ПРЕДМЕТНОЕ ОБУЧЕНИЕ

Романовский Иосиф Владимирович

ЗАДАЧА ШТЕЙНЕРА НА ГРАФАХ И ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

1. ВВЕДЕНИЕ

Задачи оптимизации, которые можно решать с помощью динамического программирования, очень разнообразны. Но иногда возможность приведения оптимизационной задачи к динамическому программированию бывает замаскирована неестественной или слишком громоздкой конструкцией.

Здесь мы рассмотрим задачу именно со слишком громоздкой конструкцией, которая позволяет, однако, решить рассмотренную задачу при не очень больших значениях параметров. Тем не менее, приемлемых алгоритмом значений параметра в некоторых ситуациях может оказаться достаточно. Речь пойдет о так называемой задаче *Штейнера на графах.* Нам будет естественно начать с классической задачи Штейнера, чтобы объяснить, откуда взялось название. Дальше последует постановка интересующей нас задачи, затем напоминание метода Дейкстры для поиска кратчайших путей в графе, затем некоторые простые модификации этого алгоритма, которые будут нам полезны, а затем уже сам алгоритм.

2. ЗАДАЧА ШТЕЙНЕРА

Исходная задача Якоба Штейнера (1796-1863) имеет очень простую формулировку:

Н*а плоскости задано n точек (можно представлять себе карту и дома на ней). Требуется соединить эти точки ломаными линиями таким образом, чтобы каждая точка была соединена с каждой и чтобы суммарная длина всех проведенных линий была минимальна.*

Алгоритма решения этой задачи, то есть метода нахождения точного минимума, не существует до настоящего времени. Хорошо известны достаточные условия: в решение могут входить промежуточные точки, и все соединения должны быть отрезками, соединяющими точки (исходные и промежуточные). В каждой промежуточной точке должны сходиться три отрезка, а в исходных точках не более трех. Угол между отрезками, сходящимися в одной точке не должен быть меньше 120°.

Типичное решение, удовлетворяющее перечисленным условиям, изображено на рисунке 1. Семь точек (черные кружки) соединены сетью, в которой четыре дополнительные точки. В каждой дополнительной точке три отрезка сходятся под углом 120°. Соединение явно не оптимально.

Трудность же заключается в том, что одному и тому же набору точек может отвечать несколько таких решений. На рисунке 2 показано два решения, соответствующих четырем точкам – вершинам прямоугольника. Здесь исходных точек четыре, а промежуточных две. И черная, и серая соединяющая сеть удовлетворяют необходимым условиям оптимальности, но длины у них различны.

В связи с ростом интереса к экстремальным задачам на графах и успехами в решении задач об оптимальных деревьях (ос-

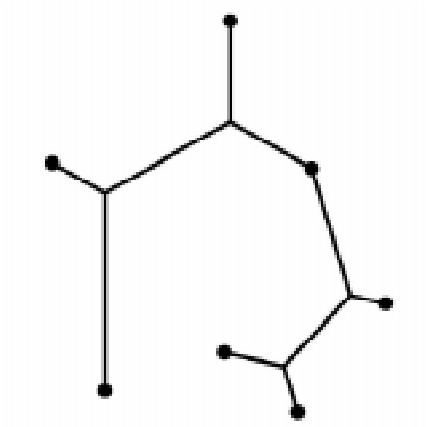

Рисунок 1.

товное дерево минимального веса, дерево кратчайших путей, кратчайшее дерево путей) естественно появились трудно решаемые варианты этих задач, в которых нашли аналогию с задачей Штейнера. Такие задачи стали называть графовыми задачами Штейнера. Мы будем рассматривать здесь одну из таких формулировок.

Пусть задан ориентированный граф с множеством вершин M и множеством дуг N. Каждой дуге u сопоставлен некоторый положительный вес c_u. Заданы также вершина i_0 (будем называть ее начальной вершиной) и непустое множество вершин M_X (которые, конечно, называются конечными). Требуется найти частичный подграф, то есть такие подмножества $M' \subset M$, $N' \subset N$, что у каждой дуги $u \in N'$ ее начало $jb(u)$ и конец $je(u)$ содержатся в M'. Этот частичный подграф должен содержать пути из i_0 в любую из вершин множества M_X, а сумма весов дуг из N' должна быть минимальной.

До того как мы будем рассматривать метод для решения этой задачи, вспомним кое-что известное (метод Дейкстры) и приспособим его к нашим потребностям.

3. КРАТЧАЙШИЕ ПУТИ ОТ ОДНОЙ ДО ОДНОЙ ВЕРШИНЫ

Обычный метод Дейкстры заключается в том, что, как полагается в динамическом программировании, единичная задача нахождения кратчайшего пути от вершины i_0 до вершины решается вместе со всеми аналогичными задачами нахождения кратчайшего пути от i_0 до любой другой вершины $i \in M$. Особенностью метода Дейкстры в сравнении с другими методами, основанными на идее динамического программирования, является особый порядок вычислений.

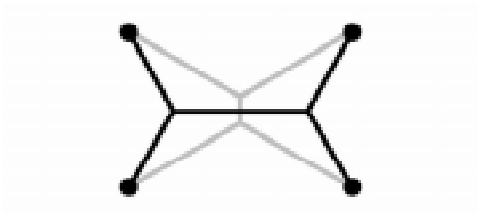

Рисунок 2.

Именно, выделяется множество вершин, расстояние до которых от i_0 находится в стадии вычисления. Это множество вершин принято обозначать через M_1. Все остальные вершины принадлежат либо множеству M_0 вершин, расстояние до которых уже вычислено, либо множеству M_2 вершин, расстояние до которых вычисляться не начиналось.

Первоначально $M_1 = \{i_0\}$, $M_2 = M \setminus M_1$, $M_0 = \varnothing$.

На каждом шаге итеративного процесса в M_1 выбирается вершина i_1, расстояние до которой минимально, эта вершина переводится в M_0, а затем она используется для корректировки других расстояний. Именно, путь до i_1 всевозможными способами наращивается на еще одну дугу (выходящую из i_1), получившиеся пути естественным образом используются для исправления тех данных о процессе, которые представлены множествами M_0, M_1, M_2.

... кратчайшие пути до заданной конечной вершины...

В реализации этого метода большое внимание уделяется эффективности задания множества M_1. Нужно, чтобы достаточно быстро реша-

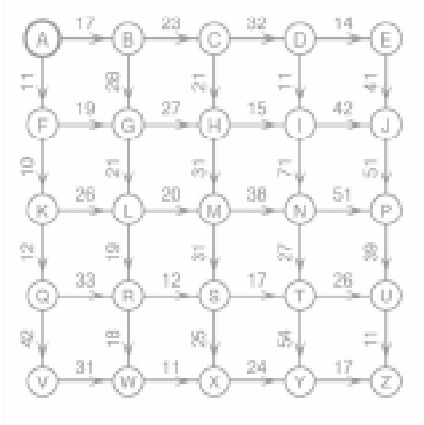

Рисунок 3. Исходный граф. Начальная вершина *A*, длины написаны около дуг.

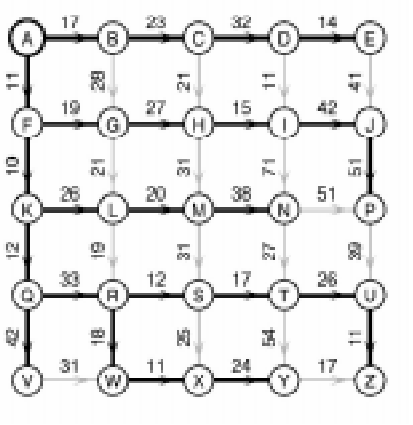

Рисунок 4. Дерево кратчайших путей от вершины *A* до остальных вершин.

лись задачи и его пополнения, и корректировки при уменьшении расстояния до вершины и поиска в нем вершины с наименьшим расстоянием.

Легко представить себе вариант метода Дейкстры, в котором граф как бы «инвертируется», то есть все дуги направляются в другую сторону. В терминах исходного графа это означает, что строить нужно кратчайшие пути до заданной конечной вершины i_C от всех остальных вершин графа.

4. ОБОБЩЕНИЕ МЕТОДА ДЕЙКСТРЫ

Метод Дейкстры можно применять и в случае, когда часть дерева путей уже вычислена.

Вычислительно такая модификация выглядит очень просто. Для определенности рассмотрим, что нужно сделать в инвертированном методе Дейкстры: при инициализации вычислительного процесса в множество M_1 загружаются вершины, расстояния до которых уже вычислены, и если пути, ведущие из таких вершин, изменять нельзя, то запрещается их модификация.

5. РЕШЕНИЕ ЗАДАЧИ ДЛЯ ДВУХ КОНЕЧНЫХ ТОЧЕК

Рассмотрим пример, в котором множество M_X состоит из двух точек, и на нем сначала опробуем эту идею. Пусть граф задан рисунке 3: в нем 25 вершин, 40 дуг, длины которых записаны рядом с дугами, начальной является вершина A, а множество $M_X = \{P, X\}$.

Применяя обычный метод Дейкстры, строим дерево кратчайших расстояний от *A* до всех вершин графа (рисунок 4).

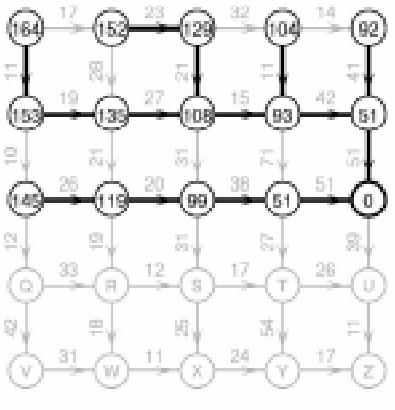

Рисунок 5. Дерево кратчайших путей до вершины *P*.

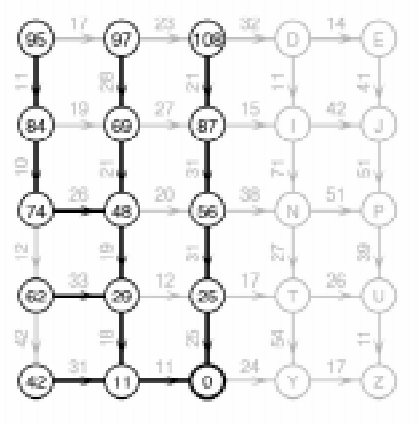

Рисунок 6. Дерево кратчайших путей до вершины *X*.

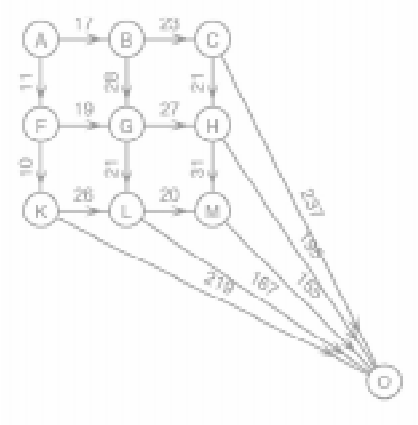

Рисунок 7. Граф избранных вершин с новыми дугами и вершиной *O*.

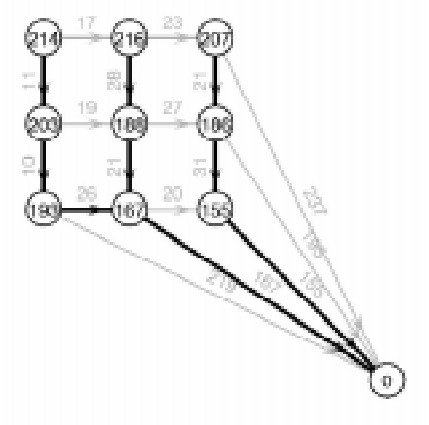

Рисунок 8. Дерево кратчайших путей в новом графе.

Но для нас сейчас будет важна возможность строить кратчайшие пути *до* заданной вершины *от* всех остальных. На рисунке 5 показано дерево кратчайших путей до вершины *P* от всех вершин, где такой путь существует. Естественно, что это условие исключает вершины, расположенные правее или ниже вершины *P*. В кружки, обозначающие вершины, вписаны кратчайшие расстояния от них до *P*. Аналогично строится и дерево кратчайших путей от всех вершин до вершины *X*, показанное на рисунке 6. В кружки, обозначающие вершины, вписаны кратчайшие расстояния от них до X.

Соединим теперь данные из рисунков 5 и 6, ограничившись вершинами, от которых есть пути до обеих вершин. Получим рисунок 7, в котором к этим «избранным» вершинам мы добавили еще одну, вершину *O*. В эту вершину идут дуги из всех имевшихся вершин (правда, на рисунке изображены только пять из них. Кстати, по какому принципу я отобрал эти пять дуг? На самом деле, в графе, который мы строим, должны быть дуги от любой из «избранных» вершин). Смысл новых дуг такой: исходные дуги графа – это дуги обычные, по которым мы добираемся обычным образом, причем, двигаясь по ним, мы движемся к обеим вершинам, в то время как каждая новая дуга символизирует способ передвижения из данной вершины в каждую из требуемых вершин отдельно. Эти отдельные пути сами по себе оптимальны:

Для получившегося графа методом Дейкстры находим дерево кратчайших путей, входящих в вершину *O*. Оно изображено на рисунке 8. Нам осталось перейти от нового графа к исходному.

В построенном дереве возьмем путь из вершины *A*. Он кончается «новой» дугой *LO*. Заменим эту дугу парой путей *LP* и *LX* в исходном графе. Мы получим частичное дерево, состоящее из путей *AL*, *LP* и *LX*. Оно изображено на рисунке 9.

6. ПРИНЦИПИАЛЬНОЕ РЕШЕНИЕ ДЛЯ НЕСКОЛЬКИХ КОНЕЧНЫХ ТОЧЕК

А что же делать в случае, когда вершин в множестве M_X больше двух. Случай трех вершин уже выглядит достаточно уг-

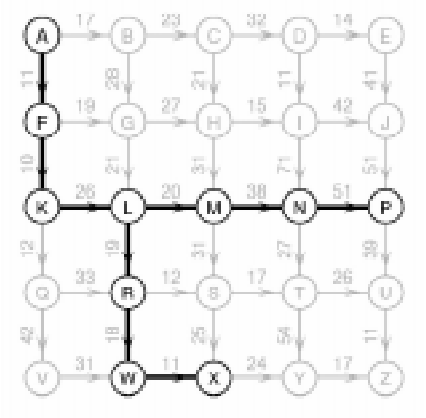

Рисунок 9. Оптимальное частичное дерево с путями из A в P и X.

Таблица 1.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	P	Q	R	S	T	U	V	W	X	Y	Z
P	™	™	™	™	™	™	™	™	™	™	™	™	™	™	™										
T	™	™	™	™		™	™	™	™		™	™	™	™		™	™	™	™						
X	™	™	™			™	™	™			™	™	™			™	™	™			™	™	™		
PT	™	™	™	™		™	™	™	™		™	™	™	™											
PX	™	™	™			™	™	™			™	™	™												
TX	™	™	™			™	™	™			™	™	™			™	™	™							

рожающе. Пусть для определенности $M_X = \{P, T, X\}$.

Порядок вычислений будет такой:

1) Решим задачу для вершин P, T и X.

2) Пользуясь полученными данными, решим, как раньше, задачу для пар $\{P, T\}$, $\{P, X\}$ и $\{T, X\}$. Чтобы не запутаться, введем, наконец-то, обозначения. Составим таблицу вычисленных расстояний $v[s, j]$, где s – подмножество множества M_X, а j – вершина. Можно схематически представить, что уже вычислено к данному моменту (см. таблицу 1).

3) (самое главное) Теперь мы будем вычислять для этой таблицы строку, соответствующую тройке $\{P, T, X\}$. Для этого впишем в каждую ячейку $v[M_X, j]$ значение $\min\{aP, aT, aX\}$, где

$$aP = v[P, j] + v[TX, j];$$
$$aT = v[T, j] + v[PX, j];$$
$$aX = v[X, j] + v[PX, j]$$

и после этого по методу Дейкстры «досчитаем» получившиеся расстояния.

...к этим «избранным» вершинам мы добавили еще одну...

Фактически у нас здесь получается схема динамического программирования, в которой состояниями являются всевозможные пары (s, j), а v – это функция Беллмана, удовлетворяющая рекуррентному соотношению

$$v[s, j] = \min\{aSplit, aMove\},$$

где введенные для удобства $aSplit$ и $aMove$ обозначают соответственно минимум, который берется по решениям, разбивающим множество s на непустые подмножества, и минимум по переходам в другие вершины с помощью дуги, выходящей из j:

$$aSplit = \min\{v[s', j] + v[s \setminus s', j] \mid s' \subset s\},$$
$$aMove = \min\{c_u + v[s, je(u)] \mid jb[u] = j\}.$$

Если угодно, можно записать это рекуррентное соотношение в более привычном виде

$$v[s, j] = \min\{\min\{\, v[s', j] + v[s \setminus s', j] \mid s' \subset s\},$$
$$\{c_u + v[s, je(u)] \mid jb[u] = j\}\}.$$

Именно это рекуррентное соотношение лежит в основе нашего расчета. Этот подход вполне приемлем, если память компьютера позволяет хранить таблицу размером $m \times 2^k$, где m – число вершин в графе, а k – число вершин в множестве M_X.

7. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Программная реализация предлагаемого алгоритма гораздо проще, чем это может показаться с первого взгляда. В ее основе лежит цикл по всем подмножествам множества M_X. Перебор подмножеств конечного множества легко реализуется как перебор характеристических векторов этих подмножеств, то есть всевозможных наборов из нулей и единиц. Если число k не слишком велико, то каждый такой набор

можно считать k-значным двоичным числом, так что перебор осуществляется простым циклом от 1 до $kSubset = 2^k$.

```
for iSubset := 1 to kSubset do begin
  Iteration
end;
```

Оператор **Iteration** состоит из двух частей: довычисление по методу Дейкстры строки **iSubset** матрицы v и использование вычисленной строки в подготовке данных для других строк.

```
Iteration ≡
  Dijkstrification;
  UseRow;
```

В методе Дейкстры на самом деле нам не потребуется вводить вершину O с примыкающими к ней новыми дугами: достаточно будет включить все вершины, для которых v уже начало вычисляться, в так называемое множество M_1.

Использование вычисленной строки, которая соответствует некоторому подмножеству *sCur* множества M_X заключается в том, что просматриваются все строки, вычисленные ранее. Каждая такая строка также соответствует некоторому подмножеству *sPrev*. Если подмножества *sCur* и *sPrev* дизъюнктны, то они составляют разбиение подмножества $sNew = sCur \cup sPrev$, являющегося их объединением. В таком случае значения $v[sNew, j]$ корректируются по сумме $v[sCur, j] + v[sPrev, j]$. На самом деле вся работа идет не с подмножествами, а с целыми числами, им соответствующими. Условие дизъюнктности подмножеств прекрасно записывается с помощью операции логического умножения этих чисел. Таким образом, мы имеем

```
UseRow ≡
 for iPrev := 1 to iSubset-1 do;
  if (iPrev and iSubset) = 0 then begin
   CorrectRow
  end;
```

Я не решился здесь же писать и корректировку, но она проста и мы не замедлим ее привести: здесь уже первый индекс массива – не подмножество, а соответствующее ему число.

```
CorrectRow ≡
 iNew := iPrev + iSubset;
  for all j do;
   if (v[iPrev,j]<infinity) and
       (v[iSubset,j]<infinity)
   then begin
     v1 := v[iPrev,j] + v[iSubset,j];
     if v[iNew,j] > v1
     then v[iNew,j] := v1;
   end;
```

Заголовок цикла по вершинам зависит от способа задания информации о графе.

Инициализация таблицы v очень проста: сначала вся таблица заполняется «бесконечным» значением, которое мы обозначили через **infinity**, а затем обнуляются k элементов $v[power(i),vert(i)]$, где $power(i) = 2^i$, а $vert(i)$ – истинный номер вершины, стоящей в M_X на месте i.

8. ЧТО ДАЛЬШЕ

Так можно решать задачи с небольшими значениями k; легко получается и можно чуть-чуть зайти за $k = 20$ на больших компьютерах. Между тем, для специальных типов графов можно сильно упростить решение, ограничивая набор рассматриваемых подмножеств.

Если исходный граф планарный, то есть если его можно без пересечений дуг

...достаточно будет включить все вершины, для которых v уже начало вычисляться...

расположить на плоскости и если все вершины множества находятся на «границе» области, содержащей граф, то можно рассматривать только подмножества, образующие сплошной участок границы.

Например, пусть в графе, показанном на рисунке 3, $M_X = \{ V, X, Z, P, E\}$. Вместо 31 подмножества, необходимого при общем подходе, здесь нам будет достаточно рассмотреть $k\,(k+1)/2 = 15$: $\{V\}$, $\{V, X\}$, $\{V, X, Z\}$, $\{V, X, Z, P\}$, $\{V, X, Z, P, E\}$, $\{X\}$, $\{X, Z\}$, $\{X, Z, P\}$, $\{X, Z, P, E\}$, $\{Z\}$, $\{Z, P\}$, $\{Z, P, E\}$, $\{P\}$, $\{P, E\}$, $\{E\}$.

Вопросы: В каком порядке следует рассматривать подмножества в этом частном случае? Какая нумерация подмножеств окажется здесь удобной:

В случае замкнутой границы число подмножеств несколько увеличивается и становится равным $k\,(k-1)+1$, но практически все возможности эффективного расчета сохраняются.

Было бы интересно предложить другие классы графов, в которых можно ограничить количество перебираемых подмножеств.

Литература

1. *Гэри М., Джонсон Д.* Вычислительные машины и трудно решаемые задачи. М.: Мир, 1982. (Здесь можно прочесть (совсем немного) про задачу Штейнера – исходную и на графах).

2. *Кормен,Т., Лейзерзон Ч., Ривест Р.* Алгоритмы: Построение и анализ. М: МЦНМО, 1999. (В этой классической книге есть изложение метода Дейкстры).

3. *Наумов Л. А.* Метод разбиения задач на подзадачи, рекурсия, «разделяй и властвуй». Динамическое программирование // Компьютерные инструменты в образовании, 2002, № 3-4. С. 94–106.

(Обзорная статья в нашем журнале. В ней подробно рассказано о динамическом программировании).

4. *Романовский И. В.* Дискретный анализ, 3-е изд., СПб: Невский диалект, 2003.

(Учебник, написанный автором данной статьи. В нем затрагиваются темы и динамического программирования, и метода Дейкстры, и (очень кратко) задачи Штейнера).

Романовский Иосиф Владимирович, доктор физико-математических наук, профессор СПбГУ.