Technische Universität München
Zentrum Mathematik

Diskrete Optimierung: Fallstudien aus der Praxis

Barbara Wilhelm | Michael Ritter

Eine Approximation für Knapsack-Probleme

Im Folgenden finden Sie eine formale Beschreibung des gewählten Approximationsalgorithmus. Arbeiten Sie die Beschreibung durch, machen Sie sich Notizen. Versuchen Sie, die Schritte des Algorithmus und die Beweise mit Hilfe von kleinen Skizzen nachzuvollziehen. Überlegen Sie sich, wie Sie die Erklärung dieses Algorithmus gestalten (Struktur, wichtige Ideen, Resultate). Wenn Sie Fragen haben, wenden Sie sich an Ihre Betreuer.

Problem: Knapsack
Input: Eine Zahl $n \in \mathbb{N}$, Gewichte $w \in \mathbb{N}^n$ und Werte $v \in \mathbb{N}^n$ sowie eine Kapazität $K \in \mathbb{N}$.

Aufgabe: Finde eine Teilmenge $I \subseteq \{1, \dots, n\}$ mit $\sum_{i \in I} w_i \leq K$, die größtmöglichen Gesamtwert besitzt.

Der vorgestellte Ansatz basiert auf dem pseudopolynomiellen Algorithmus aus der dynamischen Optimierung, der (etwas verkürzt dargestellt) so aussieht (die M_j sind dabei die „Schichten“ des Algorithmus, die jeweils aus Paaren bestehen, die den aktuellen Inhalt des Rucksacks und seinen Wert angeben):

1. Starte mit $M_0 \leftarrow \{\emptyset, 0\}$ (leerer Rucksack, Wert 0).
2. Für $i \in \{1, \dots, n\}$:
   a) Setze $M_i \leftarrow \emptyset$.
   b) Für jedes Element $(S, v) \in M_{i-1}$ füge (S, v) zu M_i hinzu. Ist $\sum_{j \in S} w_j + w_i \leq K$, so füge auch $(S \cup \{i\}, v + v_i)$ zu M_i hinzu.
   c) Bestimme für jedes $(S, v) \in M_i$ die Menge $M_i^{(v)} := \{(S', v') \in M_i : v' = v\}$ der Elemente, die in der zweiten Komponente übereinstimmen. Lösche jeweils alle Elemente in $M_i^{(v)}$ aus M_i mit Ausnahme desjenigen, das kleinstes Gesamtgewicht besitzt (wenn man den gleichen Wert auf verschiedene Weise erreichen kann, nimmt man immer die Kombination, die noch am meisten Spielraum lässt, und kann alle anderen „vergessen“).
3. Optimallösung ist ein Element $(S^*, v^*) \in M_n$ mit $v^* \geq v$ für alle $(S, v) \in M_n$.

In jedem Knoten speichern wir also den aktuellen Rucksack-Inhalt und den Wert; die Restkapazitäten berechnen wir jeweils bei Bedarf. Sei $v_{\max} := \max\{v_i : i \in \{1, \dots, n\}\}$. Wie Sie vielleicht schon wissen, hat obiger Algorithmus eine Laufzeit von $\mathcal{O}(n^3 v_{\max})$ (n Schichten mit maximal $n \cdot v_{\max}$ Knoten, für die jeweils noch eine Summe mit $\leq n$ Summanden berechnet werden muss). Damit ist der Algorithmus nicht polynomiell in der Größe des Inputs, da die Zahl v_i in $\mathcal{O}(\log_2(v_i))$ Bits codiert werden kann. (Anmerkung: Das kann man auch etwas anders implementieren und damit auch eine bessere Laufzeit erreichen. Der entscheidende Faktor in der Laufzeit ist für die Theorie aber das $v_{\max}$, weil es die Polynomialität zerstört, und das wird man auch mit einer anderen Implementierung nicht los. In der Praxis ist der Exponent beim n

natürlich auch wichtig. Für den Approximationsalgorithmus, den wir gleich betrachten, spielt das aber keine entscheidende Rolle.)

Wir konstruieren daraus nun einen Approximationsalgorithmus, indem wir von den Zahlen v_i einfach eine Anzahl Dezimalstellen abschneiden (also auf den nächsten Zehner, Hunderter, etc. abrunden). Damit liefert der Algorithmus natürlich keine genaue Lösung mehr, es zeigt sich aber, dass die Laufzeit besser wird. Durch Wahl einer geeigneten Anzahl abzuschneidender Dezimalstellen kann man dann zwischen Genauigkeit und Schnelligkeit abwägen.

1. Sei $\varepsilon > 0$, $v_{\max} := \max\{v_i : i \in \{1, \ldots, n\}\}$ und $t := \left\lfloor \log_{10}\left(\frac{\varepsilon v_{\max}}{n}\right)\right\rfloor$.
2. Für $i \in \{1, \ldots, n\}$ setze $v'_i := \left\lfloor \frac{v_i}{10^t} \right\rfloor \cdot 10^t$.
3. Wende den dynamische-Optimierung-Algorithmus oben auf die Knapsack-Instanz mit Gewichten $w_1, \ldots, w_n$, Kapazität K und Werten $v'_1, \ldots, v'_n$ an und liefere das entsprechende optimale S als Lösung zurück.

Satz 1
Für jedes $\varepsilon > 0$ gilt: Obiger Algorithmus liefert eine $1-\varepsilon$-Approximation für das ursprüngliche Knapsack-Problem. Die Laufzeit beträgt $\mathcal{O}(n^4/\varepsilon)$.

Beweis. Die Laufzeitschranke ergibt sich direkt aus der Laufzeit des exakten Algorithmus von $\mathcal{O}(n^3 v_{\max})$ und dem Zusammenhang von t und ε. Für die Approximation sei $S \subseteq \{1, \ldots, n\}$ eine exakte Lösung des Original-Knapsack-Problems und $S' \subseteq \{1, \ldots, n\}$ die Lösung des Approximationsalgorithmus. Da die Gewichte unverändert bleiben, ist S' natürlich eine zulässige Lösung für das Originalproblem. Für die Zielfunktionswerte gilt:

$$\sum_{i \in S} v_i \geq \sum_{i \in S'} v_i \geq \sum_{i \in S'} v'_i \geq \sum_{i \in S} v'_i \geq \sum_{i \in S} (v_i - 10^t) \geq \sum_{i \in S} v_i - n \cdot 10^t$$

Damit folgt:

$$\frac{\sum_{i \in S} v_i - \sum_{i \in S'} v_i}{\sum_{i \in S} v_i} \leq \frac{n \cdot 10^t}{v_{\max}} \leq \varepsilon$$

Das beweist die Behauptung. □

Wenn noch Zeit bleibt, arbeiten Sie den folgenden Approximationsalgorithmus bitte noch durch.

Wir beschäftigen uns nun noch mit einer völlig anderen Idee, die zu einem einfacheren Algorithmus führt, dafür aber auch nur eine 2-Approximation des Knapsack-Problems liefert. Dafür nehmen wir an, dass $w_i \leq K$ für alle $i \in \{1, \ldots, n\}$ gilt (Gegenstände, die diese Ungleichung verletzen, kann man einfach entfernen) und dass $\sum_{i=1}^{n} w_i > K$ (sonst ist das Problem trivial).

1. Sortiere die Gegenstände $1, \ldots, n$ nach absteigendem *relativen Wert* $\nu_i := v_i/w_i$. Wir nehmen im Folgenden an, dass die Nummerierung der Gegenstände zu dieser Sortierung passt, d. h. dass
$$\nu_1 \geq \nu_2 \geq \cdots \geq \nu_n \quad \text{gilt.}$$
2. Setze $I \leftarrow \emptyset$, $R \leftarrow K$, $i \leftarrow 1$.

3. Solange $R \geq w_i$:
   a) $I \leftarrow I \cup \{i\}$
   b) $R \leftarrow R - w_i$
   c) $i \leftarrow i + 1$
4. Falls $v(I) \geq v_i$, liefere als Lösung I zurück, sonst $\{i\}$.

Der Algorithmus füllt den Rucksack also einfach der Reihe nach auf, bis schließlich ein Gegenstand nicht mehr hineinpasst. Dann vergleicht er, ob es günstiger ist, den bisher gefüllten Rucksack zu nehmen oder einfach den ersten Gegenstand, der nicht mehr genommen wurde, einzupacken, und liefert die entsprechende Lösung zurück.

Satz 2
Der Algorithmus ist eine $\frac{1}{2}$-Approximation für das Knapsack-Problem.

Beweis. Schreibt man das Knapsack-Problem als ILP, so erhält man

$$\begin{aligned} \max\ & v^T x \\ & w^T x \leq K \\ & x \in \{0,1\}^n . \end{aligned}$$

Die LP-Relaxation und ihr Duales lauten somit:

$$\begin{aligned} \max\ & v^T x \\ & w^T x \leq K \\ & x \leq \mathbb{1} \\ & x \geq 0 \end{aligned} \qquad\qquad \begin{aligned} \min\ & Ky + \mathbb{1}^T z \\ & w_i y + z_i \geq v_i \quad \text{für } i \in \{1,\dots,n\} \\ & y, z \geq 0 \\ & y \in \mathbb{R}, z \in \mathbb{R}^n \end{aligned}$$

Sei I die im Algorithmus berechnete Menge (nicht unbedingt die zurückgelieferte Lösung), sei $R := K - \sum_{i \in I} w_i$ und $i^* := (\max I) + 1$. Setze

$$x_i^* := \begin{cases} 1, & \text{falls } i \in I, \\ \frac{R}{w_{i^*}}, & \text{falls } i = i^*, \\ 0, & \text{sonst;} \end{cases}$$
$$y^* := \frac{v_{i^*}}{w_{i^*}};$$

sowie

$$z_i^* := \begin{cases} v_i - w_i y, & \text{für } i \in I, \\ 0, & \text{sonst.} \end{cases}$$

Dann ist x^* Optimallösung der LP-Relaxation und y^*, z^* Optimallösung des Dualen: Offenbar sind die Vektoren zulässig (nachrechnen!) und es gilt

$$v^T x^* - (Ky^* + \mathbb{1}^T z^*) = \sum_{i \in I} v_i + R\frac{v_{i^*}}{w_{i^*}} - Ky^* - \sum_{i \in I}(v_i - w_i y^*) = y^* \left(R - K + \sum_{i \in I} w_i \right) = 0.$$

Die vom Algorithmus berechnete Lösung ist entweder $I = \{1, \ldots, i^* - 1\}$ oder $\{i^*\}$, je nachdem, welche von diesen beiden Lösungen den größeren Zielfunktionswert besitzt. Da aber $w_{i^*} > R$ ist, gilt

$$v^T x^* = \sum_{i=1}^{i^*-1} v_i + \frac{R}{w_{i^*}} v_{i^*} \leq v(I) + v_{i^*}.$$

Daher muss $\max\{v(I), v_{i^*}\} \geq 1/2 v^T x^*$ sein, das beweist die Behauptung. □