Куркчи В.А
Магистрант группы ПО-99, ФВТИ, ДонНТУ
научный руководитель: Ладыженский Юрий Валентинович

«Параллельные алгоритмы для решения задач на графах»

Введение

Графы являются широко распространенной структурой данных, и задачи, связанные с ними, находят свое применение в различных областях науки и техники: эти задачи решаются при проектировании БИС, при распараллеливании вычислительных и других процессов, при организации ассоциативной памяти, в теории конечных автоматов, потоков в сетях и др. Задачи на графах многие из нас часто решают в повседневной жизни, даже не догадываясь об этом.

К сожалению, большая часть задач на графах – NP-полные, что в значительной степени затрудняет поиск решений. Примером такой задачи может послужить задача о развозке: поставщику необходимо доставить товары потребителям, при этом существует множество используемых маршрутов, каждый из которых требует затрат времени и средств; требуется определить, какие маршруты следует использовать, чтобы обслужить всех потребителей при минимальных транспортных расходах (эта задача сводится к задаче о наименьшем покрытии). Если у поставщика несколько клиентов в пределах города, эту задачу можно решить устно, но если клиентов несколько тысяч, и они разбросаны по всей стране, – поиск решения даже с помощью ЭВМ может затянуться. Ну а если поставщиком является транснациональная компания, то эта задача не может практически быть решена.

Однако задачи этого класса часто просто необходимо решить. Это обусловило появление направления, изучающего приближенные, или эвристические, алгоритмы. Большая часть NP-полных задач сводится к ЗЦЛП, что позволяет использовать в качестве решение любые локальные

максимумы (минимумы). Такое решение, конечно же, не является оптимальным, но допустимо. То есть, применительно к описанной выше задаче, транспортные расходы не будут минимальными, но будут гораздо меньше, чем при доставке «наугад».

Но процесс решения задачи большой размерности, даже приближенным алгоритмом может быть весьма длительным, что справедливо для всех задач и любых алгоритмов. Для ускорения поиска решений, используют параллельные алгоритмы. Временная сложность параллельных алгоритмов обычно, как минимум на порядок ниже, чем у аналогичных последовательных, а иногда и относится к другому классу. Например, простая сортировка в последовательной реализации имеет временную сложность от $O(n^2)$ до $O(n\log n)$, то есть полиномиальную, а сортировку n элементов на n процессорах можно выполнить за $O(\log n)$, то есть логарифмическое время. А решение задач на графах, при использовании алгоритмических методов, часто сводится к сортировкам. Так, в [5] описан приближенный параллельный алгоритм для поиска наибольшего независимого множества, использующий исключительно сортировки.

В настоящий момент существует достаточно большое количество параллельных вычислительных моделей: от гипотетических – EREW PRAM, CRCW PRAM, – до реально используемых – MIMD, вычислительные сети.

1 Обозначения и определения

$G=(V,E)$ – произвольный неориентированный, взвешенный граф, где $V=\{1,2,...,n\}$ - множество вершин G, и $E\subseteq V\times V$ - множество ребер G.

w_i – Положительные веса для всех вершин $i\in V$, связанные с i, объединены в векторе весов $w\in R^n$.

$A_G = (a_{ij})_{(i,j)\in V\times V}$ – Матрица смежности G. Симметричная матрица размерностью $n\times n$, где $a_{ij} = 1$, если $(i,j)\in E$ - ребро G, и $a_j = 0$, если $(i,j)\notin E$.

$N(v) = \{j \in V : a_{vj} = 1\}$ – Функция соответствия (окрестность). Для всех вершин v из G – множество всех вершин, смежных с v.

$\overline{G} = (V, \overline{E})$ – Дополнение графа $G = (V,E)$. Граф, где $\overline{E} = \{(i,j) \mid i,j \in V \,\&\, i \neq j \,\&\, (i,j) \notin E\}$.

$W(S) = \sum_{i\in S} w_i$ – Для подмножества $S \subseteq V$ – вес S.

$G(S) = (S, E \cap S \times S)$ – Подграф, порожденный S.

Полный граф – Граф $G = (V,E)$, такой, что все его вершины попарно смежны, то есть $\forall_{(i,j\in V)}(i \neq j \Rightarrow (i,j) \in E)$.

Клика – Такое множество $C \subseteq V$, что G(C) полный граф.

$\omega(G)$ – Кликовое число G – мощность наибольшей клики, то есть $\omega(G) = \max\{|S| : S \text{ клика } G\}$.

$\omega(G,w)$ – Кликовое взвешенное число – общий вес клики наибольшего веса, то есть $\omega(G,w) = \max\{W(S) : S \text{ клика } G\}$.

Независимое множество (устойчивое множество) – Такое множество $C \subseteq V$, элементы которого попарно не смежны.

$\alpha(G)$ – Мощность наибольшего независимого множества (устойчивое число графа G).

Вершинное покрытие – такое множество $C \subseteq V$, что каждое ребро $(ij) \in E \Rightarrow i \in C \vee j \in C$.

Δ – $\Delta = \{x \in R^n : \forall_{(i\in V)}(x_i \geq 0), e^T x = 1\}$, где e^T означает транспонированный единичный вектор e (следовательно $e^T = \{1,1,...,1\}$ и $e^T x = \sum_{i\in V} x_i$).

Δ_S – Для множества $S \subseteq V$ –грань Δ, соответствующую S, то есть $\Delta_S = \{x \in \Delta : i \notin S \Rightarrow x_i = 0\}$.

x^S – Характеристический вектор, являющийся вектором в Δ, определенный как $x_i^S = \begin{cases} 1/|S|, & i \in S \\ 0, & \textit{иначе} \end{cases}$.

$x^{S,w}$ – Весовой барицентр Δ_S, определяющий весовой вектор $w \in R^n$ или, что тоже самое, взвешенный характеристический вектор S, являющийся вектором с координатами $x_i^{S,w} = \begin{cases} w_j / W(S), & i \in S \\ 0, & \textit{иначе} \end{cases}$. Конечно, $x^S = x^{S,e}$.

2 Постановка задачи

Для рассмотрения была выбрана задача о наибольшем независимом множестве вершин. Эта задача была выбрана по двум причинам. Во-первых, она является фундаментальной, и к ней сводятся задачи о наименьшем покрытии, о совершенном паросочетании, о доминирующем множестве и многие другие. То есть большинство алгоритмов для решения этой задачи можно обобщить. А во-вторых, у этой задачи довольно большое число приложений, что делает поиск эффективного алгоритма для решения этой необходимым. В силу последнего, задача о наибольшем независимом множестве вершин стала в последнее время очень популярной среди исследователей.

Эту задачу можно решать как переборными методами, так и методами линейного и нелинейного программирования. Вполне естественно, что для поиска алгоритмов различных классов, требуются различные постановки задачи.

Рассмотрим наиболее часто встречающиеся в литературе. При этом следует сразу отметить, что задача о наибольшем независимом множестве неразрывно связана с задачей о наибольшей клике. Множество S является кликой G тогда и только тогда, когда S – независимое множество $\overline{G}$. Любой результат, полученный для одной из этих задач имеет эквивалентную форму

для другой задачи. Следовательно, $\alpha(G) = \omega(\overline{G})$, и, в общем случае, $\alpha(G, w) = \omega(\overline{G}, w)$.

Далее приводятся формулировки для взвешенных графов. Для невзвешенного случая следует принять $w = \{1\}^N$.

2.1 Постановка задачи в терминах теории множеств

Пусть задан неориентированный граф, тогда существует множество множеств вершин, таких, что две любые вершины, принадлежащие одному множеству несмежны. Найти множество наибольшей мощности.

Или [4]:

пусть

$G = (V, \Gamma)$

тогда:

$\exists Q = \{S_l \subseteq V \mid \forall (x_i x_j \in S_l^2)(x_i, x_j) \notin E\}$

найти:

$S \in Q \mid |S| = \max_{S_i \in W}(W(|S_i|))$

Требование максимальности множества опущено, так как если множество является наибольшим, то оно не может являться подмножеством другого допустимого множества. Однако в приближенных методах решения задачи следует обращать внимание на максимальность множества, так как результат может оказаться меньше точного.

Здесь же приведем другую формулировку задачи.

Найти $S : (x_i, x_j) \in S^2 \Rightarrow (x_i, x_j) \notin E \,\&\, \neg \exists S' : S \subset S'$.

2.2 Постановка задачи как ЗЦЛП

Пусть задан граф G=(V,A) или G=(V,E).

Простейшей формулировкой задачи является «реберная формулировка»:

$$\max \sum_{i=1}^{N} w_i x_i,$$

$$x_i + x_j \le 1, \forall (i, j) \in E,$$

$$x_i \in \{0,1\},\ i = 1,..., N.$$

При решении обычно используется релаксация данной формулировки:

$$x_i \in (0,1),\ i = 1,..., N.$$

В 1975г Немхаузер и Троттер доказали, что если переменная x_i принимает целое значение 1 в решении релаксации, то она принимает значение 1 по крайней мере в одном из оптимальных решений. Это предполагает неявный переборный алгоритм для решения задачи посредством решения ее линейной релаксации. Однако, в большинстве случаев, лишь несколько переменных имеют целое значение в оптимальном решении линейной релаксации, и зазор между оптимальными решениями задачи и ее линейной релаксации слишком велик, что значительно ограничивает использование этого подхода [1].

2.3 Постановка задачи как задачи квадратичных нуль-единиц

В 1990г, Шор рассмотрел интересную формулировку задачи о независимом множестве наибольшего веса. Шор вначале не изменял самой функции, но преобразовал ограничения.

Очевидно, что для двоичных x_i и x_j $x_i + x_j \le 1$ тогда и только тогда, когда $x_i x_j = 0$ [1]. Также можно записать ограничение на количество значений уравнением, корни которого есть допустимые значения, то есть $(x_i - 0) * (x_i - 1) = 0 \Leftrightarrow x_i^2 - x_i = 0$.

Таким образом, получена задача с квадратичными ограничениями, эквивалентная ЗЦЛП:

$$\max \sum_{i=1}^{N} x_i,$$

$$x_i x_j = 0,\ \forall (i, j) \in E,$$

$$x_i^2 - x_i = 0,\ i = 1,..., N.$$

Так как решается сама задача, а не ее релаксация, полученное решение является правильным, с другой стороны использование нелинейных

ограничений затрудняет как поиск глобального оптимума, так и локального. Но Шор сообщал об очень хороших результатах.

Данную формулировку можно еще упростить. Можно включить ограничения в саму функцию [1]:

$$\min f(x) = -\sum_{i=1}^{N} x_i + \sum_{(i,j)\in E} x_i x_j = X(A_G)X^T,$$

где $X = \{x_1, x_2, ..., x_N\}$ - вектор-строка переменных. К сожалению, добавление остальных ограничений не позволит записать функцию в матричной форме. Впрочем, такой формулировкой для данной задачи обычно подразумевают целочисленные переменные, то есть $X \in \{0,1\}^N$.

2.4 Континуальная постановка задачи

В 1965г, Моцкин и Штраус [2] установили связь между задачей о наибольшей клике и некоторыми стандартными задачами квадратичного программирования. Пусть $G = (V, E)$ - неориентированный (не взвешенный) граф, и Δ- стандартный симплекс в n-мерном Евклидовом пространстве R^n, а также Δ_S (грань Δ) соответствует подмножеству $S \subseteq V$. Теперь, рассмотрим следующую квадратичную функцию:

$$g(x) = x^T A_{\overline{G}} x, \tag{9}$$

где $A_{\overline{G}} = (a_{ij})_{i,j\in V}$ - матрица смежности дополнения графа G. Пусть x^* - глобальный максимум g на Δ. Моцкин и Штраус доказали, что устойчивое число графа G связано с $g(x^*)$ следующей формулой:

$$\alpha(G) = \frac{1}{1 - g(x^*)} \geq \frac{1}{1 - g(x)} \forall x \in \Delta. \tag{11}$$

Вдобавок, они доказали что подмножество вершин S является наибольшей кликой G тогда и только тогда, когда его характеристический вектор x^S является глобальным максимумом g на Δ.

Формулировке Моцкина-Штрауса присущ один недостаток, который связан с существованием неинформативного решения, то есть максимум g, который не является формой характеристических векторов.

Также Моцкин и Штраус сформулировали и доказали теорему Моцкина-Штрауса: глобальный оптимум программы

$$\min f(X) = \frac{1}{2} X(A_G) X^T,$$

удовлетворяющий

$$EX = 1$$

$$x_i \geq 0,$$

где $E = \{1\}^n$, равен

$$\frac{1}{2}\left(1 - \frac{1}{\omega(G)}\right).$$

При этом $x_i > 0 \Rightarrow x_i \in S$.

2.5 Полиномиальная постановка задачи

Рассмотрим следующую постановку задачи [6]:

$$\max F(x) = \sum_{i=1}^{n} (1 - x_i) \prod_{(i,j) \in E} x_j,\ x \in [0,1]^n.$$

С одной стороны, это функция n-ой степени, что может очень сильно затруднить поиск максимума. Но авторы этой постановки не только доказали ее правильность, но и предложили быстрый алгоритм для поиска максимума.

3 Связь с другими задачами

Эквивалентность рассматриваемой задачи задаче о наибольшей клике была описана ранее. Рассмотрим связь с другими задачами.

3.1 Задача о совершенном паросочетании

Паросочетанием, или независимым множеством ребер, называется множество ребер, в котором никакие два ребра несмежны. Совершенное паросочетание – паросочетание наибольшей мощности.

Для решения этой задачи, достаточно построить новый граф, каждая вершина которого соответствует одному ребру исходного графа. Вершины нового графа смежны, если смежны соответствующие ребра. Решение задачи о наибольшем независимом множестве графа и есть совершенное паросочетание.

3.2 Задача о наименьшем доминирующем множестве вершин

Доминирующее множество вершин – множество, смежное с любой вершиной графа, не входящей в это множество. Найти доминирующее множество наименьшей мощности.

I является наибольшим независимым множеством G тогда и только тогда, когда I является наибольшей кликой $\overline{G}$, и тогда и только тогда, когда $V \setminus I$ является минимальным вершинным покрытием G. Последнее следует из тождества Галлаи (Gallai)[6].

3.3 Задача о минимальной раскраске

Раскраской графа называют такое приписывание цветов (натуральных чисел) его вершинам, что никакие две смежные вершины не получают одинаковый цвет. Минимальная раскраска – раскраска минимальной мощности [4].

Для поиска минимальной раскраски можно использовать следующий алгоритм: Покрасить наибольшее независимое множества графа в один цвет, рассмотреть граф без окрашенного множества.

4 Точные алгоритмы

Для рассмотрения приближенных алгоритмов, следует хорошо понимать сущность задачи. Для чего следует рассмотреть точный алгоритм.

Начнем с простейшего переборного алгоритма – рекурсивного поиска с возвратами [4].

Алгоритм 3.1 – ПВ.

Вход: S – текущее независимое множество, V – множество вершин.

Выход: изменение глобальной переменной X, в которую помещается решение: наибольшее независимое множество вершин.

```
e := false;
m := 0;
for v ∈ V do
    if S ∪ {v} ∈ ε then
        e := true;
        ПВ(S ∪ {v}, V \ N(v))
    end if
end for
if e then
    if | S |> m then
        X := S; m :=| S |;
    end if
end if
```

В описанной процедуре проверка множества на независимость проверяется принадлежностью к множеству независимых множеств. Это сделано для сохранения контекста, но на практике выполняется следующим циклом:

```
f := true {множество считается независимым}
for u ∈ S do
    if (u, v) ∈ E then
        f := false; exit for;
    end if
end for
```

Этот алгоритм работает следующим образом, строится независимое множество, при этом при добавлении каждой вершины проверяется на независимость. Если расширить множество нельзя – оно максимально, то есть может претендовать на то, чтобы считаться наибольшим. Оно сравнивается с найденным ранее (или пустым, если найдено первое множество) и если оказывается больше, принимается за новое текущее множество. Далее алгоритм строит новое максимальное независимое множество.

Этот алгоритм является очень неэффективным, но позволяет понять общий принцип поиска независимых множеств. Еще одним достоинством алгоритма является то, что его можно легко распараллелить, причем это можно сделать несколькими способами, что позволит рассмотреть примеры параллельных алгоритмов.

Опишем возможные параллельные реализации. Во-первых, итерации внешнего цикла алгоритма не связаны между собой, таким образом, их можно выполнять параллельно, что ускорит работу алгоритма в n раз. Такая реализация будет хорошо работать на MIMD моделях и при распределенной обработке. Во-вторых, в алгоритме часто производятся операции над множествами, которые в параллельной реализации можно выполнять за одну операцию – каждый процесс изменяет один элемент множества –, также за одну операцию можно проверить множество на независимость – для этого каждый процесс проверяет существования в графе одного ребра. Такой вариант лучше использовать для вычислительных моделей с общей памятью, таких как EREW PRAM.

Рассмотрим теперь другой алгоритм, алгоритм Брона-Кербоша [4], его нельзя распараллелить, но он использует сокращенный перебор, учитывая особенности задачи, что делает его рассмотрение очень полезным.

Идея алгоритма аналогична предыдущему, то есть, начиная с пустого, строится множество вершин с сохранением независимости. Пусть S_k, уже полученное множество из k вершин, Q_k - множество вершин, которое можно добавить к S_k, то есть $S_k \cap N(Q_k) = 0$. Среди вершин Q_k будем различать те, которые уже использовались для расширения S_k (обозначим Q_k^-), и те, которые еще не использовались (Q_k^+). Тогда общая схема алгоритма будет состоять из следующих шагов.

Шаг вперед от k к k+1 состоит в выборе вершины $x \in Q_k^+$:

$$S_{k+1} = S_k \cup \{x\},$$
$$Q_{k+1}^- = Q_k^- - N(x),$$
$$Q_{k+1}^+ = Q_k^+ - (N(x) \cup \{x\}).$$

Шаг назад от k+1 к k:

$$S_k = S_{k+1} - \{x\},$$
$$Q_k^+ = Q_{k+1}^+ - \{x\},$$
$$Q_k^- = Q_{k+1}^- \cup \{x\}.$$

Если S_k максимальное, то $Q_k^+ = \varnothing$. Если $Q_k^- \neq \varnothing$, то S_k было расширено раньше и не является максимальным. Таким образом, проверка максимальности задается следующим условием: $Q_k^+ = Q_k^- = \varnothing$.

Перебор можно улучшить, если рассмотреть следующее. Пусть $x \in Q_k^+$ и $N(x) \cap Q_k^+ = \varnothing$. Эту вершину x никогда не удалить из Q_k^-, так как из него удаляются только вершины смежные с Q_k^+. Таким образом, существование x, такого что $N(x) \cap Q_k^+ = \varnothing$, является достаточным условием для возврата. Кроме того, $k \le n-1$.

Алгоритм 3.2 строит все максимальные независимые множества, как и алгоритм 3.1, и выбирает среди них наибольшее.

Алгоритм 3.2 – БК.

Вход: G(V,E) .

Выход: изменение глобальной переменной X, в которую помещается решение: наибольшее независимое множество вершин.

$m := 0$; {*размер текущего максимального множества*}
$k := 0$; {*размер рассматриваемого множества*}
$S[k] := \subset$ {*рассматриваемое множество*}
$Q^+[k] := V; Q^-[k] := \varnothing$
$M1$: {*шаг* 1}
select $v \in Q^+[k]$;
$S[k+1] := S[k] \cup \{v\}$;
$Q^-[k+1] := Q^-[k] \setminus N(v)$;
$Q^+[k+1] := Q^+[k] \setminus (N[v] \cup \{v\})$
$k := k+1$;
$M2$: {*проверка*}
for $u \in Q^-[k]$ *do*
  if $N(u) \cap Q^+[k] = \varnothing$ *then goto* $M3$; *end if*
end for
if $Q^+[k] = \varnothing$ *then*

```
    if Q⁻[k] = ∅ then
        if k > m then {k = мощности S[k]}
            m := k; X := S[k];
        end if
    goto M3;
    end if
else
    goto M1;
M3 : {шаг назад}
v := last(S[k]); k := k − 1;
S[k] := S[k + 1] − {v};
Q⁻[k] := Q⁻[k + 1] ∪ {v};
Q⁺[k] := Q⁺[k] \ {v};
if k = 0 & Q⁺[k] = ∅ then
    stop
else goto M2;
end if
```

Выводы

Рассмотрение точных алгоритмов позволяет не просто понять суть задачи: рассматривая методы и приемы сокращения перебора можно сделать неэквивалентный переход, тем самым, получив приближенный алгоритм. Или же просто выяснить некоторые свойства искомого множества или его окрестности, для того, чтобы положить это свойство в основу принципиально новой эвристики.

В данной работе не рассматривались приближенные методы. Этих методов в настоящий момент очень много и они очень разнообразны, и они были классифицированы по используемым эвристикам.

Большая часть задач на графах сводится к задачам оптимизации, поэтому названия классов алгоритмов часто совпадает с названием широко используемых методов и математических аппаратов. Сейчас наиболее популярны следующие направления: поиск табу, нейросети, генетические алгоритмы, случайный поиск, локальный поиск, жадные алгоритмы,

непрерывные алгоритмы, моделирование отжига. Недавно появилось новое направление – оптимизация с помощью муравьиных колоний.

Время от времени появляются алгоритмы, специально разработанные для тех или иных классов графов – графов, наиболее часто встречающиеся при решении задач определенной предметной области.

Дальнейшие изыскания предполагается направить на рассмотрение каждого из типа приближенных алгоритмов и, по возможности, создание аналогов с улучшенными характеристиками.

Список литературы

1 Bomze I.M, Budinich M, Pardalos P.M, Pelillo M. The maximum clique problem. Handbook of combinatorial optimization. – Adison-Wesley: 1999.

2 T.S. Motzkin and E.G. Straus, Maxima for graphs and a new proof of a theorem of Turan, Canad. J. Math., Vol. 17: 533-540, 1965.

3 http: // dimacs.rutgers.edu/Challenges/index.html

4 Новиков Ф.А, Дискретная математика для программистов: Спб: Питер, 2001. – 304с.

5 Goldberg M., Spenser T. Constructing a maximum independent set in parallel. //SIAM J. Discr. Math. Vol. 2. – 1989, p. 322-328.

6 Abello J, Butenko S, Pardalos P.M, Resende M.G.C. Finding independent set in a graph using continious multivariable formulations//AT&T labs research technical report.