Оптимизационные задачи

**Оптимизационные задачи**

Существует много задач, в которых:

* решения можно найти только с помощью перебора вариантов,
* каждому решению приписывается вес,
* требуется найти полное решение с минимальным (максимальным) весом.

Такие задачи относятся к оптимизационным.

**Пример** – задача о ранце:

Даны грузы  с общими стоимостями . Требуется так загрузить ранец с общей грузоподъемностью , чтобы стоимость загруженного была максимальной.

Непрерывный и дискретный варианты задачи о ранце.

Жадный алгоритм или перебор вариантов.

Если

* для любого частичного решения  некоторой задачи существует вес ,
* при переходе к следующему решению  для весов выполняется ,
* требуется найти решение с минимальным весом,

то при переборе вариантов можно использовать метод ветвей и границ.

**Идея метода ветвей и границ**:

1. На начальном этапе ищется какое-нибудь полное решение и вычисляется его вес. Данное решение и данный вес считаются текущими оптимальными  и .
2. Вес каждого текущего частичного решения  сравнивается с : если , то любые последующие решения, включающие , можно отбросить, как бесперспективные.
3. Если получено полное решение , для которого , то в качестве текущих оптимальных запоминаются  и .

Удачный выбор начального решения позволяет существенно сократить последующий перебор.

**Общий вид алгоритма**, использующего метод ветвей и границ (**S** – текущее решение, **C** – вес **S**, **Sopt** – текущее опт.решение, **Copt** – вес **Sopt**):

**поиск\_опт(S,C)**

**{**

**while (существует\_подходящий\_элемент(M,S,a))**

**{**

**добавить\_к\_текущему\_решению(S,a);**

**C = вес(S);**

**if (полное\_решение(S) && C < Copt)**

**{ Sopt = S; Copt = C; }**

**else if (C < Copt) поиск\_опт(S,C);**

**удалить\_из\_текущего\_решения(S,a);**

**C = вес(S);**

**}**

**}**

**Задача коммивояжера**

Задано  городов и матрица весов  ( – стоимость проезда из города  в город ).

Требуется найти минимальный по стоимости циклический маршрут, проходящий через все города ровно по одному разу.

(Аналогичная задача на графах – поиск минимального по весу гамильтонова цикла на полном взвешенном ориентированном графе).

Задача **симметрична**, если симметрична матрица весов, т.е.  . В общем случае задача несимметрична.

Для задачи выполняется **неравенство треугольника**, если  справедливо  (например, для точек на плоскости, если веса – это расстояния между ними).

**1-й вариант перебора путей:** добавление нового города к уже построенному частичному маршруту.

Идея: перебор последовательностей вершин (как в алгоритме генерации перестановок) с постоянной проверкой текущих решений по методу ветвей и границ.

В циклическом маршруте можно всегда начинать с 1-го города,

=>

Трудоемкость в наихудшем составляет .

Преимуществом данного подхода является простота алгоритма, основным недостатком – высокая трудоемкость.

**2-й вариант перебора путей**: на каждом шаге выбирается не новый город, а некоторый допустимый переход  из города  в город .

Дерево решений – двоичное, т.е. множество решений в любой вершине всегда разбивается на 2 подмножества: маршруты, содержащие переход  (левое поддерево), и маршруты, не содержащие  (правое поддерево).

Выбор перехода  производится так, чтобы вес путей для левого поддерева был как можно меньше, а для правого – как можно больше. При этом не должно возникать замкнутых путей, включающих менее  городов.

**Пример** с 7 городами. Матрица весов несимметрична,  (inf) для исключения соответствующих переходов.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  |
| 1 | inf | 3 | 93 | 13 | 33 | 9 | 57 | 3 |
| 2 | 4 | inf | 77 | 42 | 21 | 16 | 34 | 4 |
| 3 | 45 | 17 | inf | 36 | 16 | 28 | 25 | 16 |
| 4 | 39 | 90 | 80 | inf | 56 | 7 | 91 | 7 |
| 5 | 28 | 46 | 88 | 33 | inf | 25 | 57 | 25 |
| 6 | 3 | 88 | 18 | 46 | 92 | inf | 7 | 3 |
| 7 | 44 | 26 | 33 | 27 | 84 | 39 | inf | 26 |
|  |  |  | 7 | 1 |  |  | 4 | 96 |

Найдены минимальные элементы во всех строках, после их вычитания найдены и вычтены минимальные элементы, большие 0, во всех столбцах.

96 – нижняя граница стоимости решения (в корне дерева решений).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | inf | 0 | 83 | 9 | 30 | 6 | 50 |
| 2 | 0 | inf | 66 | 37 | 17 | 12 | 26 |
| 3 | 29 | 1 | inf | 19 | 0 | 12 | 5 |
| 4 | 32 | 83 | 66 | inf | 49 | 0 | 80 |
| 5 | 3 | 21 | 56 | 7 | inf | 0 | 28 |
| 6 | 0 | 85 | 8 | 42 | 89 | inf | 0 |
| 7 | 18 | 0 | 0 | 0 | 58 | 13 | inf |

Среди всех нулевых элементов выбираем (4,6) – 4-я строка содержит максимальный из минимальных ненулевых элементов (32), т.е. нижняя граница веса для всех путей, не включающих (4,6), составляет 96 + 32 = 128.

Матрица весов в корне правого поддерева (все пути, не содержащие (4,6)):

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  |
| 1 | inf | 0 | 83 | 9 | 30 | 6 | 50 |  |
| 2 | 0 | inf | 66 | 37 | 17 | 12 | 26 |  |
| 3 | 29 | 1 | inf | 19 | 0 | 12 | 5 |  |
| 4 | 32 | 83 | 66 | inf | 49 | inf | 80 | 32 |
| 5 | 3 | 21 | 56 | 7 | inf | 0 | 28 |  |
| 6 | 0 | 85 | 8 | 42 | 89 | inf | 0 |  |
| 7 | 18 | 0 | 0 | 0 | 58 | 13 | inf |  |

Матрица по-прежнему имеет размер 7х7, но появился новый элемент, равный inf.

Матрица весов в корне левого поддерева (все пути, содержащие (4,6)):

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  |
| 1 | inf | 0 | 83 | 9 | 30 | 6 | 50 |  |
| 2 | 0 | inf | 66 | 37 | 17 | 12 | 26 |  |
| 3 | 29 | 1 | inf | 19 | 0 | 12 | 5 |  |
| 4 | 32 | 83 | 66 | inf | 49 | inf | 80 |  |
| 5 | 3 | 21 | 56 | 7 | inf | 0 | 28 | 3 |
| 6 | 0 | 85 | 8 | 42 | 89 | inf | 0 |  |
| 7 | 18 | 0 | 0 | 0 | 58 | 13 | inf |  |

Матрица имеет размер 6х6, удалены строка 4 и столбец 6. (5,1) – это единственный минимальный элемент > 0, нижняя граница весов для левого поддерева равна 96 + 3 = 99.

На приведенном примере при 1-м варианте перебора (выбор городов) потребуется 559 проверок (6! = 720).

Для 2-го варианта будет проверено 27 узлов дерева решений,

оптимальный маршрут 1-4-6-7-3-5-2-1, его вес равен 126.

Общие трудоемкости для 2-го варианта перебора:

в наихудшем , в среднем .

Во многих практических случаях, если оптимизационная задача имеет слишком большую размерность, приходится искать субоптимальное (приемлемое) решение.

Такое решение имеет практическую ценность, если можно оценить его вес относительно веса оптимального решения.

**Алгоритм ближайшего соседа** для задачи коммивояжера:

* выбирается некоторый начальный город,
*  раз выполняется основной шаг – добавление к маршруту следующего города, ближайшего к последнему включенному.

При построении маршрута можно идти прямо (по строкам) и обратно (по столбцам).

Трудоемкость составляет . Перебор начальных городов увеличивает трудоемкость до , но иногда позволяет существенно улучшить результат.

Для предыдущего примера:

маршрут 1-2-6-7-4-5-3-1 имеет вес 242,

маршрут 3-5-6-1-2-4-7-3 имеет вес 174.

Если   и , то можно оценить качество получаемого маршрута  в сравнении с оптимальным :

.

Алгоритм ближайшего соседа – жадный по реализации, но не дающий оптимального решения.

Относительно невысокое его качество связано с тем, что выбор очередного города зависит только от города, включенного в маршрут последним. Соответственно, очередной город всегда добавляется в конец маршрута.

**Алгоритм ближайшего города**

Идея:

* начальный маршрут содержит 1 город (любой),
* маршрут всегда является циклическим (даже начальный).
* города добавляются в маршрут по одному,
* на каждом шаге добавляется город, ближайший к уже построенному маршруту (т.е. ближайший к некоторому городу, включенному в маршрут раньше).

Пусть на некотором шаге в маршруте всего  городов. Тогда в простейшем варианте алгоритма для поиска включаемого города потребуется  проверок пар городов.

=>

Общая трудоемкость составит .

Для снижения трудоемкости используется такой же метод, как в алгоритме Прима – формируется дополнительный массив длины :

, если город  включен в маршрут, или

, если  – ближайший к  город, уже включенный в маршрут.

На каждом шаге алгоритма:

1. Выбирается очередной город , такой что ,  ( элементарных шагов).

2.  включается в текущий циклический маршрут перед или после города  ( элементарных шагов).

3. Модифицируется массив :  и , если , то  ( элементарных шагов).

=> Общая трудоемкость алгоритма составляет .

Пусть   и  (задача симметрична и выполняется неравенство треугольника).

Рассмотрим процесс формирования маршрута  в алгоритме ближайшего города (синие ребра на рис.) как перестроение оптимального маршрута  (красные ребра, исключаемые из маршрута – штриховые,  отмечает переход с максимальным весом).

Получаемый маршрут – всегда циклический. К маршруту последовательно добавляются города, при этом один из переходов маршрута заменяется на 2 новых и, кроме того, исключается один из переходов оптимального маршрута (т.е. общее число переходов всегда равно ).

*v1*



*v1*

*v2*

*v1*

*v2*

*v3*

*v1*

*v2*

*v3*

*v4*

На каждом шаге образуется «паукообразная» конфигурация: замкнутый маршрут ближайшего города и отдельные незамкнутые части оптимального маршрута.

Пусть на некотором шаге в маршрут ближайшего города между городами  и  добавляется город  (удаляется переход  и добавляются  и ). Кроме того, удаляется некоторый переход  из оптимального маршрута.

*i*

*a*

*j*

*k*

*b*

 (\*) по алгоритму, т.к.  – ближайший к маршруту город,

 по неравенству треугольника

=>  (\*\*).

Суммируем неравенства (\*) и (\*\*) и переносим  в левую часть:

, т.е. увеличение веса маршрута ближайшего города при добавлении нового города не превышает удвоенного веса удаляемого ребра оптимального маршрута.

Следовательно, , где  – максимальный вес перехода в оптимальном маршруте.

Маршрут ближайшего города всегда не хуже маршрута ближайшего соседа.

Маршрут ближайшего города зависит от выбора позиции для включения нового города. Рассмотрим  точек на окружности:

1

2

4

5

3

6

1

2

4

5

3

6

Порядок включения переходов (цветные позднее удаляются):

(1,2) (2,1) (1,2) (2,1)

(2,3) (3,1) (1,3) (3,2)

(3,4) (4,1) (2,4) (4,3)

(4,5) (5,1) (3,5) (5,4)

(5,6) (6,1) (4,6) (6,5)

Если развернуть точки на прямую, то  при .

**Модификации алгоритма ближайшего города**

1. Если на очередном шаге добавляется город  как ближайший к входящему в маршрут городу , то нужно включать  либо до, либо после , минимизируя приращение общего веса маршрута.

2. Для города , пока не вошедшего в маршрут, оценивать не расстояние до города  в маршруте, а изменение общего веса маршрута при включении  до или после . Учитывать это при модификации массива .

3. Для **задачи на плоскости** с весами, соответствующими длинам переходов, устранить все пересечения переходов.

*a*

*b*

*v*

*w*

 по неравенству треугольника.

При удалении пересечения в маршрут включаются 2 новых перехода, для которых также необходима проверка на пересечение с другими переходами маршрута.

**Поиск маршрута коммивояжера на основе выделения минимального остова**

Пусть задача коммивояжера симметрична и выполняется неравенство треугольника.

Рассмотрим полный взвешенный неориентированный граф : вершины – города, ребра – переходы, веса ребер – веса переходов.

Выделим на  минимальное остовное дерево  и, начиная с любой вершины, построим циклический (двойной) обход данного дерева с помощью поиска в глубину (по каждому ребру мы пройдем дважды – при рекурсивном спуске и подъеме).

(На рисунке указаны номера вершин в порядке первого прихода в них; остов – красный, двойной обход – синий.)

1

2

3

4

5

6

7

8

9

10

11

Перестроим двойной обход таким образом, чтобы получаемый циклический маршрут  проходил ровно по 1 разу через каждую вершину (обход вершин в порядке 1-го прихода в них при поиске в глубину):

1

2

3

4

5

6

7

8

9

10

11

Вес двойного обхода дерева равен  (каждое ребро остова входит дважды).

В силу неравенства треугольника .

Оптимальный маршрут, из которого удалено любое ребро, – это остов, поэтому .

Следовательно, .

Трудоемкость данного алгоритма составляет .

Вместо минимального остова в качестве основы для построения маршрута коммивояжера можно использовать паросочетание с минимальным весом (множество ребер, не имеющих общих вершин, но инцидентных всем вершинам).

Вес получаемого маршрута , трудоемкость .

**Применение динамического программирования для поиска маршрута коммивояжера**

Для задачи коммивояжера выполняются основные условия применения динамического программирования.

1. **Свойство оптимальности для подзадач**: любая подпоследовательность в оптимальном решении оптимальна. Если  – оптимальный маршрут, то минимальный по весу путь из  в  среди всех путей, проходящих по разу через произвольную последовательность узлов  – это именно .

2. **Перекрывающиеся подзадачи**: при использовании бэктрекинга и метода ветвей и границ производится проверка частичных решений, которые могут иметь совпадающие множества элементов (но лишь одно из таких решений может входить в оптимальную последовательность).

3. Общее число частичных решений экспоненциально, но число различных оптимальных подпоследовательностей полиномиально (т.е. их можно хранить в некоторой таблице).

В соответствии с порядком решения задачи с помощью ДП необходимо:

* найти рекуррентное соотношение, связывающее оптимальные решения подзадач разных уровней;
* двигаясь снизу вверх, от подзадач самого низкого уровня, вычислять их оптимальные решения только один раз и сохранять результаты в специальной таблице;
* использовать данные из таблицы при поиске оптимального решения подзадач следующего уровня.

Обозначим через  вес оптимального маршрута из  в , проходящего в точности через  в любом порядке.

Тогда для последовательности оптимальных маршрутов из  в  выполняется:

, ,

.

Требуется найти .

Для сохранения промежуточных оптимальных решений требуется много памяти, трудоемкость в наихудшем остается экспоненциальной, но часто удается сократить количество вычислений.