при невысокой временной сложности. Рассмотрены два алгоритма, один из которых имеет низкую временную сложность, а другой относительно высокую точность.

Вычислительная модель

Рассмотренные ниже алгоритмы рассчитаны на идеализированную многопроцессорную вычислительную модель EREW PRAM (Exclusive Read Exclusive Write Parallel Random Access Machine). В этой модели все процессоры имеют доступ к общей памяти, но перед доступом к участку памяти, процессор как бы занимает ее, и некоторое время только этот процессор может выполнять операции чтения и записи на данном участке памяти. В это время другие процессоры могут получить доступ к другим участкам памяти.

Все процессоры выполняют одну и ту же программу, но каждый из них имеет собственные:

- регистры, что позволяет процессорам работать независимо;
- программный счетчик, что позволяет выполнять на разных процессорах в один момент времени различные участки программы, в зависимости от ее структуры (ветвление, циклы);
- стек, что позволяет работать с различными наборами данных;
- “подпись”, то есть каждый процессор имеет свой номер, программно доступный для чтения, что позволяет использовать нумерацию процессоров в программах.

Система имеет встроенные часы, за один период которых каждым из процессоров выполняется ровно одна команда.

Анализ свойств алгоритма Голдберга-Спенсера

Основная идея алгоритма [5] состоит в следующем. Строится последовательность частичных раскрасок графа, начиная с тривиальной. Назовем цветовым классом множество вершин, окрашенных в один цвет. Каждая следующая раскраска получается из предыдущей путем попарного объединения цветовых классов.

Этот подход дает точное решение, если среди возможных разбиений цветовых классов на пары каждый раз выбирать такое, которое объединит вершины, принадлежащие искомому множеству только с теми вершинами, которые также принадлежат искомому множеству. Но если имеется $2p$ или $2p-1$ цветовых классов, то существует $(2p!)/(p!2^p)$ разбиений на пары, и перебор всех вариантов является труднорешаемой задачей. Поэтому рассматривается ограниченный набор разбиений. Введем в рассмотрение следующие функции:

$$\chi(p) = \begin{cases} p, & \text{если } p \text{ нечетное} \\ p-1, & \text{если } p \text{ четное} \end{cases} \quad (1)$$

$$rev(i,q;p) = (q-i) \bmod \chi(p) \quad (2)$$

$$index(i,j;p) = \begin{cases} (i+j) \bmod \chi(p) & \text{если } 0 \le i,j \le \chi(p)-1 \\ 2i \bmod \chi(p) & \text{если } j = \chi(p) \\ 2j \bmod \chi(p) & \text{если } i = \chi(p) \end{cases} \quad (3)$$

Для каждого q и i $(0 \le i,q \le \chi(p)-1)$, $j = rev(i,q;p)$ единственное целое число в интервале $[0,\chi(p)-1]$, такое что $index(i,j;r) = q$. Если принять $\pi_k = \{(C_i,C_j) : index(i,j;r) = k\}$, то множество $\pi = \{\pi_k\}$ есть множество разбиений цветовых классов на пары.

Важно также сохранить независимость цветовых классов. Для этого часть вершин при смешивании теряет свой цвет. Поэтому, при переходе к следующей раскраске, для каждого из найденных разбиений вычисляется число вершин, которое придется обесцветить и выбирается то, для которого это число минимально.

Этот процесс можно продолжать до тех пор, пока не останется только один цветовой класс, который и будет приближенным решением. Но такое решение может оказаться очень неточным, и в [5] предложено удалять из графа большие цветовые классы.

Если в текущей частичной раскраске есть цветовой класс C, такой что $|\Gamma(C)| \ge (n+m)/\log_2 n$, где Γ(C) – множество вершин смежных, по крайней мере, с одной вершиной из C, а n и m – количество вершин и ребер соответственно; то этот цветовой класс добавляется к множеству I и удаляется из графа вместе с Γ(C). Таким образом, алгоритм продолжает свою работу, пока в графе не останется ни одного цветового класса. Такой подход неявно содержит в себе жадную эвристику, суть которой – применение жадных алгоритмов на множествах, не составляющих матроид [1].

Все описанные выше действия можно выполнить, используя только суммирование, сортировки и нахождение минимумов и максимумов. В [6] доказано, что эти действия можно выполнить для k элементов за время $O(\log_2 k)$ на k процессорах. Количество попарных объединений равно $O(\log_2 n)$. Количество найденных больших цветовых классов также равно $O(\log_2 n)$. Следовательно, временная сложность алгоритма равна $O(\log_2^3 n)$.

В процессе работы алгоритма составляется единый список вершин и ребер, каждый элемент которого обрабатывается на отдельном процессоре. Поэтому для работы алгоритма требуется $O(n+m)$ процессоров. В [5] рассмотрена техника, позволяющая использовать

только $O((n+m)/\log_2 n)$ процессоров при увеличении временной сложности не более, чем в k раз, где k – некоторая константа. Для этого каждый процессор моделирует работу $\log_2 n$ процессоров. А при каждом удалении большого цветового класса уменьшенное количество виртуальных процессоров перераспределяется по реальным процессорам.

Применение рассмотренного алгоритма на практике показывает, что он обладает рядом существенных недостатков. Во-первых, после завершения работы алгоритма в множестве вершин может остаться несколько обесцвеченных вершин, причем эти вершины не смежны с множеством-результатом. В этом случае, полученное решение не является максимальным. Этот недостаток можно устранить, если заново построить тривиальную раскраску и продолжить выполнение алгоритма.

Во-вторых, как показали эксперименты, на графах с высокой плотностью алгоритм может зациклиться, так как невозможно построить большой цветовой класс.

Это можно доказать теоретически. Пусть $|\Gamma(C)| = k \le n$, тогда $k\log_2 n \ge (n+m)$ или $m \le k\log_2 n - n$. Рассмотрим наиболее благоприятный случай, когда $k = n$, тогда большой цветовой класс существует, если $m \le n(\log_2 n - 1)$. Это условие необходимое, но недостаточное, так как в действительности правая часть будет меньше. Учтем, что $\max(m) = n(n-1)/2 > n(\log_2 n - 1)$. Таким образом, при достаточно большом m, большого цветового класса не существует.

Этот недостаток алгоритма также можно исправить, если при построении $\Gamma(C)$ не учитывать обесцвеченные вершины.

Временная сложность алгоритма составляет $O(\log_2^3 n)$ [4], что является его достоинством и позволяет использовать его на больших графах. Но алгоритм имеет в целом невысокую точность. Например, на графах типа “звезда”, полученное алгоритмом независимое множество всегда состоит из одной вершины – центральной, так как еще при тривиальной раскраске $\Gamma(C) = n - 1 \ge (n + n - 1)/\log_2 n$, для $n \ge 5$. Однако очевидно, что наибольшее независимое множество состоит из всех нецентральных вершин и имеет мощность $n - 1$.

Жадный алгоритм

В [7] приведены различные жадные эвристики для поиска наибольшего независимого множества вершин, и показано, что последовательная жадная эвристика в общем случае имеет низкую точность. Однако, если совместить простые последовательные

жадные эвристики “Худший вон” и “Лучший внутрь”[7] с ограниченным перебором, то как показано далее это может привести к значительному повышению точности метода, хотя несколько увеличит временную сложность.

Алгоритм “Худший вон” заключается в поиске вершины с наибольшей степенью и удалении ее из графа до тех пор, пока в графе есть ребра. После завершения работы алгоритма оставшееся множество вершин и есть искомое.

Алгоритм “Лучший внутрь” заключается в поиске вершин с наименьшей степенью и добавление их в множество I. При этом найденные вершины удаляются из множества вершин графа вместе со всеми смежными вершинами. После завершения работы алгоритма множество I – наибольшее независимое множество.

Под ограниченным перебором понимается построение всех немаксимальных множеств заданного размера. Использование перебора придаст алгоритму некоторые свойства точных алгоритмов, а ограничение его степени фиксированным размером создаваемых множеств позволит сохранить полиномиальную временную сложность.

Погрешность жадной эвристики зависит от наличия в графе “плохих” вершин, которые не принадлежат искомому множеству, но имеют очень малую степень. Обычно таких вершин немного, поэтому можно строить полным перебором все независимые множества размером a, и дополнять их до максимальных с помощью жадного алгоритма “Лучший внутрь”. Будем называть полученные таким образом множества множествами уровня a и обозначим их как S_x^a, где x – множество вершин полученных перебором – базовые вершины.

Временная сложность жадного алгоритма составляет $O(n)$ [1]. Время, затрачиваемое на перебор можно определить как произведение времени построения одного множества, равное a, на количество множеств $\prod_{i=0}^{a-1} O(n-i) = O(n^a)$. То есть $T(n) = O\left(n^a (a+n)\right) = O(n^{a+1})$. Следовательно, временная сложность такого алгоритма составит $O(n^{a+1})$.

Этот подход улучшает алгоритм при использовании его для малых графов, так как a должно быть небольшим. Поэтому основной идеей предлагаемого алгоритма является поиск максимальных независимых множеств для $a = k$ и $a = k+1$, а затем получение либо вершины которая точно не принадлежит искомому множеству, либо вершины, которая ему принадлежит. Временная сложность такого алгоритма

равна $T(n) = O(n^{k+3})$, так как $O(n^{k+1} + n^{k+2})$ шагов выполняется не более чем n раз.

Для большинства графов, если перебор не увеличивает множества, скорее всего все “плохие” вершины уже удалены из графа. Тогда можно предположить, что, если $\left|S_x^a\right| = \left|S_{x\cup\{k\}}^{a+1}\right|$, то дальнейшее увеличение уровня не увеличит мощности независимого множества.

Известно, что если независимое множество максимально, но не является наибольшим, его можно увеличить, удалив несколько вершин, подобранных так, что это позволит добавить к множеству больше вершин, чем было удалено. Рассмотрим случай, когда $\left|S_x^a\right| = \left|S_{x\cup\{k\}}^{a+1}\right| < \max_y\left(\left|S_y^{a+1}\right|\right)$. S_x^a уже не будет расти, то есть ему принадлежит, по крайней мере, одна плохая вершина. Но существует множество с другими базовыми вершинами, мощность которого больше мощности S_x^a. Следовательно, плохая вершина принадлежит x. Если принять $k = 1$, то $|x| = 1$, следовательно, $x_l \in x$ не принадлежит искомому множеству и может быть удалена.

Таким образом, если при переходе от множества первого уровня ко множеству второго уровня мощности множеств не увеличились, то наибольшее найденное множество и есть искомое. Если некоторые множества увеличились, то мы можем найти вершины, которые не принадлежат искомому множеству.

Отдельно рассмотрим случай, когда увеличились все множества. Рассмотрим множество первого уровня и, основанные на нем, множества второго уровня. Зафиксируем базовую вершину x множества первого уровня. По крайней мере, одно множество второго уровня совпадает по мощности с множеством первого. Это то множество, вторая базовая вершина которого совпадет с первой эвристической вершиной соответствующего множества первого уровня. Но по условию у нас есть множества большей мощности. Таким образом, мы можем все множества второго уровня разделить на два множества:

$$S_\alpha = \left\{S_{x,k}^2 \middle\| S_{x,k}^2 \mid = \mid S_x^1 \mid\right\}$$

$$S_{d\alpha} = \left\{S_{x,k}^2 \middle\| S_{x,k}^2 \mid > \mid S_x^1 \mid\right\}$$

Обозначим, $S_x^- = \bigcup_{S_{d\alpha}} S_{x,k}^2, S_x^+ = \bigcup_{S_\alpha} S_{x,k}^2$. S^+ содержит в себе плохие вершины, удаленные при расширении, а S^- их не содержит. То есть, $K_x = S_x^+ \setminus S_x^-$ не принадлежит наибольшему независимому множеству.

Таким образом, в любом случае можно найти вершины, которые не принадлежат искомому множеству. Этот процесс следует продолжать до тех пор, пока у множеств первого и второго уровня различные мощности, то есть до тех пор, пока не останется граф, на котором жадная эвристика дает точный результат.

Формально, рассмотренный алгоритм состоит из следующих этапов:

1. Найти все множества первого уровня и $m_1 = \max_x(|S_x^1|)$.
2. Найти все множества второго уровня и $m_2 = \max_x(|S_x^2|)$.
3. Если $m_1 = m_2$, то наибольшее независимое множество найдено, иначе найти $K = \max_x(|K_x|)$, удалить его из множества вершин и вернуться к шагу 1.

При последовательной реализации временная сложность алгоритма равна $O(n^4)$. Этот показатель можно уменьшить, если использовать $O(n)$ процессоров. Каждый процессор построит “свои” множества первого и второго уровня, после чего анализирует их как описано выше. То есть временная сложность алгоритма при параллельной реализации равна $O(n^3)$.

Использование $O(n)$ процессоров также позволит сократить время, необходимое для другой работы, например поиск K, хотя это не повлияет на порядок временной сложности.

Экспериментальные данные

Оба рассмотренных алгоритма были программно реализованы и протестированы на эталонных графах DIMACS Challenge [4]. Результаты экспериментов представлены в таблице 1, где n – количество вершин графа, d – плотность (отношение числа ребер к $n(n-1)/2$), EX – точный результат (размер наибольшего независимого множества), GS – результаты по алгоритму Голдберга-Спенсера, GE – по предложенному алгоритму.

Из таблицы видно, что алгоритм GS имеет большую погрешность. Алгоритм GE для всех рассмотренных графов нашел точный результат. Временная сложность этого алгоритма даже в последовательной реализации равна $O(n^4)$. Это позволяет его использовать на достаточно больших графах.

Заключение

Показано, что алгоритм Голдберга-Спенсера зацикливается на графах с высокой плотностью и имеет низкую точность. Алгоритм

был исправлен и модифицирован. Полилогарифмическая временная сложность делает модификацию подходящей для приложений, критичных по времени решения задачи.

Таблица 1 Результаты тестирования алгоритмов

Название графа	n	D	EX	GS	GE
Brock200_1	200	0.75	21	13	21
Brock200_2	200	0.5	12	7	12
Brock200_3	200	0.4	15	10	15
Brock200_4	200	0.35	17	9	17
c-fat200-1 [2]	200	0.9223	15	5	15
c-fat200-2 [2]	200	0.8374	24	4	24
c-fat200-5 [2]	200	0.5742	58	7	58
Hamming-6-2	64	0.0952	32	11	32
Hamming-6-4	64	0.6508	4	2	4
Johnson-16-2-4	120	0.2353	8	8	8
Johnson-8-2-4	28	0.4444	4	4	4
Johnson-8-4-4	70	0.2319	14	6	14
Keller4	171	0.3509	11	7	11
MANN_a9	45	0.0727	16	16	16
Sanr200_0.7	200	0.3035	18	12	18

Предложен алгоритм, являющийся комбинацией жадных эвристик и перебора. Экспериментально показано, что предложенный алгоритм имеет высокую точность, что делает его привлекательным для дальнейших исследований.

Перечень источников

1. Новиков Ф.А. Дискретная математика для программистов. – СПб: Питер, 2002. – 304с.
2. Berman P. and Pelc A. Distributed Fault Diagnosis for Multiprocessor Systems, //Proc. 20th Annual International Symposium on Fault-Tolerant Computing Newcastle. – UK, 1990. – p. 340-346.
3. Bomze M., Budinich M., Pardalos P.M., Pelillo M. The maximum clique problem. in: Handbook of combinatorial optimization. – Adison-Wesley Publishing Company: 1998, 2410pp.
4. http://dimacs.rutgers.edu/Challenges/
5. Goldberg M., Spenser T. Constructing a maximum independent set in parallel. //SIAM J. Discr. Math. Vol. 2. – 1989, p. 322-328.
6. Cole R. Parallel merge sort. //Proc. 27th Annual Symp. on Foundations of Computer Science.- 1986.-p. 511-516.
7. Kopf R., Ruhe G. A computational study of the weighted independent set problem for general graphs. //Found. Control Engin. Vol.12. – 1987, p. 167-180.