Hausarbeit Rucksackproblem

Das Rucksackproblem

Diskrete Mathematik
Dualen Hochschule Baden-Württemberg
Stuttgart

von
Paul Walker und Tom Hofer

Matrikelnummer:	3610783, 4775319
Abgabedatum:	30.06.2022

Inhaltsverzeichnis

Abkürzungsverzeichnis

1 Einleitung

2 Problemstellung

Ein Rucksack hat eine bestimmte Tragekapazität (z.b. 6 Kg) ein Dieb packt bei einem Wohnungsraub sein Diebesgut in diesen Rucksack. In der Wohnung befinden sich Gegenstände mit unterschiedlichem Gewicht und Geldwert. Gegenstände können verschieden oft vorhanden sein (z.b. Schmuck 4kg 500€ 1stk, Elektrogeräte 3kg 400€ 3stk, Schuhe 2kg 300€ 1stk, Geld 1kg 200€ 2stk). Der Dieb möchte bei dem Wohnungsraub einen möglichst großen Gewinn erzielen, er möchte also den Geldwert der in den Rucksack gepackten Gegenstände maximieren, da der Rucksack aber nur eine bestimmte Kapazität hat, muss zuerst ein Optimierungsproblem gelöst werden. Dieses Problem nennt sich das *Rucksackproblem* (oder Englisch: *Knapsack Problem*)

In diesem Beispiel wäre die beste Kombination: 1stk Geld, 1stk Schuhe, 1stk Elektrogeräte damit wird ein Gesamtgewinn von 900€ und einem Gesammtgewicht von 6kg erreicht.

Mathematisch formuliert: Es gibt $m \in \mathbb{N}$ Gegenstände. Sei $c_i \in \mathbb{N} : i \in I$ der Wert des Gegenstandes i, $a_i \in \mathbb{N} : i \in I$ das Gewicht des Gegenstandes i und $u_i \in \mathbb{N} : i \in I$ wie oft der Gegenstand vorhanden ist jeweils mit $i \in I = \{1, 2, \ldots, m\}$. Die Tragekapazität des Rucksacks ist $b \in \mathbb{N}$. Dann wird für den maximal erreichbaren Geldwert in Abhängigkeit zur Tragekapazität des Rucksacks $f(b)$ definiert:

$$f(b) := \max(\sum_{i=1}^{m} c_i x_i : \sum_{i=1}^{m} a_i x_i \leq b, 0 \leq x_i \leq u_i, x_i \in \mathbb{N}, i \in I)$$

Das hier dargestellte Rucksackproblem ist ein begrenztes Rucksackproblem. Beim unbegrenzten Rucksackproblem kann x_i jeden beliebigen wert in $\mathbb{N}$ annehmen und ist nicht von u_i begrenzt.

Das am häufigsten auftretende Rucksackproblem ist das 0-1-Rucksackproblem. Ein Begrenztes Rucksackproblem kann zu einem 0-1-Rucksackproblem vereinfacht werden, indem $u_i = 1$ gesetzt wird. Dann gilt:

$$f(b) := \max(\sum_{i \in I} c_i : \sum_{i \in I} a_i \leq b, i \in I)$$

Anschaulich bedeutet das, dass jeder Gegenstand nur genau einmal vorhanden ist und daher auch nur einmal mitgenommen werden kann. Da das 0-1-Rucksackproblem das Häufigste in der Kategorie der Rucksackprobleme ist, soll der Fokus im Folgenden auf dem 0-1-Rucksackproblem liegen.

Das Rucksackproblem gehört zur Kategorie der am schwersten zu lösenden Probleme, den NP-vollständigen Problemen [1]. In der Praxis gibt es aber einige Lösungsverfahren und einige Approximationsverfahren.

2.1 NP-vollständige Probleme

Zusätzlich zur Problemklasse NP gibt es auch die Klasse P. P enthält alle Probleme, welche durch einen polynomiellen Algorithmus gelöst werden können. Die Frage ob $P = NP$ gilt, konnte bisher weder bestätigt, noch wiederlegt werden. Die NP-vollständigen Probleme sind die schwersten Probleme der Klasse NP, da sie genau dann in polinomieller Zeit gelöst werden können, wenn $P = NP$ gilt. [2, Kap. 15]

3 Anwendungen

Das Rucksackproblem tritt in der realen Welt häufiger bei Optimierungsproblemen, speziell bei Problemen in der Ressourcen Allokation auf. Man denke beispielsweise an einen LKW, der ein bestimmtes Transportvolumen hat und Güter, die bei verschiedenem Volumen einen unterschiedlichen Gewinn erzielen. Oder ein Containerschiff das ein bestimmtes Gewicht Tragen kann und Container mit unterschiedlichem Gewicht und Gewinn.

Zusätzlich zu den praktischen Anwendungen gibt es auch theoretische Anwendungen für das Rucksackproblem. Rucksackprobleme sind ganzzahlige lineare Optimierungsprobleme. Andere ganzzahlige lineare Optimierungsprobleme können in Rucksackprobleme umgeformt werden[1], die umgeformte Version des Problems hat dabei dieselben Lösungen wie das Originalproblem[1]. Das heißt es können die bekannten Lösungsverfahren zur Lösung des Rucksackproblems auch bei anderen ganzzahligen linearen Optimierungsproblemen verwendet werden.

Eine Problemvariante des Rucksackproblems, die Subset sum wird in der Kryptographie verwendet, beispielsweise beim Merkle-Hellman-Kryptosystem (das sich nicht als besonders sicher herausstellte).

4 Lösungsansätze

4.1 Brute Force

Bei n zur Auswahl stehender Elemente, gibt es 2^n Möglichkeiten verschiedene Kombinationen von Gegenständen daraus zu wählen. Bei jeder Itteration wird dann überprüft, ob die jetzige Kombination von Gegenständen die beste bisher ist. Ist sie das, wird die bisher beste Kombination ersetzt und der neue beste Wert gespeichert [4]. Der Algorithmus hat damit eine Komplexität von $O(n2^n)$. Er ist damit der schlechtest mögliche Algorithmus.

4.2 Algorhitmus von Gilmore und Gomoroy

Das im Folgenden vorgestellte Verfahren funktioniert zur Lösung des 0-1-Rucksackproblems. Der Algorhitmus von Gilmore un Gomoroy basiert auf *dynamischer Optimierung*. Dafür wird das Rucksackproblem zuerst Vollständig reduziert und dann schrittweise wieder zum Originalproblem aufgebaut. Begonnen wird mit dem

wert	gewicht	index
500€	4kg	1
400€	3kg	2
300€	2kg	3
200€	1kg	4

Tabelle 1: Werte

	0kg	1kg	2kg	3kg	4kg	5kg	6kg
0	0€	0€	0€	0€	0€	0€	0€
1	0€	0€	0€	0€	500€	500€	500€
2	0€	0€	0€	400€	500€	500€	500€
3	0€	0€	300€	400€	500€	700€	800€
4	0€	200€	300€	500€	600€	700€	900€

Tabelle 2: Lösungsmatrix

simpelsten Problem: einem Rucksack mit Kapazität 0 und mit einem Gegenstand. Die Kapazität wird schrittweise erhöht bis die Originalkapazität erreicht ist. Es wird bei jeder Erhöhung der Kapazität geprüft, ob der Gegenstand hinzugefügt werden kann und ob dies einen Vorteil gegenüber der vorherigen Iteration bringt. Der Gegenstand wird hinzugefügt oder nicht basierend darauf was am meisten gewinn erzielt. Dann wird der nächste Gegenstand genommen und die Kapazität des Rucksacks wieder auf 0 gesetzt. Wichtig ist dabei, dass die Gegenstände im Wert absteigend ausgewählt werden. Nun wird wieder mit der Erhöhung der Rucksack Kapazität begonnen. Das wird so lange wiederholt, bis versucht wurde alle Gegenstände hinzuzufügen [3]. Der maximal erzielbare Wert im Rucksack steht im unteren rechten Feld der Matrix, und ist das Ergebnis des Algorithmus.

In 1 ist dieser Algorithmus beispielhaft in Python implementiert. Die Tabelle 2 zeigt das Ergebnis dieses Algorithmus für die Eingabe aus 1. Um herauszufinden welche Gegenstände in den Rucksack aufgenommen wurden, muss die Matrix einfach schrittweise zurückverfolgt werden.

Code-Auszug 1: Gilmore Gomoroy in Python

```python
from numpy import zeros

def gilmore_gomoroy(v, c):
  """
  Parameters:
  v (array) list of (value, weight) tuples
  c (int) knapsack capacity

  Returns
  m (array): solution matrix
  """

```

```
  # Matrix with solution
  m = zeros((len(v)+1, c+1))

  for i in range(1, len(v)+1):
    for j in range(1, c+1):
      weight = v[i-1][1]
      value = v[i-1][0]
      if weight > j:
        m[i, j] = m[i-1, j]
      else:
        m[i, j] = max(m[i-1, j], m[i-1, j-weight] + value)

  return m
```

Der Algorithmus hat die Komplexität $O(nc)$, wobei c die maximale Kapazität des Rucksacks darstellt. Algorithmen mit einer solchen Komplexität werden *pseudo-polynomial* genannt [3].

4.3 Algorithmus von Bellman und Dantzig

Der Algorithmus von Bellman und Dantzig ist ein pseudopolynomieller Algorithmus für das ganzzahlige Rucksackproblem. Tabelle 3 zeigt die Ein- und Ausgabewerte des Algorithmus.

1. C ist eine beliebige obere Schanke für den Optimalwert
2. $x(0,0) := 0, x(0,k) := \infty$ für $k = 1, ..., C$
3. Berechnune die Teilmengen:
   for j in range(1,n):
       for k in range(0,C):
           s(j,k)=0
           x(j,k)=x(j-1,k)
       for k in range(c_j,C):
           if $x(j-1, k-c_j) + w_j <= min\{W, x(j,k)\}$:
               $x(j,k) = x(j-1, k-c_j) + w_j$
               $s(j,k) = 1$
4. Wähle beste zulässige Teilmenge:
   $k = \max\{i \in \{0, ..., C\} : x(n,i) < \infty\}$
   S={}
   for j in range(n,1,-1):
       if s(j,k)==1:
           $S = S \cup \{j\}$

Eingabe	$n, c_1, ...c_n, w_1, ..., w_n$ und W, jeweils aus $\mathbb{N}$
Ausgabe	Eine Teilmenge $S \subseteq \{1, ..., n\}$ mit $\sum_{j \in S} w_j \leq W$, die $\sum_{j \in S} c_j$ maximiert

Tabelle 3: Ein- und Ausgabe des Algorithmus

$k = k - c_j$

Dieser Algorithmus hat eine Laufzeit von O(nW). Durch den Trick $C := \sum_{j \in S}^{n} c_j$ zu setzten erhält man eine Laufzeit von O(nC). Diese O(nC) Schranke ist nicht polynomiell in der Eingabegröße, jedoch erweist sich dieser Algorithmus in der Praxis, für kleinere Zahlen, als effizient. [2, Kap. 17]

4.4 Annäherungsalgorithmen

4.4.1 Greedy Algorhitmus

Bei vielen Optimierungsproblemen wird ein Greedy Algorithmus eingesetzt, um das Problem zu lösen oder die Lösung zumindest gut annähern zu können [4].

Für das 0-1-Rucksackproblem Problem gibt es mehrere Greedy Strategien.

1. Schritt für Schritt Gegenstände mit absteigendem Wert wählen und in den Rucksack Packen.
2. Schritt für Schritt Gegenstände mit aufsteigendem Gewicht in den Rucksack Packen.
3. Schritt für Schritt Gegenstände mit aufsteigendem Wert pro Gewicht in den Rucksack Packen.

Dabei erzielt die letzte Strategie in der Praxis die besten Ergebnisse [4].
Die Zeitkomplexität dieses Algorithmus ist $O(n \log n)$

Literatur

[1] Ernst-Peter Beisel. *Verfahren zur Lösung von 0-1-Rucksack Problemen. Theorie und Implementierung*. 2010. URL: http://www2.math.uni-wuppertal.de/~beisel/Rucksack/mainKnapsack.pdf (besucht am 21. 06. 2022).

[2] Bernhard Korte und Jens Vygen, Hrsg. *Kombinatorische Optimierung*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2018. ISBN: 978-3-662-57690-8. DOI: 10.1007/978-3-662-57691-5.

[3] Guntram Scheiterhauer. *Zuschnitt- und Packungsoptimierung. Problemstellungen, Modellierungstechniken, Lösungsmethoden*. Hrsg. von Ulrich Sandten & Kerstin Hoffmann. Vieweg+Teubner, 2008. ISBN: 978-3-8351-0215-6.

[4] Maya Hristakeva & Dipti Shrestha. *Different Approaches to Solve the 0/1 Knapsack Problem*. URL: http://micsymposium.org/mics_2005/papers/paper102.pdf (besucht am 23. 06. 2022).