Федеральное государственное бюджетное образовательное учереждение
высшего
профессионального образования
«Уфимский государственный авиационный технический университет»

Пояснительная записка к курсовой работе
по дисциплине: «Методы Оптимизации»
на тему: «Задача о рюкзаке: Задача об отправке грузов».

Выполнил:
Пахтусов Н. Г., ПРО-306
Проверила:
Валеева А. Ф.

Уфа, 2015 г.

Содержание

Введение

Для множества задач в прикладной математике нахождение решения прямым перебором за приемлимое время невозможно. К таким задачам относится, например, класс NP-полных задач.

Одной из задач этого класса является так называемая «Задача о рюкзаке». Задача о рюкзаке – одна из NP-задач комбинаторной оптимизации. Своё название она получила от максимизационной задачи укладки как можно большего числа ценных вещей в рюкзак при условии, что вместимость рюкзака ограничена. С различными вариациями задачи о ранце можно столкнуться в экономике, прикладной математике, криптографии, генетике и логистике.

В работе рассмативется одна из разновидностей этой задачи – «Задача об отправке грузов».

1. Постановка задачи

Пусть существует некоторое *количество* авиалайнеров и некоторое *количество* контейнеров. У каждого контейнера есть свой *вес*, а у авиалайнеров есть *ограничение по суммарному весу* контейнеров.

Пусть также различна *выгода* от отправки различными авиалайнерами одного и того же контейнера.

Задача состоит в том, чтобы перевезти контейнеры на авиалайнерах с *максимальной выгодой*.

2. Математическая модель

Пусть $I = \{1, \cdots, n\}$ – авиалайнеры, $J = \{1, \cdots, m\}$ – контейнеры.

p_{ij} – доход от доставки авиалайнером i контейнера j.

w_j – вес контейнера j.

c_i – вместимость авиалайнера i.

$x_{ij} \in \{0, 1\}$ – количество контейнеров j в авиалайнере i.

Таким образом, необходимо найти:

$$\sum_{i=0}^{n} \sum_{j=0}^{m} p_{ij} x_{ij} \to max$$

При ограничениях:

$$\sum_{i=0}^{n} x_{ij} \leq 1, j \in J$$

$$\sum_{j=0}^{m} w_j x_{ij} \leq c_i, i \in I.[2]$$

3. Методы решения задачи

Для решения задач о рюкзаке используется несколько различных эвристических и оптимизационных. Рассмотрим некоторые из них.

3.1 Полный перебор

Временная сложность алгоритма O(N!), т.е он работоспособен для небольших значений N. С ростом N задача становится неразрешимой данным методом за приемлемое время.

3.2 Метод ветвей и границ

Метод ветвей и границ (англ. *branch and bound*) – общий алгоритмический метод для нахождения оптимальных решений различных задач оптимизации, особенно дискретной и комбинаторной оптимизации. По существу, метод является вариацией полного перебора с отсевом подмножеств допустимых решений, заведомо не содержащих оптимальных решений.

Общая идея метода может быть описана на примере поиска минимума функции `f(x)` на множестве допустимых значений переменной `x`. Функция `f` и переменная x могут быть произвольной природы. Для метода ветвей и границ необходимы две процедуры: ветвление и нахождение оценок (границ).

Процедура ветвления состоит в разбиении множества допустимых значений переменной x на подобласти (подмножества) меньших размеров. Процедуру можно рекурсивно применять к подобластям. Полученные подобласти образуют дерево, называемое деревом поиска или деревом ветвей и границ. Узлами этого дерева являются построенные подобласти (подмножества множества значений переменной `x`).

Процедура нахождения оценок заключается в поиске верхних и нижних границ для решения задачи на подобласти допустимых значений переменной `x`.

В основе метода ветвей и границ лежит следующая идея: если нижняя граница значений функции на подобласти `A` дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти `B`, то `A` может быть исключена из дальнейшего рассмотрения (правило отсева).

Если нижняя граница для узла дерева совпадает с верхней границей, то это значение является минимумом функции и достигается на соответствующей подобласти.

3.3 Жадный алгоритм

В жадном алгоритме (greedy algorithm) всегда делается выбор, который кажется самым лучшим в данный момент – т.е. производится локально оптимальный выбор в надежде, что он приведет к оптимальному решению глобальной задачи[3].

Согласно жадному алгоритму предметы сортируются по убыванию стоимости единицы веса каждого. В рюкзак последовательно складываются самые дорогие за единицу веса предметы из тех, что помещаются внутри.

Сложность сортировки предметов $O(N \log_2(N))$. Далее происходит перебор всех `N` элементов.

Жадный алгортим является эвристическим, таким образом, точное решение можно получить не всегда (жадный алгоритм всегда даёт точное решение, если структура задачи задается матроидом: тогда применение жадного алгоритма выдаст глобальный оптимум, однако наша задача к ним не относится)[1].

4. Алгоритм решения задачи

Для решения задачи был выбран «**Жадный алгоритм**».

Входные данные: на вход алгоритму подаётся массив значений весов грузов $w_j j \in \{1 \ldots J\}$, массив вместимости контейнеров $c_i, i \in \{1 \ldots I\}$ и матрица стоимости $I \times J$.

Выходные данные: общая стоимость всех выбранных грузов.

Введём некоторые структуры данных, которые будут использоваться в алгоритме:

- `Cargo` – структура, использующаяся для представления одного груза, с двумя полями:

    1. Поле `is_used` $\in \{1, 0\}$, изначально значение $= 0$, если предмет уже был обработан, значение изменяется на 1.
    2. Вес предмета w_j.

- `Knapsack` – структура, использующаяся для представления одного вместилища, с полями:

    1. Поле-список, хранящий некоторые значения $j \in J$ и означающим то, что j предмет лежит в данном вместилище.
    2. Поле-значение, означающее максимальный возможный вес для этого контейнера c_i.

- `Cost` – структура, использующаяся для хранения стоимости. Она хранит следующие значения: $i \in I$ и $j \in J$, которые означают i вместилище и j груз, а также выгоду отправки p_{ij}.

Таким образом, алгоритм будет состоять из следующих шагов:

1. Сформировать из входных данных массивы структур `Cargo`, `Knapsack` и `Cost` длин i, j и $i \times j$ соответственно.

2. Отсортировать массив структур `Cost` по стоимости в порядке убывания.

3. Для каждого $k \in \{1 \dots i \times j\}$ повторять:

```
i = информация из Cost[k] о номере вместилища;
j = информация из Cost[k] о номере груза;
Если Cargo[j].is_used = 0:
    w = вес Cargo[j];
    prev_w_sum = вес уже положенных в Knapsack[i] грузов;
    leftover = Knapsack[i].c - prev_w_sum;
    Если w =< leftover:
        положить j в список принадлежности Knapsack[i];
        Cost[j].is_used = 1;
    Иначе:
        продолжить цикл;
Иначе:
    продолжить цикл;
```

4. Посчитать сумму стоимостей получившегося набора.

5. Напечатать результат.

5. Программная реализация алгоритма

Для реализации был выбран язык программирования **Haskell**.

Введём некоторые абстракции над типами данных и создадим необходимые программные представления для структур `Cargo`, `Knapsack` и `Cost`:

```haskell
type W = Int
type P = Int
type I = Int
type J = Int
type Cargo = (J, Bool, W)
type Knapsack = (I, [J], W)
type Cost = (I, J, P)
```

Создадим некоторые вспомогательные функции:

- Функции для удобного взятия первого, второго и третьего элемента кортежа соответственно:

```haskell
mfst (x, _, _) = x
msnd (_, x, _) = x
mthd (_,_,x) = x
```

- Функция для обновления элемента с индексом `i` на элемент `el`:

```
substitude i el xs = take i xs ++ [el] ++ drop (i + 1) xs
```

- Функция для создания списка элементов `Cargo` из входного списка весов `w`:

```
makeCargo = makeCargo' 0
        where
            makeCargo'::Int -> [W] -> [Cargo]
            makeCargo' _ [] = []
            makeCargo' n (w:ws) = (n, False, w) : (makeCargo' (n + 1) ws)
```

- Функция для создания списка элементов `Knapsack` из входного списка весов `c`:

```
makeKnapsack = makeKnapsack' 0
        where
            makeKnapsack'::Int -> [C] -> [Knapsack]
            makeKnapsack' _ [] = []
            makeKnapsack' n (c:cs) = (n, [], c) : (makeKnapsack' (n + 1) cs)
```

- Функция для создания списка `Cost` из матрицы выгоды:

```
makeCost::[[P]] -> [Cost]
makeCost = makeCostI 0
            where
        makeCostI _ [] = []
        makeCostI i (x:xs) = (makeCostIJ i 0 x) ++ (makeCostI (i + 1) xs)
                where
            makeCostIJ i j [] = []
            makeCostIJ i j (w:ws) = (i, j, w) : (makeCostIJ i (j + 1) ws)
```

Реализуем основной алгоритм:

```
findSoluton cargo ks ((i,j,p):cs)
                        | msnd (cargo !! j) = findSoluton cargo ks cs
                        | (mthd (ks !! i)) > mthd (cargo !! j) = let
                                (_, _, cargoW) = cargo !! j
                                newCargo = substitude j (0, True, 0) cargo
                                (ki, clst, w) = ks!!i
                                newKnapsack = substitude i (ki, (j:clst), (w - cargoW)) ks
                                in findSoluton newCargo newKnapsack cs
                        | otherwice = findSoluton cargo ks cs
```

6. Тестовые примеры

Протестируем алгоритм на следующих данных:

Пусть у нас будут веса $W = \{1, 3, 4, 5, 7, 6\}$, вместилища $C = \{10, 10, 7\}$ и матрица стоимостей C:

$$C = \begin{pmatrix} 1 & 2 & 3 & 2 & 5 & 8 \\ 4 & 10 & 6 & 3 & 4 & 7 \\ 7 & 8 & 9 & 2 & 9 & 3 \end{pmatrix}.$$

Начнём алгоритм:

1. Сформируем стоимости:

   Knapsack = [(0,0,1),(0,1,2),(0,2,3),(0,3,2),(0,4,5),
   (0,5,8),(1,0,4),(1,1,10),(1,2,6),(1,3,3),
   (1,4,4),(1,5,7),(2,0,7),(2,1,8),(2,2,9),
   (2,3,2),(2,4,9),(2,5,3)].

2. Отсортируем в порядке убывания стоимостей: Knapsack = [(1,1,10),(2,2,9),(2,4
   (1,5,7),(2,0,7),(1,2,6),(0,4,5),(1,0,4),
   (1,4,4),(0,2,3),(1,3,3),(2,5,3),(0,1,2),
   (0,3,2),(2,3,2),(0,0,1)].

3. Начнем основной цикл:

   а) Положили в рюкзак 1 предмет 1 с весом 3 осталось места: 7
   б) Положим в рюкзак 2 предмет 2 с весом 4 осталось места: 3
   в) Не Положим в рюкзак 2 предмет 4 с весом 7, так как осталось места: 3
   г) Предмет 1 уже лежит в каком-то рюкзаке
   д) Предмет 5 уже лежит в каком-то рюкзаке
   е) Положим в рюкзак 0 предмет 5 с весом 6 осталось места: 4
   ж) Предмет 2 уже лежит в каком-то рюкзаке
   з) Положим в рюкзак 2 предмет 0 с весом 1 осталось места: 2
   и) Не Положим в рюкзак 0 предмет 4 с весом 7, так как осталось места: 4
   к) Предмет 0 уже лежит в каком-то рюкзаке
   л) Предмет 2 уже лежит в каком-то рюкзаке
   м) Положим в рюкзак 1 предмет 4 с весом 7 осталось места: 0
   н) Не Положим в рюкзак 1 предмет 3 с весом 5, так как осталось места: 0
   о) Предмет 5 уже лежит в каком-то рюкзаке
   п) Предмет 1 уже лежит в каком-то рюкзаке

р) Не Положим в рюкзак 0 предмет 3 с весом

с) 5, так как осталось места: 4

т) Не Положим в рюкзак 2 предмет 3 с весом

у) 5, так как осталось места: 2

ф) Предмет 0 уже лежит в каком-то рюкзаке

4. Посчитаем результат: стоимость всех сложенных предметов будет 38.

Изменим алгоритм. Уберем из него действие 3.

1. Сформируем стоимости:
   Knapsack = [(0,0,1),(0,1,2),(0,2,3),(0,3,2),(0,4,5),
   (0,5,8),(1,0,4),(1,1,10),(1,2,6),(1,3,3),
   (1,4,4),(1,5,7),(2,0,7),(2,1,8),(2,2,9),
   (2,3,2),(2,4,9),(2,5,3)].
2. Начнем основной цикл:

а) Положили в рюкзак 0 предмет 0 с весом 1 осталось места: 9

б) Положили в рюкзак 0 предмет 1 с весом 3 осталось места: 6

в) Не Положим в рюкзак 2 предмет 4 с весом 7, так как осталось места: 3

г) Положили в рюкзак 0 предмет 2 с весом 4 осталось места: 2

д) Не положили в рюкзак 0 предмет 3 с весом 5, так как осталось места: 2

е) Не положили в рюкзак 0 предмет 4 с весом 7, так как осталось места: 2

ж) Не положили в рюкзак 0 предмет 5 с весом 6, так как осталось места: 2

з) Предмет 0 уже лежит

и) Предмет 1 уже лежит

к) Предмет 2 уже лежит

л) Положили в рюкзак 1 предмет 3 с весом 5 осталось места: 5

м) Не положили в рюкзак 1 предмет 4 с весом 7, так как осталось места: 5

н) Не положили в рюкзак 1 предмет 5 с весом 6, так как осталось места: 5

о) Предмет 0 уже лежит

п) Предмет 1 уже лежит

р) Предмет 2 уже лежит

с) Предмет 3 уже лежит

т) Положили в рюкзак 2 предмет 4 с весом 7 осталось места: 0

у) Не положили в рюкзак 2 предмет 5 с весом 6, так как осталось места: 0

3. Посчитаем результат: стоимость всех сложенных предметов будет 18.

Заключение

Очевидно, что жадный аогоритм сделал перевозку предметов гораздо эффективнее, с его помощью стоимость всех предметов получилась 38, а простым алгоритмом вышло всего 18. Однако, так как жадный алгоритм является ивристическим, а не точным, то возможно существует ещё более выгодное решение.

Список использованных источников

1. В. Липский, Комбинаторика для программистов, с. 174. Изд. Москва «Мир» 1988г.

2. http://www.math.nsc.ru/LBRT/k5/TPR/lec4.pdf

3. Томас Х. Кормен, Алгоритмы: построение и анализ. Изд. дом «Вильямс».