ЖУРНАЛ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И МАТЕМАТИЧЕСКОЙ ФИЗИКИ, 2012, том 52, № 1, с. 164–176

УДК 519.7

ГЕНЕТИЧЕСКИЙ ЛОКАЛЬНЫЙ ПОИСК ДЛЯ ЗАДАЧИ О РАЗБИЕНИИ ГРАФА НА ДОЛИ ОГРАНИЧЕННОЙ МОЩНОСТИ[1)]

© 2012 г. Ю. А. Кочетов, А. В. Плясунов

(630090 Новосибирск, пр-т Акад. Коптюга, 4, Ин-т матем. СО РАН)

e-mail: jkochet@math.nsc.ru

Поступила в редакцию 29.03.2011 г.
Переработанный вариант 06.07.2011 г.

Для задачи о разбиении графа на доли ограниченной мощности разработан метод генетического локального поиска. На каждой итерации метода имеется набор локальных оптимумов задачи. Этот набор используется для целенаправленного поиска новых локальных оптимумов с меньшей погрешностью. Установлена плотная PLS-полнота задачи нахождения локальных оптимумов с рядом полиномиально проверяемых окрестностей. Показано, что в худшем случае число локальных улучшений может оказаться экспоненциальным при любых правилах выбора направления спуска. Для частного случая задачи, когда веса ребер равны единице и нахождение локальных оптимумов является полиномиальной процедурой, проведены численные эксперименты. Результаты экспериментов свидетельствуют о высокой эффективности разработанного метода и возможности решать задачи большой размерности. Библ. 25. Фиг. 2.

Ключевые слова: задача о разбиениях графа, плотная PLS-полнота, локальный поиск, генетические алгоритмы.

ВВЕДЕНИЕ

Рассмотрим простой неориентированный граф с четным числом вершин. Каждому ребру приписан неотрицательный вес. Требуется найти разбиение множества вершин на два таких равных по мощности подмножества, чтобы сумма весов ребер, концы которых лежат в разных подмножествах, была минимальной. Это – известная NP-трудная задача о разбиении графа (Graph Partitioning, или кратко GP), исследованию которой посвящены десятки статей. В приложениях возникают более сложные ситуации, когда требуется разбить множество вершин на несколько подмножеств. Если число подмножеств разбиения k задано, то получаем задачу, известную под названием задачи о минимальном k-разрезе (см. [1]). Эта задача полиномиально разрешима для фиксированного k (см. [2]).В общем случае, когда параметр k является частью входа, она оказывается NP-трудной (см. [3]).Также известно, что для этой задачи существует полиномиальный приближенный алгоритм с оценкой точности $(2 - 2/k)$ (см. [4]). Если мощности подмножеств разбиения должны совпадать, то задача полиномиально аппроксимируема с точностью $|V|(1 - 1/k)$ (см. [4]), где V – множество вершин графа. Вариант задачи, в котором подмножества разбиения имеют заданные мощности, а веса удовлетворяют неравенству треугольника, принадлежит классу APX для каждого фиксированного k (см. [5]).

Далее рассматривается задача, в которой требуется разбить множество вершин на несколько подмножеств ограниченной мощности, но число подмножеств не фиксируется. Также предполагается, что веса ребер могут иметь произвольный знак. При малой размерности задачи оптимальное решение может быть найдено методом ветвей и границ. Задачу легко представить в терминах целочисленного линейного программирования и воспользоваться, например, пакетом CPLEX. Однако уже при 50 вершинах разбить граф на доли таким способом крайне затруднительно из-за больших затрат машинного времени. Задача остается NP-трудной, поэтому в настоящей работе основной акцент сделан на получение локальных оптимумов с малой относительной погрешностью. Для решения задачи разработан генетический метод локального поиска. Вводится несколько разных по мощности окрестностей, и исследуется вопрос о вычислительной сложности получения локальных оптимумов. Показано, что такие задачи локального поиска являются

[1)] Работа выполнена при финансовой поддержке РФФИ (коды проектов 09-06-00032, 11-07-00474).

плотно PLS-полными (см. [6], [7]). Следовательно, для любого алгоритма локального поиска в худшем случае потребуется экспоненциальное число шагов при любом правиле выбора направления спуска. Ситуация не изменится при использовании любых детерминированных или рандомизированных, полиномиальных или неполиномиальных правил или их комбинаций. Тем не менее, как показывает вычислительный эксперимент, алгоритмы локального поиска оказываются быстрыми в среднем (см. [8]). Более того, они полиномиальны, если веса ребер одинаковы. В электронной библиотеке [9] для сравнения таких алгоритмов собраны лучшие достижения в области построения сложных тестовых примеров и результаты расчетов для них. Разработанный метод локального поиска тестировался на этих примерах и примерах из реальных приложений (см. [10]). Во всех случаях он оказался высоко эффективным и быстро находил локальные оптимумы высокого качества.

Статья организована следующим образом. В разд. 1 приводится математическая постановка задачи и обсуждаются возможности ее точного решения. В разд. 2 излагается общая схема генетического локального поиска, подробно обсуждаются ее основные элементы: построение начальной популяции, жадные алгоритмы и алгоритмы локального улучшения, процедуры скрещивания двух решений и возникающие здесь проблемы нумерации подмножеств. В разд. 3 доказывается экспоненциальная нижняя оценка в худшем случае на число шагов локального улучшения для ряда полиномиально проверяемых окрестностей. Этот результат устанавливается как для произвольных весов ребер, когда можно показать плотную PLS-полноту задачи локального поиска, так и для случая неотрицательных весов, когда такой полноты может и не быть. В последнем разделе приводятся результаты численных экспериментов на графе автомобильных дорог Советского района города Владивостока и графах из библиотеки (см. [9]). Показано, что для некоторых из них разработанный метод позволяет находить лучшие решения, чем ранее известные.

1. ПОСТАНОВКА ЗАДАЧИ

Рассмотрим взвешенный неориентированный граф $G = (V, E)$ с множеством вершин V и множеством ребер E. Каждому ребру $e \in E$ приписан вес w_e. Для произвольного разбиения множества вершин на непересекающиеся подмножества определим вес разреза как сумму весов ребер, каждое из которых имеет вершины в разных подмножествах. Для заданного натурального числа p задача состоит в нахождении разбиения множества V на подмножества, каждое мощности не более p, имеющего минимальный вес разреза (см. [11], [12]).

Эту задачу будем обозначать через $\mathrm{GP}_{\leq p}$.

Представим эту задачу в терминах целочисленного линейного программирования. Пусть множество J задает номера подмножеств в разбиении множества V. Введем две группы булевых переменных:

$$x_{ij} = \begin{cases} 1, & \text{если вершина } i \text{ принадлежит подмножеству } j, \\ 0 & \text{в противном случае,} \end{cases}$$

$$y_e = \begin{cases} 1, & \text{если ребро } e \text{ принадлежит разрезу,} \\ 0 & \text{в противном случае.} \end{cases}$$

С использованием этих переменных задача может быть представлена следующим образом:

$$\min \sum_{e \in E} w_e y_e$$

при ограничениях

$$y_e \geq x_{i_1 j} - x_{i_2 j}, \quad y_e \geq x_{i_2 j} - x_{i_1 j}, \quad e = (i_1, i_2) \in E, \quad j \in J,$$

$$\sum_{j \in J} x_{ij} = 1, \quad i \in V,$$

$$\sum_{i \in V} x_{ij} \le p, \quad j \in J,$$

$$y_e, x_{ij} \in \{0, 1\}, \quad e \in E, \quad i \in V, \quad j \in J.$$

Целевая функция задачи определяет вес разреза. Первая группа ограничений требует включения ребра в разрез, если его вершины принадлежат разным подмножествам. Вторая группа ограничений гарантирует включение каждой вершины в одно из подмножеств. Последнее ограничение устанавливает верхнюю границу на мощность каждого подмножества.

Запись задачи в терминах целочисленного линейного программирования позволяет использовать уже разработанные программные средства для решения данной задачи. Одним из наиболее мощных средств в этой области является пакет CPLEX. С его помощью можно решать как задачи линейного программирования, причем большой размерности, так и задачи частично целочисленного и полностью целочисленного программирования. К сожалению, задача о минимальном разрезе оказалась трудной для этого программного средства. Даже на 50 вершинах для случайных графов с вероятностью появления ребер 0.33 поиск нужного разбиения оказался малоэффективным. За 12 ч работы на PC Pentium IV, RAM 512 Mb было найдено решение, которое заметно хуже решения, полученного за несколько секунд методом генетического локального поиска. Разрыв целочисленности для этой задачи оказывается значительным, часто больше 50%, что сильно затрудняет работу методов ветвей и границ, а также их способность быстро находить хорошие приближенные решения для отсечения бесперспективных вариантов.

Заметим, что задачу можно представить и без переменных y_e, сделав замену $y_e = 1 - \Sigma_{j \in J} x_{i_1 j} x_{i_2 j}$ при $e = (i_1, i_2)$. Получаем следующую задачу:

$$\max \sum_{e \in E} w_e \sum_{j \in J} x_{i_1 j} x_{i_2 j}$$

при ограничениях

$$\sum_{j \in J} x_{ij} = 1, \quad i \in I,$$

$$\sum_{i \in V} x_{ij} \le p, \quad j \in J,$$

$$x_{ij} \in \{0, 1\}, \quad i \in V, \quad j \in J.$$

Эта задача имеет квадратичную целевую функцию и линейные ограничения и также может быть решена пакетом CPLEX. Результаты численных экспериментов показывают, что такое изменение математической модели не приводит к успеху. На примерах той же размерности дерево ветвлений возрастает еще быстрее и через 2.5 ч машинного времени метод останавливается из-за нехватки памяти. Найденное к этому моменту наилучшее решение часто оказывается хуже, чем в предыдущем случае.

Существенным недостатком этих постановок (и не только этих) является наличие большого числа симметрии. Любая перенумерация множества J по сути не меняет решения, но формально приводит к новому решению. В частности, оптимальных решений получается не менее $|J|!$. Это обстоятельство приводит к огромному числу избыточных ветвлений в методе ветвей и границ и заметному ослаблению нижних границ. Эффективных методов распознавания и борьбы с симметриями пока не найдено. Определенные успехи в этом направлении можно найти в [13], [14]. Таким образом, разработка приближенных методов и, в частности, методов локального поиска, особенно для больших графов, является актуальной проблемой.

2. ГЕНЕТИЧЕСКИЙ ЛОКАЛЬНЫЙ ПОИСК

Разработке генетических алгоритмов и на их основе – гибридных схем посвящена обширная литература (см., например, [15]). Успех того или иного подхода здесь во многом определяется

учетом специфики задачи, адаптацией общих схем метаэвристик к особенностям решаемой задачи.

Генетический локальный поиск по сути является гибридной схемой, сочетающей в себе идеи генетических алгоритмов и локальной оптимизации. В англоязычной литературе такие алгоритмы получили название Memetic Algorithms. Это – итерационные методы, на каждом шаге которых имеется некоторый набор локальных оптимумов. Согласно принятой терминологии (см. [15]), его принято называть популяцией. Шаг состоит в выборе двух элементов из популяции (их называют родителями), построении на их основе нового решения и применении к нему методов локального улучшения. После получения нового локального оптимума принимается решение о пополнении популяции новым элементом. Шаги повторяются до тех пор, пока не будет выполнен критерий остановки. Общую схему такого метода можно представить следующим образом.

Генетический локальный поиск

Шаг 1. Построить начальную популяцию.

Шаг 2. Пока не выполнен критерий остановки, делать следующее:

Шаг 2.1. Выбрать два решения из популяции.

Шаг 2.2. Построить по ним новое решение.

Шаг 2.3. Применить к нему алгоритм локального улучшения.

Шаг 2.4. Добавить новый локальный оптимум в популяцию и удалить из нее наихудший локальный оптимум.

Шаг 3. Предъявить лучшее найденное решение.

Остановимся подробнее на основных элементах этого подхода.

2.1. Выбор начальной популяции

С теоретической точки зрения для сходимости метода выбор начальной популяции не имеет принципиального значения. Тем не менее хорошая стартовая популяция может заметно снизить время получения оптимального или приближенного решения с заданной погрешностью. Для того чтобы добиться этого, используют различные жадные стратегии и локальный поиск. Рассмотрим следующие жадные алгоритмы.

Random Greedy. Положим $k = \lceil |V|/p \rceil$. Для каждого из k подмножеств выбирается по одной вершине случайно с равномерным распределением. Далее делается $|V| - k$ шагов, на каждом из которых одно из подмножеств пополняется одной вершиной. Вершина выбирается случайным образом с равномерным распределением на множестве вершин, еще не распределенных среди подмножеств. Вершина помещается в то подмножество, для которого приращение целевой функции оказывается минимальным.

HeavyGreedy. Этот алгоритм отличается от предыдущего только выбором вершины. На каждом шаге используется новое правило: вершина выбирается так, чтобы суммарный вес инцидентных ей ребер был максимальным.

LightGreedy. Этот алгоритм является аналогом предыдущего, но используется противоположная стратегия. На каждом шаге выбирается вершина, имеющая минимальный суммарный вес инцидентных ей ребер.

К сожалению, эти полиномиальные алгоритмы, как и многие другие, приводят к решениям с большой относительной погрешностью. Например, для графа add20 (см. [9]), где $|V| = 2395$, $|E| = 7462$, $w_e \equiv 1$, $p = 1198$, наилучшее известное решение имеет величину разреза 596, Алгоритм RandomGreedy в среднем дает 3730, HeavyGreedy 1928, LightGreedy 1923. Аналогичные результаты получаются на других графах и других жадных стратегиях.

Чтобы улучшить результаты, можно многократно применять вероятностные жадные алгоритмы и выбирать из полученных решений наилучшее. Дальнейшее улучшение может быть получено применением алгоритмов локального спуска к каждому из таких решений. Алгоритмы, построенные по такому принципу, получили название GRASP (Greedy Randomized Adaptive Search Procedure) (см. [15]). Именно такой подход применяется для получения начальной популяции. Его эффективность существенно зависит от выбора окрестностей. Мощные окрестности дают возможность получать решения с малой погрешностью, но требуют больших затрат на выполнение каждого шага алгоритма. Малые окрестности избавлены от этого недостатка, но часто приводят к плохим результатам. Найти “золотую середину” и подобрать эффективную структуру

данных для просмотра окрестности и поиска в ней наилучшего элемента представляется серьезной проблемой. Для каждой задачи ее приходится решать заново. Поэтому имеет смысл пользоваться как малыми, так и большими окрестностями в сочетании с различными жадными алгоритмами.

2.2. Локальный поиск

Рассмотрим следующие окрестности, которые будут использоваться на шагах 1 и 2.3 общей схемы генетического локального поиска для получения локальных минимумов.

Окрестность MS (Move or Swap). Для любого допустимого решения задачи элементами этой окрестности являются допустимые решения, получаемые из данного переносом одной вершины в другое, быть может новое, подмножество или выбором двух вершин в разных подмножествах и заменой их одна на другую. Использование этой окрестности может приводить к росту или сокращению числа подмножеств. Число ее элементов оценивается величиной $O(n^2)$.

Окрестность KL (см. [16]). Для любого допустимого решения задачи элементы этой окрестности строятся с помощью следующей итерационной процедуры.

Шаг 1. В окрестности MS текущего решения находится наилучший элемент, т.е. допустимое решение, которое в наибольшей степени уменьшает значение целевой функции или, если таких нет, в наименьшей степени увеличивает его.

Шаг 2. Для выбранного таким способом наилучшего решения, определяемого одной вершиной или парой вершин, осуществляется переход из текущего решения в наилучшее.

Шаги 1, 2 выполняются, пока это возможно, таким образом, чтобы используемые вершины или пары вершин не повторялись. Каждое выполнение шагов 1, 2 порождает соседнее решение для исходного. Их число также оценивается величиной $O(n^2)$.

Окрестность KL_1. Эта окрестность состоит из одного элемента, получаемого на первом шаге предыдущей процедуры. По построению она является частью окрестности MS.

Окрестность FM (см. [17]). Данная окрестность строится аналогично окрестности KL. Единственное отличие состоит в том, что каждый шаг этой процедуры, на котором просматриваются пары вершин, разбивается на два этапа. На первом этапе выбирается первая вершина, перенос которой в другое подмножество приводит к наибольшему уменьшению целевой функции или, если таких нет, к наименьшему его увеличению. На втором этапе для уже выбранных подмножеств аналогичным образом выбирается вторая вершина. Если таких несколько, то берется любая из них. Мощность такой окрестности совпадает с мощностью окрестности KL.

Окрестность FM_1. Она состоит из одного элемента, получаемого на первом шаге предыдущей процедуры. По построению она может не совпадать с окрестностью KL_1.

2.3. Выбор родительской пары

Существуют различные стратегии выбора родительской пары. Наиболее известным является турнирный отбор и пропорциональная селекция (см. [15]). Основная идея этих процедур состоит в том, чтобы дать некоторое предпочтение “хорошим” решениям. В данном алгоритме начальная популяция состоит из локальных оптимумов. Поэтому можно ожидать, что решения не сильно отличаются друг от друга по целевой функции и в качестве родителей можно брать любые элементы популяции.

Пусть A и B – два допустимых решения задачи. Будем считать, что каждое решение представлено вектором длины $|V|$, где в i-позиции стоит номер подмножества, содержащего вершину i. У такой кодировки решений имеется следующий недостаток. Меняя нумерацию подмножеств, можно получать разные коды одного и того же решения. Если же решения по сути различны, то, меняя нумерацию подмножеств одного из решений, можно сделать эти решения максимально похожими друг на друга. Таким образом, возникает следующая задача оптимальной нумерации подмножеств: найти такую нумерацию подмножеств решения B, чтобы число вершин, оказавшихся в разных подмножествах, для решений A и B было минимальным. Если этот минимум равен нулю, то решение A совпадает с решением B. Решение этой задачи играет ключевую роль для сохранения структурных свойств родительских решений.

Пусть величина m_{ij} задает число вершин, которые входят в одно из подмножеств i или j и не входят в другое. Введем переменные задачи

$$x_{ij} = \begin{cases} 1, & \text{если подмножество } i \text{ назначается на подмножество } j, \\ 0 & \text{в противном случае.} \end{cases}$$

Тогда задача оптимальной нумерации представима в виде (см. [19])

$$\min \sum_{i \in J} \sum_{j \in J} m_{ij} x_{ij}$$

при ограничениях

$$\sum_{i \in J} x_{ij} = 1, \quad j \in J,$$

$$\sum_{j \in J} x_{ij} = 1, \quad i \in J,$$

$$x_{ij} \in \{0, 1\}, \quad i, j \in J.$$

Получили известную задачу о назначениях, которая легко решается точно. Далее будем предполагать, что после выбора двух решений для скрещивания всегда решается задача о назначениях для оптимальной нумерации подмножеств одного из решений.

2.4. Скрещивание

Цель процедуры скрещивания состоит в получении нового допустимого решения по двум заданным решениям. Новое решение должно унаследовать характерные особенности родительских решений; так, например, если оба родительских решения имеют в разбиениях одно и то же подмножество, то оно должно сохраниться и в новом решении. Процедуры скрещивания, или кроссоверы (crossover), должны порождать различные решения при многократном применении к одной и той же паре. Это важное свойство гарантирует нужное разнообразие в популяции и предотвращает преждевременное зацикливание. Пусть A и B – два родительских решения, представленные строками длины $|V|$. Требуется построить новое решение C. Ниже приводятся пять рандомизированных алгоритмов скрещивания, которые с равной вероятностью используются на шаге 2.2 общей схемы метода.

Стандартный k-точечный кроссовер (см. [15]). Выберем случайным образом k позиций $i_1, \ldots, i_k$. Присвоим первым позициям $1, 2, \ldots, i_1$ в решении C значения из решения A, следующим позициям $i_1 + 1, \ldots, i_2$ значения из решения B, затем снова из A и т.д. Получим строку C, где каждой вершине приписано некоторое подмножество. Такое решение может оказаться недопустимым. Некоторые подмножества могут исчезнуть, а некоторые получить более чем p вершин. В этом случае из "переполненных" подмножеств случайным образом удаляются лишние вершины, а затем применяются жадные алгоритмы для получения допустимого решения задачи.

Вероятностный кроссовер. Он является наиболее простым и естественным для данной задачи. Если вершина в родительских решениях принадлежит одному и тому же подмножеству, то она остается в этом подмножестве. Если же решения A и B дают разные подмножества, то выбирается одно из них случайным образом: подмножество из решения A с вероятностью P_A, подмножество из решения B – с вероятностью P_B:

$$P_A = \frac{S_A}{S_A + S_B}, \quad P_B = \frac{S_B}{S_A + S_B},$$

где $S_A(S_B)$ – суммарный вес ребер, связывающих данную вершину с подмножеством из решения $A(B)$. Вершины распределяем по подмножествам, руководствуясь данным правилом, до тех пор, пока не будет достигнута граница p на мощность подмножеств. Если для очередной вершины одно из родительских решений указывает на подмножество, которое в решении C уже имеет мощность p, то вершина включается во второе подмножество.

Кроссовер Лазевского (см. [18]). Выберем в решении A некоторые подмножества $j_1, \ldots, j_s$ и сохраним их в решении C. Эти подмножества уже не будут меняться. Рассмотрим оставшиеся подмножества в решении B. Удалим из них вершины множеств $j_1, \ldots, j_s$ и добавим часть этих подмножеств в решение C. Чтобы получить допустимое решение задачи, достаточно распределить оставшиеся вершины между подмножествами, не нарушая ограничений на мощность каждого подмножества. Для этих целей применяются жадные эвристики.

Циклический кроссовер (см. [19]). Предыдущий кроссовер стремится сохранить следующее свойство решений A, B: некоторые вершины должны быть в одном подмножестве. В циклическом кроссовере акцент делается на другом свойстве: некоторые вершины должны быть в разных подмножествах.

Выпишем решения A, B одно под другим и получим перестановку с повторениями. Будем раскладывать эту перестановку на циклы и цепи следующим образом. Выберем произвольную позицию в решении A и пометим ее. В этой позиции стоит номер подмножества. Обозначим его через j_0. Если номер j_0 стоит в этой же позиции решения B, то получим цикл, и процедура останавливается. В противном случае в решении B стоит номер $j \neq j_0$. В решении A выберем произвольное вхождение номера j в непомеченных позициях. Найденная новая позиция помечается, и процедура повторяется до тех пор, пока на очередном шаге не будет получен номер j_0 или очередного вхождения номера j обнаружить не удается. В первом случае получаем цикл, во втором – цепь. Указанная процедура расщепляет исходную перестановку на цепи и циклы. Теперь каждому циклу и цепи произвольным образом присваивается метка A или B. Новое решение C получается по исходным решениям выбором в каждой позиции номера подмножества, соответствующего решениям A или B в зависимости от принадлежности этой позиции одному из циклов или цепей.

Кроссор связывающих путей (см. [20]). Его идея состоит в построении последовательности решений, конечными элементами которой являются решения A и B и выбором одного из элементов этой последовательности в качестве решения C. Последовательность строится с помощью окрестности MS так, что каждый следующий элемент является соседним для предыдущего. Каждый элемент такой последовательности “наследует” свойства своих родителей. Средний элемент, равноудаленный от A и B, выбирается в качестве решения C. Легко заметить, что таких последовательностей можно построить много. При построении последовательности используются жадные стратегии для выбора очередного элемента.

3. АНАЛИЗ ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ

На шагах 1 и 2.3 общей схемы генетического локального поиска предполагается получение локальных минимумов для окрестностей MS, KL, KL_1, FM, FM_1. В данном разделе исследуется вопрос о вычислительной сложности такой задачи. Наша цель – показать экспоненциальную нижнюю оценку, в худшем случае – для числа шагов алгоритмов локального улучшения для каждой из указанных окрестностей как в общем случае, так и для частного случая неотрицательных весов.

Далее понадобятся некоторые определения из [6]–[8] и, в частности, понятия класса PLS и сведения одной задачи к другой в этом классе. Неформально класс PLS (*polynomial time local search problems*) состоит из пар вида $\Pi = (\mathrm{OP}, \mathrm{N})$, где OP – оптимизационная задача и N – окрестность, относительно которой требуется найти локальный оптимум. Такая пара принадлежит классу PLS, если за полиномиальное время от длины записи исходных данных задачи OP можно выполнить следующее:

проверить корректность исходных данных и найти допустимое решение задачи;

проверить допустимость решения и вычислить для него значение целевой функции;

проверить локальную оптимальность любого допустимого решения и, если оно не является локальным оптимумом, найти соседнее решение с лучшим значением целевой функции.

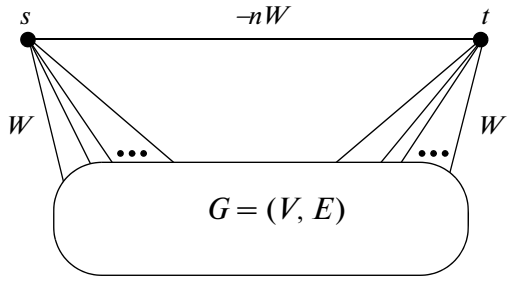
Так как каждая из рассматриваемых окрестностей полиномиальна по мощности, то задача $\mathrm{GP}_{\leq p}$ с каждой из этих окрестностей принадлежит классу PLS.

Говорят, что задача Π_1 *PLS-сводится* к задаче Π_2, если существуют такие полиномиально вычислимые функции h и g, что выполняется следующее:

1) по любому входу x задачи Π_1 функция h вычисляет некоторый вход $h(x)$ задачи Π_2;

2) по любому решению s для $h(x)$ функция g находит некоторое решение $g(s, x)$ для входа x;

3) если s – локальный оптимум для $h(x)$, то $g(s, x)$ – локальный оптимум для x.



Фиг. 1.

Задачу П называют *PLS-полной*, если любая задача этого класса PLS-сводится к задаче П. Известно, что задача GP с каждой из окрестностей Swap, KL, KL_1, FM, FM_1 является PLS-полной (см. [6], [21]). Окрестности KL, KL_1, FM и FM_1 строятся для задачи GP по тем же правилам, что и для задачи $GP_{\leq p}$ с заменой окрестности MS на окрестность Swap. В свою очередь определение окрестности Swap для задачи GP получается из определения окрестности MS при $p = n$, если ограничиться операциями только с двумя вершинами.

Лемма 1. *Задача* $GP_{\leq p}$ *с каждой из окрестностей* MS, KL, KL_1, FM, FM_1, *является* PLS-*полной*.

Доказательство. Предъявим функции h и g, обладающие нужными свойствами для сведения задачи GP к $GP_{\leq p}$ с соответствующими окрестностями. Пусть граф $G = (V, E)$ с неотрицательными весами на ребрах задает вход задачи GP. Построим новый граф $G' = (V', E')$ с весами w'_e для задачи $GP_{\leq p}$ по следующему правилу (см. фиг. 1). Введем две дополнительные вершины s и t и положим $V' = V \cup \{s, t\}$. Пусть $W = 1 + \Sigma_{e \in E} w_e$. Добавим ребра вида (s, v) и (t, v) для всех $v \in V$ с весом W и ребро (s, t) с весом $-W|V|$, т.е.

$$E' = E \cup (s, t) \cup \{(s, v), (t, v) \mid v \in V\}.$$

Наконец, положим $p = n + 1$ для целого $n = |V|/2$. Это построение задает функцию h.

Прежде чем определить функцию g, заметим следующее свойство локальных минимумов задачи $GP_{\leq p}$. У каждого из них вершины s и t лежат в разных подмножествах. Большой отрицательный вес ребра (s, t) это гарантирует. Более того, большой положительный вес ребер вида (s, v) и (t, v) для $v \in V$ обеспечивает разбиение множества V' только на два подмножества, а выбор $p = n + 1$ гарантирует равную мощность этих подмножеств. Такие разбиения будем называть правильными. Остальные разбиения будем называть неправильными. Итак, локальные минимумы являются правильными разбиениями. Зададим функцию g следующим образом. Правильное разбиение $V' = V'_1 \cup V'_2$ переводится в разбиение множества V на два подмножества: $V_1 = V'_1 \setminus \{s, t\}$ и $V_2 = V'_2 \setminus \{s, t\}$. Любое неправильное разбиение переводится в разбиение $V = \{1, 2, ..., n\} \cup \{n + 1, ..., 2n\}$. Легко проверить, что построенные функции действительно задают PLS-сведение. Лемма доказана.

Графом переходов $TG_{\Pi}(x)$ для входа x задачи П называют ориентированный граф, множество вершин которого совпадает с множеством допустимых решений задачи ОР. Каждой паре вершин s и s' соответствует дуга (s, s'), если решение s' входит в окрестность решения s и целевая функция для s больше, чем для s'. Высотой вершины называют длину кратчайшего пути из этой вершины в сток (локальный минимум). Высота графа $TG_{\Pi}(x)$ определяется как максимальная высота его вершин.

Граф переходов является ациклическим, а его высота дает нижнюю оценку в худшем случае на число шагов любого алгоритма локального улучшения.

Пусть задача Π_1 сводится к задаче Π_2 и h, g – соответствующие функции. Говорят, что сводимость является *плотной*, если для любого входа x задачи Π_1 можно указать подмножество R допустимых решений входа $y = h(x)$ задачи Π_2 такое, что верно следующее.

1. В R содержатся все локальные минимумы для входа y.

2. Существует полиномиальный алгоритм, который для каждого решения p входа x позволяет находить решение $q \in R$ для входа y такое, что $g(q, x) = p$.

3. Пусть граф переходов $\mathrm{TG}_{\Pi_2}(y)$ содержит ориентированный путь из вершины $q \in R$ в вершину $q' \in R$ такой, что в нем нет промежуточных вершин из R, и пусть $p = g(q, x)$, $p' = g(q', x)$ – соответствующие решения для входа x. Тогда $p = p'$, или граф переходов $\mathrm{TG}_{\Pi_1}(x)$ содержит дугу из вершины p в вершину p'.

Задачу П называют *плотно-полной* в классе PLS, если любая задача из этого класса плотно сводится к задаче П. Задача GP с каждой из окрестностей Swap, KL, KL_1, FM, FM_1 является плотно-полной (см. [6], [21], [22]).

Теорема 1. *Задача* $\mathrm{GP}_{\leq_p}$ *с каждой из окрестностей* MS, KL, KL_1, FM, FM_1 *является плотно-полной.*

Доказательство. Убедимся в том, что построенное сведение в лемме 1 является плотным для окрестности MS. Возьмем в качестве R множество правильных разбиений, тогда первые два условия, очевидно, выполняются. Проверим третье условие. Рассмотрим ориентированный путь в графе $\mathrm{TG}_{\mathrm{GP}_{\leq_p}}$ из вершины $q \in R$ в вершину $q' \in R$ без промежуточных вершин из R и покажем, что такой путь может быть только дугой (q, q'). Предположим, что это не так. Значит, из правильного разбиения можно получить неправильное локальным изменением по окрестности MS с уменьшением величины разреза. Окрестность допускает две операции: перенос вершины в другое, быть может, пустое подмножество и замену вершины из одного подмножества на вершину из другого подмножества. Первая операция может приводить к неправильным разбиениям, если одна вершина переносится в пустое подмножество. Обозначим эту вершину через v. Если v не совпадает с s и t, то величина разреза увеличивается как минимум на W и такое разбиение не может соответствовать вершине на выбранном ориентированном пути. Если же v совпадает, например, с s, то величина разреза возрастает еще больше – на nW, и снова приходим к противоречию.

Вторая операция может приводить к неправильным разбиениям, если в результате вершины s и t оказываются в одной доле. В этом случае величина разреза возрастает не меньше, чем на $2nW$. Противоречие.

Итак, каждый ориентированный путь, начинающийся в R, проходит только через вершины из R. По каждому правильному разрезу однозначно восстанавливается допустимый разрез в задаче GR путем удаления вершин s, t и всех ребер, инцидентных им. Величина разреза меняется на константу. Следовательно, наличие ребра (q, q') влечет существование ребра (p, p').

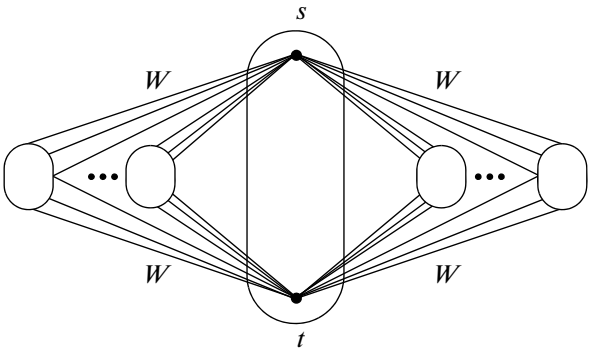
Для завершения доказательства остается заметить, что при определении окрестностей KL и FM используется итеративная процедура, на каждом шаге которой выбирается наилучшее соседнее решение по окрестности MS или ее модификации. Значит, для правильных разбиений на каждом шаге будут выбираться правильные соседние разбиения, а пути в графе переходов будут обладать указанным свойством и для окрестностей KL, KL_1, FM, FM_1. Теорема доказана.

Из полученного плотного сведения и существования задачи экспоненциальной высоты в классе PLS (см. [21]) вытекает

Следствие 1. Для задачи $\mathrm{GP}_{\leq_p}$ любой алгоритм локального улучшения с каждой из окрестностей MS, KL, KL_1, FM, FM_1 требует в худшем случае экспоненциального числа итераций при любом правиле выбора направления спуска.

Полученные нижние оценки остаются экспоненциальными при любых детерминированных или рандомизованных процедурах выбора соседнего решения с лучшим значением целевой функции и не зависят от того, являются ли эти процедуры полиномиально ограниченными или нет.

Еще одно приложение теоремы 1 связано с задачей нахождения локального минимума, достижимого алгоритмом локального улучшения из заданной стартовой точки. В целом это значительно более сложная задача, чем обычная задача нахождения произвольного локального минимума, которая рассматривалась до сих пор. Из определения класса PLS следует, что алгоритм локального поиска требует полиномиально ограниченной памяти. Действительно, на каждой итерации алгоритма необходимо хранить только текущее решение, длина которого, по определению класса PLS, ограничена полиномом от длины входа. Поиск лучшего соседа, если он существует, осуществляется за полиномиальное время и, следовательно, также требует полиномиально ограниченной памяти. Известно (см. [21]), что в классе PLS существует задача, для которой нахождение локального оптимума, достижимого локальными улучшениями из заданной стартовой точки, является PSPACE-трудной задачей. Там же показано, что плотная сводимость двух задач из класса PLS влечет полиномиальную сводимость соответствующих задач отыскания локального оптимума, достижимого из заданной стартовой точки. Из этих результатов и теоремы 1 следует утверждение.



Фиг. 2.

Следствие 2. Задача нахождения локального минимума, достижимого алгоритмом локального улучшения из заданной стартовой точки, является PSPACE-трудной для задачи $GP_{\leq p}$ с любой из окрестностей MS, KL, KL_1, FM, FM_1.

Рассмотрим задачу $GP_{\leq p}$ с неотрицательными весами на ребрах. Если в предложенной конструкции положить $w_{(st)} = 0$, то появляются новые локальные минимумы, у которых вершины s и t находятся в одном подмножестве, а число подмножеств больше двух (см. фиг. 2). Другими словами мы не можем утверждать, что задача в этом частном случае остается PLS-полной. Тем не менее экспоненциальная нижняя оценка, полученная в следствии 1, остается справедливой.

Теорема 2. *Для задачи* $GP_{\leq p}$ *с неотрицательными весами на ребрах, любой алгоритм локального улучшения с каждой из окрестностей* MS, KL, KL_1, FM, FM_1 *требует в худшем случае экспоненциального числа итераций при любом правиле выбора направления спуска.*

Доказательство. Воспользуемся снова конструкцией из доказательства леммы 1, но положим $w(st) = 0$. Рассмотрим граф переходов $TG_{GP_{\leq p}}$ для окрестности MS. Выделим в этом графе вершины, соответствующие правильным разбиениям. Если среди них найдутся вершины, высота которых экспоненциальна, то утверждение теоремы будет доказано.

Выделим произвольный ориентированный путь, начинающийся с вершины для правильного разбиения. Убедимся в том, что в этом пути по-прежнему все вершины соответствуют только правильным разбиениям. Мы уже проверяли, что выделение одной вершины в новое подмножество не может приводить к разбиениям с меньшим весом разреза. Проверим теперь вторую операцию – замену одной вершины на другую. Пусть вершина s переносится в другую долю, заменяя вершину v. Такая замена удаляет из разреза n ребер, соединявших вершину s, с другой долей, но добавляет $(n + 1)$ ребер для вершин из доли без s и t. В результате величина разреза возрастает на W, не считая ребер для вершины v. Следовательно, любой ориентированный путь в графе переходов $TG_{GP_{\leq p}}$, начинающийся с вершины для правильного разбиения, снова будет проходить только через вершины для правильных разбиений.

Рассуждая далее, как в конце доказательства теоремы 1, получаем, что для любой из окрестностей KL, KL_1, FM, FM_1 любой ориентированный путь в соответствующем графе переходов $TG_{GP_{\leq p}}$, начинающийся с вершины для правильного разбиения, снова будет проходить только через вершины для правильных разбиений.

При доказательстве теоремы 1 было установлено, что длина таких путей в худшем случае оказывается экспоненциальной от длины записи исходных данных. Следовательно, для любой из окрестностей MS, KL, KL_1, FM, FM_1 в соответствующем графе переходов $TG_{GP_{\leq p}}$ имеются вершины, высота которых экспоненциальна. Теорема доказана.

4. РЕЗУЛЬТАТЫ ЧИСЛЕННЫХ ЭКСПЕРИМЕНТОВ

Разработанный метод запрограммирован на языке PASCAL (Delphi-6). Он тестировался на графе автомобильных дорог Советского района г. Владивостока и графах из библиотеки “Graph Partitioning Archive” (см. [9]). В первом случае граф имел 217 вершин и 258 ребер. Интерес к этому

графу вызван работой по оптимизации транспортных потоков (см. [10]), где в целях декомпозиции требовалось быстро находить приближенные решения задачи $GP_{\leq p}$. Результаты расчетов для разных значений величины p таковы:

p	150	130	120	110	100	90	80	70	60	50	40	30
Количество итераций	208	265	206	245	450	402	316	409	293	723	356	452
Размер разреза	4	5	5	5	7	8	10	11	11	15	16	22

Размер популяции выбирался равным 20. Если в течение 20 итераций подряд не удавалось обновить популяцию, вычисления прекращались. При построении начальной популяции использовалась жадная эвристика Random Greedy. При скрещивании использовался один из пяти кроссоверов, выбираемый каждый раз случайным образом.

К получаемым решениям применялась процедура локального улучшения для получения локального минимума сначала по окрестностям малой мощности, а затем по окрестности KL. Заметим, что число ребер в получаемом разрезе достаточно мало. Так как граф связен, то погрешность не может быть большой. Точное решение задачи неизвестно. Чтобы понять, насколько можно улучшить результаты за счет большего расчетного времени, проведен следующий эксперимент. Для $p = 50$ размер популяции увеличен до 50 и метод останавливался после выполнения 1500 итераций. В результате граф был разбит на 6 долей и число разрезаемых ребер оказалось равным 13, что на 2 меньше, чем было указано. Таким образом, алгоритм способен порождать лучшие решения за счет больших временны́х затрат. Однако это не всегда оправдано в методах декомпозиции, когда достаточно хорошего приближенного решения.

Многие графы дорог обладают свойством планарности. В частности, этим свойством обладает и упомянутый выше граф. Эта специфика может влиять на результаты расчетов. Поэтому интересно попробовать данный метод на других, например, случайно сгенерированных графах. В библиотеке [9] имеются примеры таких графов и результаты расчетов для других алгоритмов. Наилучшие найденные решения хранятся в библиотеке в качестве рекордных значений. В отличие от упоминавшегося графа автомобильных дорог, графы из данной библиотеки имеют большу́ю размерность: несколько тысяч вершин. Применение окрестности KL в таких условиях не оправдано. Поэтому все расчеты проводились только с окрестностью FM. Их использование потребовало разработки специальной структуры данных, позволяющей быстро находить наилучший элемент в окрестности. Такая структура данных получена модификацией структуры данных из [17]. Приведем результаты расчетов для графа с $|V| = 2851$, $|E| = 15093$:

p	1426	1497	713	746
Размер популяции	50	50	50	50
Количество итераций	647	713	1377	889
Рекорд	189	181	429	378
Решение алгоритма	189	181	399	363

Рассматривались четыре значения величины p: 1426, 1497, 713, 746. Первый вариант $p = 1426$ максимально близок к классической задаче о разбиении графа на две равные доли. Подчеркнем еще раз, что в задаче $GP_{\leq p}$ наряду с такими решениями допускаются и другие, где имеется большее число долей мощностью не более p. Второй вариант $p = 1497$ допускает мощность одной из долей больше половины, т.е. любое решение для первого случая является допустимым для второго. В результате получаем, что значение разреза для $p = 1426$ (189) оказывается больше, чем для $p = 1497$ (181). Для этих значений p генетический алгоритм повторил известные рекордные значения и не смог их улучшить даже при бóльшей популяции (150 вместо 50) и значительно большем числе итераций. Однако при меньших значениях $p = 713$ и $p = 746$ удалось получить новые рекордные значения разреза, заметно лучше известных ранее.

Приведем результаты расчетов для графа с $|V| = 2395$, $|E| = 7462$:

p	1198	599	628	300	1256
Размер популяции	50	50	50	50	50
Количество итераций	1054	1000	1000	1216	763
Рекорд	596	1203	1184	1758	551
Решение алгоритма	599	1174	1130	1761	541

Рассматривались пять значений величины p. Для двух значений $p = 1198$ и $p = 300$ величина разреза оказалась чуть больше рекордного. Для трех оставшихся значений удалось получить новые рекордные значения целевой функции. Таким образом, можно утверждать, что разработанный алгоритм локального поиска является конкурентоспособным. Он позволяет быстро находить приближенные решения с малой величиной разреза.

5. ЗАКЛЮЧЕНИЕ

В работе рассмотрена задача о минимальном разрезе графа на доли ограниченной мощности. Для этой задачи разработан генетический алгоритм локального поиска. В качестве начальной популяции используются локальные оптимумы относительно некоторых полиномиально проверяемых окрестностей. Показано, что задача локального поиска для любой из них является плотно PLS-полной. Этот результат позволяет получить в худшем случае экспоненциальные нижние оценки для любого алгоритма локального улучшения, использующего данные окрестности. Если ограничиться исходными данными с одинаковыми весами ребер, то задача локального поиска с любой из рассматриваемых окрестностей становится полиномиально разрешимой. Для этого случая приводятся результаты вычислительного эксперимента как для реальных данных, так и для данных из электронной библиотеки “Graph Partitioning Archive”. Полученные результаты позволяют заключить, что предлагаемый метод позволяет быстро находить приближенные решения высокого качества и может быть использован для решения прикладных задач.

Для дальнейшего повышения эффективности численных методов решения данной задачи можно исследовать различные гибридные схемы локального поиска, например заменить алгоритмы локального улучшения на траекторные метаэвристики: вероятностный поиск запретами, поиск с чередующимися окрестностями или алгоритм имитации отжига (см. [15]). Как правило, гибридные схемы позволяют повысить эффективность поиска, если удается воспользоваться сильными сторонами каждого из методов (см. [23]).

Другим многообещающим направлением является возможность распараллеливания алгоритмов. Исследования в области параллельных алгоритмов для задачи о рюкзаке (см. [24]) и задачи линейного программирования (см. [25]) показывают, что такой подход позволяет существенно повысить размерность решаемых задач.

СПИСОК ЛИТЕРАТУРЫ

1. *Ausiello G., Crescenzi P., Gambosi G. et al.* Complexity and approximation: combinatorial optimization problems and their approximability properties. Berlin: Springer, 1999.
2. *Goldschmit O., Hochbaum D.S.* Polynomial algorithm for the k-cut problem // Proc. 29-th Ann. IEEE Symp. Foundations of Computer Sci. IEEE Comput. Soc., 1988. P. 444–451.
3. *Vazirani V.V.* Approximation algorithms. Berlin: Springer, 2001.
4. *Saran H., Yazirani V.V.* Finding k-cuts within twice the optimal // SIAM J. Comput. 1995. V. 24. P. 101–108.
5. *Guttmann-Beck N., Hassin R.* Approximation algorithms for minimum k-cut // Algorithmica. 2000. V. 27. № 2. P. 198–207.
6. *Кочетов Ю.А.* Вычислительные возможности локального поиска в комбинаторной оптимизации // Ж. вычисл. матем. и матем. физ. 2008. Т. 48. № 5. С. 788–807.
7. *Кочетов Ю.А., Пащенко М.Г., Плясунов А.В.* О сложности локального поиска в задаче о p-медиане // Дискретный анализ и исслед. операций. Сер. 2. 2005. Т. 12. № 2. С. 44–71.
8. *Alekseeva E., Kochetov Yu., Plyasunov A.* Complexity of local search for the p-median problem // European J. Operat. Res. 2008. V. 191. P. 736–752.
9. *Walshaw C.* Graph partitioning archive. http://staffweb.cms.gre.ac.uk/~c.walshaw/partition
10. *Нурминский Е.А.* Декомпозиция и параллелизация вычислительных процессов на основе фейеровских процессов с малым возмущением // Тр. XIV Байкальской междунар. школы-семинара “Методы оптимизации и их приложения”. Т. 1. Иркутск: ИСЭМ СО РАН, 2008. С. 128–137.
11. *Chopra S., Rao M.R.* The partition problem // Math. Program. 1993. V. 59. P. 87–115.
12. *Rendl F., Wolkowicz H.* A projection technique for partitioning the nodes of a graph // Ann. Operat. Res. 2005. V. 58. P. 155–179.
13. *Bertold T.* Automatic detection of orbitopal symmetries // Abstracts of OR-2008 Conf. Augsburg, 2008. P. 198.
14. *Peinhardt M.* Breaking model symmetry for graph partitioning // Abstracts of OR-2008 Conf. Augsburg, 2008. P. 198.
15. *Dreo J., Petrowski A., Siarry P., Taillard E.* Metaheuristics for hard optimization. Berlin: Springer, 2006.

16. *Kernighan B.W., Lin S.* An effective heuristic procedure for partitioning graphs // Bell System Techn. J. 1970. V. 49. P. 291–307.
17. *Fiduccia C.M., Mattheyses R.M.* A linear-time heuristic for improving network partitions // Proc. XIX Design Automation Conf. Los Alamitos. CA: IEEE Comput. Soc. Press. 1982. P. 175–181.
18. *Laszewski G.* Intelligent structural operators for the k-way graph partitioning problem // Proc. IV Internat. Conf. Genetic Algorithms. 1991. P. 45–52.
19. *Moraglio A., Kim Y.-H., Yoon Y., Moon B-R.* Geometric crossovers for multiway graph partitioning // Evolutionary Comput. 2007. V. 15. P. 445–474.
20. *Glover F., Laguna M.* Tabu search. Boston: Kluwer Acad. Publ., 1997.
21. *Yannakakis M.* Computational complexity // Local Search in Combinatorial Optimizat. Chichesfer: Wiley, 1997. P. 19–55.
22. *Kochetov Yu.* Facility location: Discrete models and local search methods // Combinatorial Optimizat. Meth. and Applic. Amsterdam: IOS Press, 2011. P. 97–134.
23. *Hooker J.N.* Integrated methods for optimization. New York: Springer, 2007.
24. *Посыпкин М.А., Сигал И.Х.* Применение параллельных эвристических алгоритмов для ускорения параллельного метода ветвей и границ // Ж. вычисл. матем. и матем. физ. 2007. Т. 47. № 9. С. 1524–1537.
25. *Гаранжа В.А., Голиков А.И., Евтушенко Ю.Г., Нгуен М.Х.* Параллельная реализация метода Ньютона для решения больших задач линейного программирования // Ж. вычисл. матем. и матем. физ. 2009. Т. 49. № 8. С. 1369–1384.

Сдано в набор 16.09.2011 г. Подписано к печати 05.12.2011 г. Формат бумаги $60 \times 88^{1}/_{8}$
Цифровая печать Усл. печ. л. 22.0 Усл. кр.-отт. 4.4 тыс. Уч.-изд. л. 22.0 Бум. л. 11.0
Тираж 196 экз. Зак. 2005

Учредители: Российская академия наук, Вычислительный центр им. А.А. Дородницына РАН

Издатель: Российская академия наук. Издательство “Наука”, 117997, Москва, Профсоюзная ул., 90
Оригинал-макет подготовлен МАИК “Наука/Интерпериодика”
Отпечатано в ППП “Типография “Наука”, 121099, Москва, Шубинский пер., 6