Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ Фундаментальные науки

КАФЕДРА Прикладная математика

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Задача об ограниченном ранце

Студент ФН2-42Б (Группа) __________ (Подпись, дата) З. И. Абрамов (И. О. Фамилия)

Руководитель курсовой работы __________ (Подпись, дата) А. Ю. Попов (И. О. Фамилия)

2022 г.

Оглавление

Введение . . . 3

1. Постановка задачи . . . 4

2. Метод ветвей и границ . . . 4

3. Реализация метода ветвей и границ . . . 7

3.1. Особенности реализации на языке C++ . . . 7

3.2. Особенности реализации на Wolfram Mathematica . . . 7

4. Метод динамического программирования . . . 7

5. Реализация метода динамического программирования . . . 9

5.1. Особенности реализации на языке C++ . . . 9

5.2. Особенности реализации на Wolfram Mathematica . . . 9

6. Примеры работы программ . . . 9

Заключение . . . 9

Список использованных источников . . . 9

Введение

Классическая задача о рюкзаке (о загрузке) известна очень давно, и формулируется следующим образом. Пусть есть n разных предметов, каждый предмет имеет вес w_i и полезность c_i, так же имеется максимальный вес W, который можно положить в рюкзак. Требуется собрать такой набор предметов, чтобы полезность их была наибольшей, а суммарный вес не превышал W. Несмотря на такую "бытовую" формулировку, решение данной задачи находит гораздо более широкое применение.

Задача о загрузке (или задача о рюкзаке) и различные ее модификации широко применяются на практике в прикладной математике, криптографии, экономике, логистике, для нахождения решения оптимальной загрузки различных транспортных средств: самолетов, кораблей, железнодорожных вагонов и т.д. Приведем в качестве примера задачу, которая представляет собой переиначенную задачу о ранце, чтобы показать, что алгоритмы для решения данной проблемы могут быть полезны не только в упаковке багажа:

Технология DVS[1] позволяет снижать напряжение на процессоре и добиваться экономии электроэнергии за счет увеличения времени выполнения задачи.

Пусть процессор поддерживает два уровня напряжения — $U_L < U_H$. Есть набор из n задач, каждая из которых имеет энергоемкость и время выполнения для обоих режимов, т.е. для $i = 1, \ldots, n$ даны энергоемкости c_i^H и c_i^L и длительности t_i^H и t_i^L.

Нужно выполнить все задачи на одном процессоре за время не более заданного T, при этом добиться минимального энергопотребления.

Рассматриваемая нами задача является *NP*-полной, то есть для нее не существует полиномиального алгоритма, решающего ее за разумное время, в этом и заключается основная проблема данной задачи. Решения *NP*-полных задач обычно делят на два типа: приближенные, работающие быстро, но далеко не всегда решающие задачу наилучшим образом, и точные, выполняемые сильно дольше, но гарантированно выдающие лучший результат.

Существует несколько модификаций задачи о ранце. Ниже приведены некоторые из них.

1) Каждый предмет можно брать только один раз.
2) Каждый предмет можно брать сколько угодно раз.
3) Каждый предмет можно брать определенное количество раз.
4) Можно брать дробную часть предмета.

[1] DVS (от англ. Dynamic Voltage Scaling) — динамическое изменение напряжения.

Цель данной работы — реализовать решение для задачи 3) двумя точными алгоритмами, а именно методом ветвей и границ и динамическим программированием, на языке программирования C++ и в системе компьютерной алгебры Wolfram Mathematica. Также исследовать время выполнения и сложность алгоритмов.

1. Постановка задачи

Математически задачу об ограниченном рюкзаке можно представить следующим образом:

$$\begin{gathered}\sum_{i=1}^{n} c_i x_i \to \max, \\ \sum_{i=1}^{n} w_i x_i \leqslant W, \\ x_i \in \{0, 1, \ldots, k_i\},\end{gathered} \tag{1}$$

где c_i, w_i и k_i — стоимость, вес и максимальное количество предмета i соответственно; W — максимальная грузоподъемность ранца; n — количество предметов.

Очевидно, что сложность алгоритма, осуществляющего полный перебор, т.е. проверяющего всевозможные наборы предметов, пропорциональна количеству этих возможные наборов. Таким образом он имеет сложность $O(K^n)$, где $K = \max\limits_{i=1,\ldots,n} k_i$, что очень неэффективно. Поэтому, как и во многих других задачах, есть методы решения задачи, которые выполняются быстрее обычного перебора.

2. Метод ветвей и границ

Метод ветвей и границ (МВГ) относится к числу основных методов решения задач оптимизации. Этот метод основан на древовидной декомпозиции исходной задачи на подзадачи.

В процессе дальнейшего изложения мы будем часто употреблять термин рекорд, означающий наилучшее найденное значение целевой функции к данному шагу алгоритма. Предполагается, что рекорд, также как и целевая функция, принимает действительные значения.

Различные варианты МВГ имеют специфичные для решаемой задачи особенности, но при этом подчиняются следующей общей схеме [4]:

Данные: рекорд, список подзадач.

Шаг 1. В список подзадач помещается исходная задача.

Шаг 2. Если список подзадач пуст, то завершить алгоритм. В противном случае из списка выбирается и удаляется подзадача P.

Шаг 3. Вычисляется значение целевой функции, и при необходимости обновляется значение рекорда. Для задачи P проверяется выполнимость условия отсева. Если подзадача P удовлетворяет условию отсева, то осуществляется переход к шагу 2.

Шаг 4. Задача P подвергается декомпозиции. Полученные в результате подзадачи помещаются в список подзадач. Перейти к шагу 2.

Т.е. конкретный вариант метода ветвей и границ определяется следующими тремя функциями: выбором подзадачи из списка; условием отсева; правилом декомпозиции, определяющим, каким образом разбивается очередная подзадача.

Опишем данные функции для задачи об ограниченном ранце.

Для начала запишем линейную задачу релаксацию, соответствующей задаче (1):

$$\begin{aligned} &\sum_{i=1}^{n} c_i x_i \to \max, \\ &\sum_{i=1}^{n} w_i x_i \leqslant W, \\ &0 \leqslant x_i \leqslant k_i. \end{aligned} \tag{2}$$

Оптимум задачи (2) не меньше оптимума исходной задачи (1). Задача релаксации представляет собой одномерную задачу линейного программирования и может быть решена методом Данцига следующим образом. Переменные нумеруются в порядке возрастания удельной стоимости:

$$\frac{c_1}{w_1} \geqslant \ldots \geqslant \frac{c_n}{w_n}.$$

Сначала определяется номер s дробной переменной по следующему правилу:

$$s = \min\left\{ j \in \{1, \ldots, n\} : \sum_{i=1}^{j} k_i w_i > W \right\}.$$

Таким образом решение состоит из предметов $i = 1, \ldots, (s-1)$ в количестве k_i и предмета s в количестве

$$K = \frac{W - \sum\limits_{i=1}^{s-1} k_i w_i}{w_s},$$

а стоимость итогового набора задачи (2)

$$S = \sum_{i=1}^{s-1} k_i c_i + K \cdot c_s.$$

Данный алгоритм имеет сложность $O(n)$. На основании данного решения введем функцию $S(i, w)$, решающую аналогичную задачу для предметов $j = i, \ldots, n$ и вместимости рюкзака w, т.е. S, определенное выше, для задачи (2) есть значение $S(1, W)$.

Теперь сформулируем, что будем понимать под подзадачей в рамках загрузки рюкзака. Пусть мы зафиксировали количество у первых m предметов. Данный набор имеет стоимость C' и вес W'. Тогда под подзадачей будем называть следущее:

$$\begin{gathered} C' + \sum_{i=m+1}^{n} c_i x_i \to \max, \\ \sum_{i=m+1}^{n} w_i x_i \leqslant W - W', \\ x_i \in \{0, 1, \ldots, k_i\} \text{ при } i = (m+1), \ldots, n. \end{gathered} \tag{3}$$

Теперь мы готовы, наконец, сформулировать три функции, характеризующие метод ветвей и границ.

Правило декомпозиции. Исходную подзадачу (3) будем разбивать на $(k_{m+1} + 1)$ подзадач, фиксируя в каждой новой такой задаче предмет $(m + 1)$ в количестве $0, 1, 2, \ldots, k_{m+1}$ раз. Отметим, что если в подзадаче (3) присутствует всего один предмет n, то нет смысла разбивать ее на подзадачи, потому что мы можем сразу дать на нее ответ.

Выбор подзадачи из списка. В основном, выбор подзадачи можно сделать произвольной, но мы будем пользоваться логикой жадных алгоритмов. Чтобы задача релаксации выполнялась за $O(n)$ необходимо, чтобы предметы изначально были отсортированы по убыванию удельной стоимости, поэтому будем предполагать, что предметы изначально отсортированы. Таким образом можно предположить, что нам выгодней взять как можно больше первых предметов. Поэтому после декомпозиции мы сначала будем рассматривать задачу при условии, что мы взяли предмет $(m+1)$ в количестве k_{m+1}, потом $(k_{m+1} - 1)$ и т.д.

Условие отсева. Подзадача не подлежит дальнейшей декомпозиции если:

1) подзадача не имеет решения, что возможно только при условии $W' < 0$;
2) оптимум задачи (2) не превосходит значения рекорда (тогда, как было отмечено выше, оптимум задачи (3) тоже не превосходит значения рекорда).

Сложность алгоритма метода ветвей и границ составляет, как и при переборе, $O(K^n)$, где $K = \max\limits_{i=1,\ldots,n} k_i$, т.к. может произойти ситуация, что условие отсева выполнено для каждой подзадачи и мы будем вынуждены рассмотреть все возможные подзадачи данной подзадачи.

3. Реализация метода ветвей и границ

3.1. Особенности реализации на языке C++

3.2. Особенности реализации на Wolfram Mathematica

4. Метод динамического программирования

Словосочетание "динамическое программирование" впервые было использовано в 1940-х годах Ричардом Беллманом[1] для описания процесса нахождения решения задачи, где ответ на одну задачу может быть получен только после решения задачи, "предшествующей" ей. В 1953 году он уточнил это определение до современного. Вклад Беллмана в динамическое программирование был увековечен в названии уравнения Беллмана, центрального результата теории динамического программирования, который переформулирует оптимизационную задачу в рекурсивной форме.

Слово "программирование" в словосочетании "динамическое программирование" в действительности к "традиционному" программированию (написанию кода) почти никакого отношения не имеет. Слово "программа" в данном контексте скорее означает оптимальную последовательность действий для получения решения задачи. К примеру, определенное расписание событий на выставке иногда называют программой.

Динамическое программирование — способ решения сложных задач путем разбиения их на более простые подзадачи. Ключевая идея в динамическом программировании достаточно проста. Как правило, чтобы решить поставленную задачу, требуется решить отдельные части задачи (подзадачи), после чего объединить решения подзадач в одно общее решение. Часто многие из этих подзадач одинаковы. Подход динамического программирования состоит в том, чтобы решить каждую подзадачу только один раз, сократив тем самым количество вычислений. Это особенно полезно в случаях, когда число повторяющихся подзадач экспоненциально велико.

Чтобы успешно решить задачу динамикой нужно:

1) состояние динамики: параметр(ы), однозначно задающие подзадачу;
2) значения начальных состояний;
3) переходы между состояниями: формула пересчета;
4) порядок пересчета;

[1] Ричард Эрнест Беллман (26 августа 1920, Нью-Йорк, США – 19 марта 1984, Лос-Анджелес, США) — американский математик, один из ведущих специалистов в области математики и вычислительной техники. Член Национальной инженерной академии США (1977), Национальной академии наук США (1983).

5) положение ответа на задачу: иногда это сумма или, например, максимум из значений нескольких состояний.

Для решения нашей задачи введем двумерный массив dp размером $n+1$ на $W+1$, где в $dp[i,w]$ будем хранить максимально возможную стоимость набора состоящая из первых i предметов при максимальной допустимой нагрузке w. Теперь распишем пункты выше.

1) состояние динамики: $dp[i,w]$;
2) начальные состояния: $dp[0,w]$ и $dp[i,0]$ равны нулю, т.е. если у нас нет никаких вещей или мы не можем ничего положить в рюкзак, то и максимально возможная стоимость равна нулю;
3) формула пересчета: $dp[i,w] = \max\limits_{x_j=0,1,\ldots,m(i,w)} (x_j \cdot c_i + dp[i-1, w - x_j \cdot w_i])$, где $m(i,w)$ — максимальное количество предмета i, которое влезает в рюкзак вместимости w, ограниченное при этом максимальным количеством k_i, таким образом $m(i,w) = \min\left\{k_i;\ \left\lfloor \frac{w}{w_i} \right\rfloor\right\}$;
4) пересчитывать будем в следующем порядке: для каждого i будем заполнять строку $dp[i,w]$, w изменяется в диапазоне от 1 до W, т.е. постепенно будем добавлять каждый предмет по порядку;
5) ответ будет находится в ячейке $dp[n,W]$, т.к. это именно та задача, которую нам надо решить, для всех n предметов при емкости рюкзака W.

Таким образом, заполнение массива dp можно представить в виде следующего псевдокода:

for $w = 0$ **to** W **do**
    $dp[0,w] = 0$
for $i = 1$ **to** n **do**
    for $w = 1$ **to** W **do**
        $dp[i,w] = dp[i-1,w]$
        for $x_j = \min\left\{k_i, \left\lfloor \frac{w}{w_i} \right\rfloor\right\}$ **downto 1 do**
            $dp[i,w] = \max\left\{dp[i,w];\ x_j \cdot c_i + dp[i-1, w - x_j \cdot w_i]\right\}$

Такое решение работает за $O(n\,W^2)$, так как k_i могут быть очень большими, а $w_i = 1$.

Теперь опишем восстановление одного из решений также в виде псевдокода.

$w = W$
$solution = \{\ \}$
for $i = n$ **downto** 1 **do**
    if $w = 0$ **then**
        break

6: Находим $mVal = \max\limits_{x_j=0,1,\ldots,m(i,w)} (x_j \cdot c_i + dp[i-1, w - x_j \cdot w_i])$
и $mCnt = x_j$, на котором достигается $mVal$
7: **if** $mCnt \neq 0$ **then**
8: Добавляем в решение *solution* предмет i в количестве $mCnt$ штук
9: w -= $w_i \cdot mCnt$

Впоследствии выполнения алгоритма итоговое решение будет представлено массивом *solution*. Сложность алгоритма восстановления одного решения составляет $O(n)$, потому что в общем случае мы пройдем по всем n предметам.

В итоге алгоритм имеет временную сложность $O(n\,W^2)$, а по памяти $O(n\,W)$, которая требуется на хранение массива *dp*. Также заметим, что если требуется найти только максимальную стоимость набора, не восстанавливая само решение, то потребление памяти можно сократить до $O(W)$, храня только две последние рассматриваемые строчки.

5. Реализация метода динамического программирования

5.1. Особенности реализации на языке C++

5.2. Особенности реализации на Wolfram Mathematica

6. Примеры работы программ

Заключение

Список использованных источников

1. Kempka S.N., Glass M.W., Peery J.S., Strickland J.H., Ingber M.S. Accuracy considerations for implementing velocity boundary conditions in vorticity formulations // SANDIA report. SAND96-0583, UC-700. 1996. 52 p.