Allianz

DHBW
Duale Hochschule
Baden-Württemberg

DAS RUCKSACKPROBLEM

HAUSARBEIT

Studiengang Informatik
an der Dualen Hochschule
Baden-Württemberg Stuttgart

von Alexander Regemann, Sven Sendke

Abgabedatum: 15.07.2024

Kurs:	TINF22F	Matrikelnummer:	4296627 & 8469950
Modul:	Diskrete Mathematik	Dozent:	Dipl.-Math. Christian Kratochwil

Abstract

In dieser Arbeit wird das Rucksackproblem als eines der prominentesten Probleme der kombinatorischen Optimierung untersucht. Das Rucksackproblem zeichnet sich durch seine breite Anwendbarkeit in Bereichen wie Logistik, Finanzplanung und Produktionsoptimierung aus, wobei es um die Auswahl einer Teilmenge von Objekten mit maximalem Gesamtwert innerhalb einer festen Gewichtsbeschränkung geht. Trotz seiner scheinbaren Einfachheit stellt das Rucksackproblem eine erhebliche rechnerische Herausforderung dar, da es zur Klasse der NP-schweren Probleme gehört und daher in der Regel nicht in Polynomialzeit lösbar ist. Die vorliegende Arbeit untersucht verschiedene Lösungsansätze, darunter dynamische Programmierung, Greedy-Algorithmen und heuristische Verfahren wie ”First Fit Decreasing”. Anhand eines praktischen Beispiels wird das Problem nochmals verdeutlicht. Darüber hinaus werden mögliche Erweiterungen und zukünftige Forschungsrichtungen diskutiert, die das Potential haben, die Lösungsstrategien für das Rucksackproblem weiter zu verbessern.

Inhaltsverzeichnis

Verzeichnisse

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

FFD First Fit Decreasing

1. Einleitung

Das Rucksackproblem ist eines der bekanntesten und am häufigsten untersuchten Probleme in der Informatik und der mathematischen Optimierung. Es gehört zu den klassischen NP-schweren Problemen und findet Anwendung in den verschiedensten Anwendungsgebieten, von der Logistik über die Finanzplanung bis hin zur Produktionsplanung. Die Grundidee des Rucksackproblems besteht darin, eine begrenzte Kapazität eines Rucksacks optimal zu nutzen, um Gegenstände von unterschiedlichem Wert und unterschiedlicher Größe auszuwählen.

1.1 Problemstellung

Das Rucksackproblem birgt trotz seiner scheinbaren Einfachheit eine erhebliche Schwierigkeit, die sich aus seiner exponentiellen Komplexität ergibt. Die Suche nach der optimalen Kombination von Gegenständen, um die Kapazität des Rucksacks zu maximieren, erfordert eine umfassende Untersuchung aller möglichen Kombinationen. Diese exponentiell wachsende Anzahl von Kombinationen macht das Problem selbst mit modernen Computerressourcen schwierig. Effiziente Algorithmen versuchen, eine Schätzung des Ergebnisses zu erhalten, die nahe an der optimalen Lösung liegt. Solche Algorithmen, wie z.B. die dynamische Programmierung oder heuristische Ansätze, können die benötigte Rechenzeit erheblich reduzieren und praktikable Lösungen in akzeptabler Zeit liefern.

1.2 Ziel und Struktur der Hausarbeit

Ziel dieser Hausarbeit ist es, das Rucksackproblem vorzustellen, mögliche Lösungsansätze zu betrachten und ein vertieftes Verständnis für die Komplexität des Rucksackproblems und die verschiedenen Lösungsstrategien zu vermitteln. Abschließend soll das Problem anhand eines Beispiels untersucht werden.

2. Grundlagen des Rucksackproblems

Wir werden uns nun mit dem Rucksackproblem befassen.

2.1 Definition

Das Rucksackproblem befasst sich mit der Frage, wie eine begrenzte Anzahl von Gegenständen mit jeweils einem bestimmten Gewicht und einem bestimmten Wert so in einen Rucksack mit begrenzter Tragfähigkeit gepackt werden kann, dass der Gesamtwert der darin enthaltenen Gegenstände maximiert wird. Zur Veranschaulichung des Problems dient die grafische Darstellung des Rucksackproblems (siehe Abbildung 2.1).

Abbildung 2.1: Rucksackproblem anhand eines Beispiels [1]

Das Problem ist NP-vollständig und kann mathematisch wie folgt formuliert werden: Gegeben sei eine Menge U von Objekten. Jedem Objekt $u \in U$ wird ein Gewicht $w(u)$ und ein Nutzenwert $v(u)$ zugeordnet. Es gibt außerdem eine Gewichtsschranke $B \in \mathbb{R}$.
Gesucht ist eine Teilmenge $K \subseteq U$, die die Bedingung

$$\sum_{u \in K} w(u) \leq B$$

einhält und die Zielfunktion

$$\sum_{u \in K} v(u)$$

maximiert. [2]

2.2 Komplexitätsanalyse

Das Rucksackproblem gehört zur Komplexitätsklasse NP (nichtdeterministische Polynomialzeit). Dies bedeutet, dass das Problem mit einem deterministischen Algorithmus nicht in Polynomialzeit gelöst werden kann, sofern P ≠ NP. Eine Lösung in pseudopolynomialer Zeit kann jedoch z.B. mit Hilfe der dynamischen Programmierung gefunden werden. [3] Im Folgenden sollen nun drei Lösungsmöglichkeiten betrachtet werden.

3. Dynamisches Programmieren

Im ersten Schritt wird der Ansatz der dynamischen Programmierung betrachtet.

3.1 Prinzip des dynamischen Programmierens

Dynamische Programmierung basiert auf der Idee, ein Problem in kleinere Teilprobleme zu zerlegen und deren Lösungen zu speichern, um sie später wiederzuverwenden. Durch systematisches Kombinieren der Lösungen der Teilprobleme kann das Gesamtproblem effizient gelöst werden. Dieser Ansatz ist besonders nützlich bei Problemen mit überlappenden Teilstrukturen, bei denen viele Teilprobleme mehrfach auftreten. Die dynamische Programmierung ermöglicht eine optimale Lösung, indem es die Gesamtoptimalität aus lokalen optimalen Lösungen konstruiert wird. [4]

3.2 Implementierung des dynamischen Programmierens für das Rucksackproblem

Algorithm 1 Dynamisches Programmieren für das Rucksackproblem

Input: Liste von Gegenständen mit Gewicht w_i und Wert v_i, maximales Gewicht W des Rucksacks
Output: Maximale Wertsumme, die in den Rucksack passt
Initialisiere ein 2D-Array dp mit Dimensionen $(n+1) \times (W+1)$ mit allen Einträgen auf 0, wobei n die Anzahl der Gegenstände ist
for $i \leftarrow 1$ to n **do**
    for $j \leftarrow 0$ to W **do**
        if $w_i \leq j$ **then**
            $dp[i][j] \leftarrow \max(dp[i-1][j], dp[i-1][j-w_i] + v_i)$
        else
            $dp[i][j] \leftarrow dp[i-1][j]$
        end if
    end for
end for
return $dp[n][W]$

3.3 Vor- und Nachteile

Vorteile:

- Optimale Lösungen: Es garantiert die optimale Lösung durch systematisches Durchsuchen aller möglichen Teilprobleme.

- Wiederverwendung von Teilproblemen: Durch das Speichern und Wiederverwenden von Lösungen für Teilprobleme werden redundante Berechnungen vermieden.

Nachteile:

- Hoher Speicherbedarf: Erfordert eine große Menge an Speicherplatz, da Lösungen für alle Teilprobleme gespeichert werden müssen (Speicherkomplexität von $O(n \cdot W)$, wobei n die Anzahl der Gegenstände und W die Kapazität des Rucksacks ist).
- Nicht für große Eingaben geeignet: Bei sehr großen Problemen (z.B. sehr viele Gegenstände oder sehr große Kapazitäten) kann der Speicher- und Zeitbedarf trotz pseudopolynomialer Laufzeit sehr hoch sein.

4. Greedy-Algorithmen

Im Folgenden wird der Greedy-Algorithmus betrachtet.

4.1 Prinzip der Greedy-Algorithmen

Die Idee des Greedy-Algorithmus für das Rucksackproblem besteht darin, Elemente fortlaufend nach ihrer Gewichtsdichte $c_j = \frac{a_j}{w_j}$ auszuwählen, solange die Kapazität des Rucksacks dies zulässt. Der Algorithmus beginnt mit einer Anfangslösung $x = (0, \ldots, 0)$ und ersetzt schrittweise Nullen durch Einsen, und zwar in der Reihenfolge abnehmender Gewichtsdichte c_j (d.h. von den am besten geeigneten Elementen zu den am wenigsten geeigneten Elementen). Bei jedem Schritt wird überprüft, ob die neue Lösung noch machbar ist, d.h., ob die Gesamtgewichte die Kapazität des Rucksacks nicht überschreiten.
Der Prozess endet, wenn keine weiteren Elemente mehr hinzugefügt werden können, ohne die Kapazität zu überschreiten. Die so gefundene Lösung x_G wird als gierige Lösung bezeichnet. [5]

4.2 Beispiele für Greedy-Lösungsansätze

Algorithm 2 Greedy-Algorithmus für das Rucksackproblem

Input: Liste von Gegenständen mit Gewicht w_i und Wert v_i, maximales Gewicht W des Rucksacks
Output: Liste von ausgewählten Gegenständen für den Rucksack
Sortiere die Gegenstände absteigend nach dem Verhältnis v_i/w_i
Initialisiere eine leere Liste S für die ausgewählten Gegenstände
Setze das aktuelle Gesamtgewicht $totalWeight$ des Rucksacks auf 0
for each Gegenstand i in der sortierten Liste **do**
    if $totalWeight + w_i \leq W$ **then**
        Füge den Gegenstand i zu S hinzu
        Erhöhe $totalWeight$ um w_i
    end if
end for
return S

4.3 Vor- und Nachteile

Vorteile:

- Effizienz: Hat eine kürzere Laufzeit (meist $O(nlogn)$ wegen des Sortierens),

und ist daher bei großen Datensätzen schneller als dynamische Programmierung.

- Schnelle Entscheidungsfindung: Trifft schnelle Entscheidungen ohne Tracing, was die Ausführungszeit weiter verkürzt.

Nachteile:

- Keine optimale Lösung des Rucksackproblems: Führt nicht immer zu einer optimalen Lösung des klassischen Rucksackproblems.
- Eingeschränkte Flexibilität: Weniger flexibel bei der Anpassung an komplexere Variationen des Rucksackproblems im Vergleich zur dynamischen Programmierung.

5. First fit decreasing

Es wird nun das Prinzip ”First fit decreasing” betrachtet.

5.1 Prinzip des First fit decreasing

Der heuristische Ansatz First Fit Decreasing (FFD), sortiert die Gegenstände absteigend nach ihrem Gewicht und legt jeden Gegenstand in den ersten Rucksack, in den er passt. Wird kein passender Rucksack gefunden, wird ein neuer Rucksack erstellt, und der Gegenstand wird in diesen eingefügt. Der FFD-Algorithmus ist einfach zu implementieren und liefert in der Regel gute, wenn auch nicht optimale, Lösungen für das Rucksackproblem. [6]

5.2 Beispiel für First fit decreasing

Algorithm 3 Heuristischer Ansatz ”First-Fit-Decreasing” für das Rucksackproblem

Input: Liste von Gegenständen mit Gewicht w_i und Wert v_i, maximales Gewicht W des Rucksacks
Output: Liste von ausgewählten Gegenständen für den Rucksack
Sortiere die Gegenstände absteigend nach dem Gewicht w_i
Initialisiere eine Liste von Rucksäcken B mit einem leeren Rucksack
for each Gegenstand i in der sortierten Liste **do**
    $j \leftarrow 1$
    while $j \leq |B|$ **do**
        if Gegenstand i passt in Rucksack $B[j]$ **then**
            Packe Gegenstand i in Rucksack $B[j]$
            break
        end if
        $j \leftarrow j + 1$
    end while
    if kein passender Rucksack gefunden **then**
        Erstelle einen neuen Rucksack und packe Gegenstand i hinein
        Füge den neuen Rucksack zu B hinzu
    end if
end for
return Alle Gegenstände in den Rucksäcken in B

5.3 Vor- und Nachteile

Vorteile:

- Schnelle Implementierung: Kann schnell implementiert werden, was es für einfache und schnelle Lösungsansätze attraktiv macht.

- Geringer Speicherbedarf: Benötigt neben den Eingabedaten nur wenig zusätzlichen Speicherplatz.

Nachteile:

- Keine Garantie für optimale Lösungen: Liefert nicht immer die optimale Lösung, insbesondere bei komplexeren oder ungünstigeren Eingaben.
- Kann ineffizient sein: In einigen Fällen kann es zu einer schlechten Ausnutzung der Kapazität kommen, da Gegenstände früh platziert werden, ohne den zukünftigen Platzbedarf zu berücksichtigen.

6. Rucksackproblem in der Praxis

Im Folgenden werden die Anwendungsbereiche des Rucksackproblems sowie ein Beispiel für das Rucksackproblem dargestellt.

6.1 Anwendungsgebiete

Das Rucksackproblem findet in vielen Bereichen der realen Welt Anwendung. Hier einige Beispiele:

- **Ressourcenoptimierung:**
  - **Zuschnitt von Rohmaterialien:** Das Rucksackproblem kann verwendet werden, um den Verschnitt beim Zuschnitt von Rohmaterialien zu minimieren. [2]
  - **Auswahl von Investitionen:** Das Rucksackproblem kann verwendet werden, um die optimale Auswahl von Investitionen für ein Portfolio zu finden. [2]
  - **Auswahl von Vermögenswerten:** Das Rucksackproblem kann verwendet werden, um die optimale Auswahl von Vermögenswerten für eine Asset-Backed Securitization zu finden. [2]
- **Kryptographie:**
  - **Generierung von Schlüsseln:** Das Rucksackproblem kann verwendet werden, um Schlüssel für kryptographische Systeme wie das Merkle-Hellman-Kryptosystem zu generieren. [2]

6.2 Beispiel

Es folgt ein Beispiel für den Greedy-Algortihmus.

6.2.1 Problemstellung

Ein Kobold steht vor einem Dilemma: Eine Schatzkammer voller wertvoller Gegenstände, aber in seinem Rucksack ist nur begrenzt Platz. Wie kann er seinen Gewinn maximieren, ohne zu viel Gewicht zu tragen?
Rucksackkapazität des Kobolds: 5kg
Verfügbare Gegenstände 6.1:

Name	Preis (Gold)	Gewicht (kg)	Preis/Gewicht (Gold/kg)
Glücksbringer	125	0,25	500
Opalglitzerschmuck	150	0,5	300
Amethystkette	60	0,25	240
Glühender Rubin	200	1	200
Mystischer Aquamarin	300	2	150
Säckchen Goldmünzen	50	0,5	100
Kugel des Untergangs	70	1	70
Antiker Elfenring	50	1	50
Handvoll Kupferstücke	10	0,5	20
Silberbarren	20	2	10

Tabelle 6.1: Preis und Gewichtsverhältnisse der Gegenstände

6.2.2 Lösung des Beispiels

Typisch für seine Art verwendet der Kobold für dieses Problem den Greedy-Algorithmus. Er nimmt also die Gegenstände mit dem höchsten Wert pro Gewicht, solange sie noch in seinen Rucksack passen. Vom Glücksbringer bis einschließlich zum Säckchen Goldmünzen passen alle Gegenstände in den Rucksack. Danach kann der Kobold weder die Kugel des Untergangs noch den antiken Elfenring mitnehmen und nimmt stattdessen noch die Handvoll Kupferstücke mit. Damit erreicht der Kobold 895 Gold mit 5kg Gewicht.

6.2.3 Diskussion der Ergebnisse

Der Greedy-Ansatz weicht hier nur unwesentlich von der besten Lösung ab, bei der der Kobold statt der Handvoll Kupferstücke und dem Säckchen Goldmünzen die Kugel des Untergangs mitnehmen sollte, für insgesamt 905 Gold. Der Greedy-Alogithmus liefert, wie man sieht, nicht immer die beste Lösung, aber eine gute Annäherung in sehr kurzer Zeit.

7. Zusammenfassung und Ausblick

Das Rucksackproblem ist ein grundlegendes Optimierungsproblem der Informatik und Mathematik mit einem breiten Anwendungsspektrum. Trotz seiner scheinbaren Einfachheit ist das Problem aufgrund seiner exponentiellen Komplexität eine Herausforderung. Es gibt verschiedene Lösungsansätze, z.B. die dynamische Programmierung, der Greedy-Algorithmus oder das First-Fit-Descreasing, das zwar eine Lösung in pseudopolynomialer Zeit liefert, aber nicht immer korrekt ist. Die vorgestellten Ansätze bieten verschiedene Herangehensweisen an das Rucksackproblem, aber es gibt noch Raum für weitere Untersuchungen und Verbesserungen. Die Integration von Techniken des Machine-Learning könnte neue Möglichkeiten zur Lösung des Rucksackproblems eröffnen und gleichzeitig den steigenden Anforderungen in verschiedenen Anwendungsbereichen gerecht werden.

Literaturverzeichnis

[1] VectorVoyager, “Knapsack problem illustration,” 06 2023, einsichtnahme: 12.07.2024. [Online]. Available: https://commons.wikimedia.org/wiki/File:Knapsack_Problem_Illustration.svg

[2] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer, 2004. [Online]. Available: https://books.google.de/books?id=u5DB7gck08YC

[3] M. Assi and R. A. Haraty, “A survey of the knapsack problem,” in *2018 International Arab Conference on Information Technology (ACIT)*, 2018, pp. 1–6.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.

[5] G. Diubin and A. Korbut, “The average behaviour of greedy algorithms for the knapsack problem: general distributions,” *Mathematical Methods of Operations Research*, vol. 57, pp. 449–479, 2003.

[6] S. Martello and P. Toth, “Algorithms for knapsack problems,” *North-Holland Mathematics Studies*, vol. 132, pp. 213–257, 1987.