Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»

Інститут Прикладного системного аналізу
Кафедра Системного проектування

Лабораторна робота №3

з дисципліни «Теорія прийняття рішень»
«Прийняття рішень в умовах повної інформації»

Виконала:
студентка групи ДА-42
Балан Катерина

Київ - 2017

Мета роботи

Ознайомитись з методами прийняття рішень в умовах повної інформації на прикладі задачі про упакування в контейнери та дослідити особливості їх використання

Варіант №1

№	C	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	83	86	77	15	93	35	86	92	49	21	62	27	90	59	63	26	40	26	72	36
	2	11	38	67	29	82	30	62	23	67	35	29	02	22	58	59	67	93	56	11	42
	3	29	73	21	19	84	37	98	24	15	70	13	26	91	80	56	73	62	70	96	81

Короткі теоретичні відомості

Задача про упакування в контейнери відноситься до NP–важких комбінаторних задач. Завдання полягає в упаковці об'єктів зумовленої форми в кінцеве число контейнерів зумовленої форми таким чином, щоб кількість використаних контейнерів була найменшою. Іноді розглядають зворотну задачу, щоб кількість або обсяг об'єктів, які упаковують, були найбільшими.

Існує безліч різновидів цієї задачі (двовимірна упаковка, лінійна упаковка, упаковка по вазі, упаковка по вартості і т.і.), які можуть застосовуватися в різних областях, наприклад, в задачі оптимального заповнення контейнерів, завантаження вантажівок з обмеженням по вазі, створення резервних копій на змінних накопичувачах і т.і.

Оскільки задача є NP-важкою, часто використовують алгоритми з евристичним та метаевристичним методом вирішення для отримання оптимальних результатів. Також активно використовуються методи штучного інтелекту, як, наприклад, нейронні мережі.

Математична постановка задачі

Розглянемо умову задачі. Дано:

– нескінчена кількість контейнерів розміром C;

– перелік вантажів розміром (вагою) c_i.

$$x_i^j = \begin{cases} 1, \textit{якщо і-й об'єкт пакується в j-й контейнер} \\ 0, \textit{в іншому випадку} \end{cases} \quad (2.1)$$

Бінарна змінна y_j дорівнює 1, якщо j-й контейнер використовується, та 0 в

протилежному випадку.

Необхідно так упакувати вантажі в контейнери, щоб загальна кількість контейнерів була найменшою

$$\min \sum_{j=1}^{n} y_j$$

при наступних обмеженнях

$$\sum_{i=1}^{N} c_i x_i^j \le C y_j,\ j = 1(1)n$$

$$\sum_{j=1}^{n} x_i^j = N, i = 1(1)N$$

$$x_i^j \in \{0,1\}, i = 1(1)N,\ j = 1(1)n$$

$$y_j \in \{0,1\},\ j = 1(1)n$$

Перша нерівність фіксує, що розмір (вага) вантажів в одному контейнері не буде більшою, ніж його розмір (вантажопідйомність). Друга рівність потребує, щоб всі елементи були розміщені по контейнерах.

Для вирішення цієї задачі розглянемо наступні алгоритми:

– NFA (Next Fit Algorithm) – алгоритм «заповнення наступного»;
– FFA (First Fit Algorithm) – алгоритм «заповнення першого, що підходить»;
– WFA (Worst Fit Algorithm) – алгоритм заповнення «найменш повного»;
– BFA (Best Fit Algorithm) – алгоритм заповнення «найкращого».

Хоча всі алгоритми націлені на мінімізацію кількості контейнерів, але для різних алгоритмів є відмінності в пошуку контейнера і, відповідно в кількості обчислень, що потребується для цього.

Виконання роботи

Результати розрахунку

Дані	Аналітичний розрахунок (кількість контейнерів)
1 рядок	11
2 рядок	9
3 рядок	11

1+2+3 рядок	31							
Дані	Кількість контейнерів				Обчислювальна складність			
	Без впорядкування				Без впорядкування			
	NFA	FFA	WFA	BFA	NFA	FFA	WFA	BFA
1 рядок	15	14	14	14	20	127	143	130
2 рядок	12	11	11	11	20	85	101	91
3 рядок	17	14	14	14	20	121	138	125
1+2+3 рядок	42	36	37	36	60	820	937	900
Дані	З впорядкуванням				З впорядкуванням			
	NFA	FFA	WFA	BFA	NFA	FFA	WFA	BFA
1 рядок	14	13	13	13	20	140	162	154
2 рядок	12	10	10	10	20	104	134	134
3 рядок	14	12	12	12	20	142	188	190
1+2+3 рядок	41	33	34	33	60	1130	1301	1535

Результати роботи програми

```
***************
DATA: [83, 86, 77, 15, 93, 35, 86, 92, 49, 21, 62, 27, 90, 59, 63, 26, 40, 26, 72, 36]
***************
NFA: [[83], [86], [77, 15], [93], [35], [86], [92], [49, 21], [62, 27], [90], [59], [63, 26], [40, 26], [72],
[36]]
Comparations: 20
Conteiners: 15
***************
FFA: [[83], [86], [77, 15], [93], [35, 49], [86], [92], [21, 62], [27, 59], [90], [63, 26], [40, 26], [72],
[36]]
Comparations: 127
Conteiners: 14
***************
WFA: [[83], [86], [77, 15], [93], [35, 49], [86], [92], [21, 62], [27, 59], [90], [63, 26], [40, 26], [72],
[36]]
Comparations: 143
Conteiners: 14
***************
BFA: [[83], [86], [77, 15], [93], [35, 49], [86], [92], [21, 62], [27, 59], [90], [63, 26], [40, 26], [72],
[36]]
Comparations: 130
Conteiners: 14
--------------------
SORTED:
***************
NFA: [[93], [92], [90], [86], [86], [83], [77], [72], [63], [62], [59], [49, 40], [36, 35, 27], [26, 26, 21,
15]]
Comparations: 20
Conteiners: 14
***************
FFA: [[93], [92], [90], [86], [86], [83], [77], [72, 27], [63, 36], [62, 35], [59, 26], [49, 40], [26, 21,
15]]
Comparations: 140
Conteiners: 13
***************
WFA: [[93], [92], [90], [86], [86], [83], [77], [72, 26], [63, 27], [62, 35], [59, 36], [49, 40], [26, 21,
15]]
Comparations: 162
Conteiners: 13
***************
BFA: [[93], [92], [90], [86], [86], [83], [77], [72, 27], [63, 36], [62, 35], [59, 26], [49, 40], [26, 21,
15]]
Comparations: 154
Conteiners: 13
Process finished with exit code 0
```

Лістинг програми

https://github.com/katebalan/Decision_theory/blob/master/lab_tpr3/something.py

```
# laboratory work #3 for Decision theory
# Next Fit Algorithm
def next_fit(boxes, container_capacity, sort = False):
  if sort:
    boxes = sorted(boxes, reverse=True)
  containers = [[]]
  containers_count = 0
  sum_container = 0
  comparison_count = 0
  for box in boxes:
    sum_container += int(box)
    comparison_count += 1
    if sum_container <= container_capacity:
      containers[containers_count].append(box)
    else:
      sum_container = int(box)
      containers.append([])
      containers_count += 1
      containers[containers_count].append(box)
  print "*" * 15
  print "NFA: {}".format(containers)
  print "Comparations: {}".format(comparison_count)
  print "Conteiners: {}".format(len(containers))
# First Fit Algorithm
def first_fit(boxes, container_capacity, sort = False):
  if sort:
    boxes = sorted(boxes, reverse=True)
  containers = [[]]
  containers_count = 0
  containers_weight = [0]
  comparison_count = 0
  for box in boxes:
    comparison_count += 1
    if (containers_weight[containers_count] + box) <= container_capacity:
      containers[containers_count].append(box)
      containers_weight[containers_count] += box
    else:
      placed = False
      for i in range(containers_count):
        comparison_count += 1
        if (containers_weight[i] + box) <= container_capacity:
          containers[i].append(box)
          containers_weight[i] += box
          placed = True
          break
      comparison_count += 1
      if not placed:
        containers.append([])
        containers_count += 1
        containers[containers_count].append(box)
        containers_weight.append(box)
  print "*" * 15
  print "FFA: {}".format(containers)
  print "Comparations: {}".format(comparison_count)
  print "Conteiners: {}".format(len(containers))
# Worst Fit Algorithm
def worst_fit(boxes, container_capacity, sort = False):
  if sort:
    boxes = sorted(boxes, reverse=True)
  containers = [[]]
  containers_count = 0
  containers_weight = [0]
  comparison_count = 0
  for box in boxes:
    comparison_count += 1
    if (containers_weight[containers_count] + box) <= container_capacity:
      containers[containers_count].append(box)
      containers_weight[containers_count] += box
    else:
      min_weight = min(containers_weight)
      comparison_count += len(containers_weight)
      min_index = containers_weight.index(min_weight)
      comparison_count += 1
```

```
    if (min_weight + box) <= container_capacity:
      containers[min_index].append(box)
      containers_weight[min_index] += box
    else:
      containers.append([])
      containers_count += 1
      containers[containers_count].append(box)
      containers_weight.append(box)
  print "*" * 15
  print "WFA: {}".format(containers)
  print "Comparations: {}".format(comparison_count)
  print "Conteiners: {}".format(len(containers))
# Best Fit Algorithm
def best_fit(boxes, container_capacity, sort = False):
  if sort:
    boxes = sorted(boxes, reverse=True)
  containers = [[]]
  containers_count = 0
  containers_weight = [0]
  comparison_count = 0
  for box in boxes:
    comparison_count += 1
    if (containers_weight[containers_count] + box) <= container_capacity:
      containers[containers_count].append(box)
      containers_weight[containers_count] += box
    else:
      best_fit = []
      for iter in range(containers_count):
        comparison_count += 1
        if containers_weight[iter] + box <= container_capacity:
          best_fit.append(containers_weight[iter])
      comparison_count += 1
      if not best_fit:
        containers.append([])
        containers_count += 1
        containers[containers_count].append(box)
        containers_weight.append(box)
      else:
        max_weight = max(best_fit)
        comparison_count += len(best_fit)
        max_index = containers_weight.index(max_weight)
        containers[max_index].append(box)
        containers_weight[max_index] += box
  print "*" * 15
  print "BFA: {}".format(containers)
  print "Comparations: {}".format(comparison_count)
  print "Conteiners: {}".format(len(containers))
```

Висновки :

З огляду на сумарні результати, можна зробити висновок, що при попередньому впорядкуванні отримуємо кращі результати, але ціною більшої обчислювальної складності. Щодо самих алгоритмів, то на даній множині контейнерів, незважаючи на різницю в обчислювальній складності, алгоритми FFA, WFA показали однакові результати, тому у даному випадку найкращим є алгоритм FFA як найменш вимогливий до ресурсів ЕОМ. Алгоритм NFA ще менш вимогливий, але дає на порядок гірші результати за кількістю контейнерів.