Approximationsalgorithmen

In polynomieller Zeit lässen sich nicht exakte Lösungen von NP-harten Problemen berechnen.

Approximationsalgorithmen versuchen das beste aus einer Polynomialzeitberechnung zu machen, indem sie wenigstens eine Näherunslösung bestimmen.

Im Gegensatz zu Heuristiken, verlangen wir aber von Approximationsalgorithmen eine Garantie, daß die berechnete Näherungslösung nah am Optimum ist.

Approximationsalgorithmen

Sei A ein Algorithmus, der in polynomieller Zeit eine Näherungslösung zu einem gegebenen Optimierungsproblem berechnet.

Sei $F^*(I)$ der optimale Wert der Zielfunktion und $F(I)$ der vom Algorithmus A erzielte Wert.

Definition

Das *Approximationsverhältnis* von A ist α, wenn

$$\frac{F(I) - F^*(I)}{F^*(I)} \leq \alpha \text{ und } \frac{F^*(I) - F(I)}{F^*(I)} \leq \alpha$$

gilt.

Approximationsalgorithmen

Diese Definition ist sowohl für Minimierungs- als auch für Maximierungsprobleme sinnvoll.

Für Minimierungsprobleme ist diese Ungleichung ausschlaggebend:

$$\frac{F(I) - F^*(I)}{F^*(I)} \leq \alpha$$

Für Maximierungsprobleme dagegen die andere:

$$\frac{F^*(I) - F(I)}{F^*(I)} \leq \alpha$$

Beispiel: Vertex Cover

Das Approximationsverhältnis ist 2.

$C := \emptyset$;
while $E \neq \emptyset$ **do**
  choose some $e \in E$;
  $V := V - e$;
  $C := C \cup e$;
  $E := \{ e' \in E \mid e \cap e' = \emptyset \}$
od;
return C

Kompetitive Analyse

Um das Approximationsverhältnis zu analysieren, bietet sich eine *kompetitive Analyse* der folgenden Form an:

Wir gehen von einer Instanz und einer zugehörigen optimalen Lösung aus.

Wir müssen dann beweisen, daß der Approximationsalgorithmus eine Lösung findet, die nur um α schlechter als die vorgegebene optimale Lösung ist.

Kompetitive Analyse

Im Falle von Vertex Cover ist eine kompetitive Analyse recht einfach:

Sei $G = (V, E)$ ein Graph und $C \subseteq V$ ein minimales Vertex Cover.

Der Algorithmus wählt eine Kante $e = \{v_1, v_2\} \in E$ aus fügt v_1 und v_2 zu seinem VC hinzu. Einer von beiden Knoten ist aber auch in C. Zu einem Knoten in C, fügt der Algorithmus also höchstens 2 zu seiner Lösung hinzu.

Das macht ein Verhältnis von höchstens 2 zwischen konstruierter und optimaler Lösung aus.

Approximationsalgorithmen

Definition

1. Ein Approximationsalgorithmus ist ein *$f(n)$-Approximationsalgorithmus*, wenn das Approximationsverhältnis höchstens $f(n)$ für alle Instanzen der Länge n ist.
2. Ein *ϵ-Approximationsalgorithmus* ist ein $f(n)$-Approximationsalgorithmus mit $f(n) \leq \epsilon$ für alle $n \in \mathbf{N}$.

Approximationsschemata

Definition

Sei $A(\epsilon)$ ein Approximationsalgorithmus mit einem Parameter ϵ.

1. $A(\epsilon)$ ist ein *PTAS (polynomial time approximation scheme)*, falls $A(\epsilon)$ ein ϵ-Approxmiationsalgorithmus für jedes $\epsilon > 0$ ist und polynomielle Laufzeit in der Eingabelänge für jedes $\epsilon > 0$ hat.

2. $A(\epsilon)$ ist ein *FPTAS (fully polynomial time approximation scheme)*, falls $A(\epsilon)$ ein ϵ-Approxmiationsalgorithmus für jedes $\epsilon > 0$ ist und polynomielle Laufzeit in der Eingabelänge und $1/\epsilon$ für jedes $\epsilon > 0$ hat.

Scheduling

Wir betrachten folgendes Problem:

Gegeben sind n Aufgaben und m identische Prozessoren, sowie die Ausführungszeiten $t_1, \ldots, t_n$ der n Aufgaben.

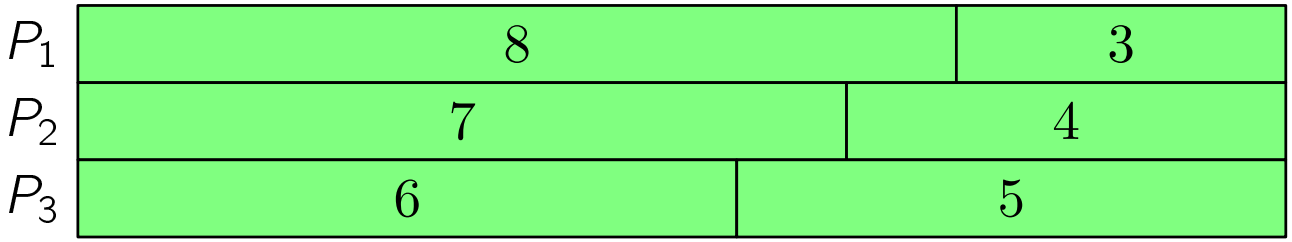
Gesucht ist eine Verteilung der Aufgaben auf die m Prozessoren, so daß die *Schlußzeit* minimiert wird, also die Zeit, bis alle Aufgaben erledigt wurden.

- Wir nennen dieses Problem im folgenden *Scheduling*.
- Für $m = 1$ ist es sehr leicht zu lösen.
- Ab $m = 2$ ist es NP-vollständig.

Beispiel

Sei $m = 3$, $n = 7$ und $(t_1, \ldots, t_6) = (8, 7, 6, 5, 4, 3)$.

Dieser Schedule ist optimal:



Die Optimalität folgt hier aus der Tatsache, daß alle Prozessoren gleichzeitig fertig werden.

Die LPT-Regel

Ein sehr einfaches Verfahren, in polynomieller Zeit einen Schedule zu erzeugen, nennt sich *LPT = longest processing time*.

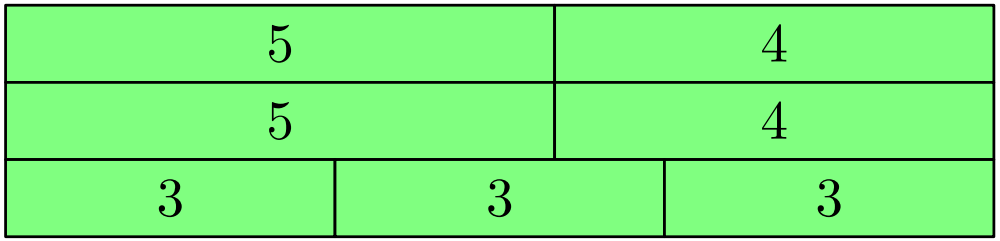
Definition

Die LPT-Regel weist dem nächsten freiwerdenden Prozessor die Aufgabe mit der größten Bearbeitungszeit zu. (Gibt es mehrere gleich große, dann nehmen wir irgendeine von ihnen.)

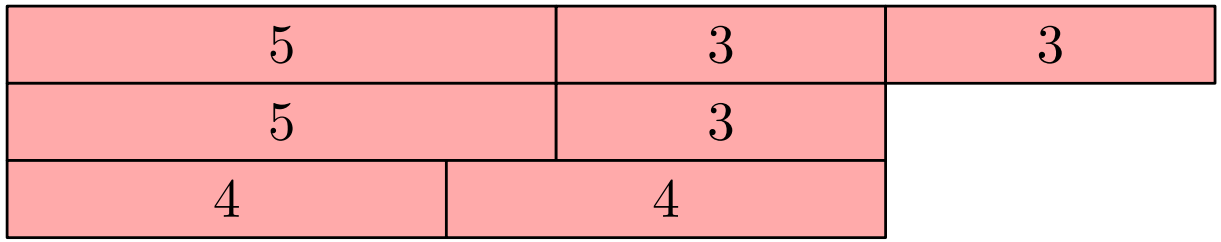
Beispiel

Sei $m = 3$, $n = 7$, $(t_1, \ldots, t_7) = (5, 5, 4, 4, 3, 3, 3)$.

Ein optimaler Plan ist offensichtlich:



Die LPT-Regel liefert dagegen:



Natürlich kann die LPT-Regel nicht immer optimale Pläne konstruieren.

Wie gut ist aber ein LPT-Schedule?

LPT-Schedules

Theorem

Sei I eine Instanz des m-Prozessorscheduling und $F^*(I)$ die optimale Schlußzeit. Sei $F(I)$ die Schlußzeit eines LPT-Schedules.

Dann gilt:

$$\frac{F(I) - F^*(I)}{F^*(I)} \leq \frac{1}{3} - \frac{1}{3m}$$

Insbesondere liefert die LPT-Regel einen 1/3-Approximationsalgorithmus.

Beweis

Für $m = 1$ ist ein LPT-Schedule optimal, also $F(I) = F^*(I)$. Sei nun $m > 1$.

Nehmen wir an, daß Theorem ist falsch. Dann gibt es eine Instanz $(t_1, \ldots, t_n)$, für die der LPT-Schedule schlechter ist, als vom Theorem behauptet.

Wir können annehmen, daß das Theorem für alle Instanzen mit weniger als n Aufgaben gilt und daß $t_1, \ldots, t_n$. Die LPT-Regel soll die Aufgaben in der Reihenfolge ihrer Indizes zuordnen.

Sei $F(I)$ die Schlußzeit des LPT-Schedules.

Behauptung 1: Die Aufgabe mit Index n wird genau zur Schlußzeit, aber nicht früher beendet.

Wenn nicht, dann gibt es eine Aufgabe $k < n$, die später als Aufgabe n zur Schlußzeit beendet wird. Sei I' die Instanz, die aus den Aufgaben $1, \ldots, k$ besteht.

$$\frac{F(I') - F^*(I')}{F^*(I')} \geq \frac{F(I) - F^*(I)}{F^*(I)} > \frac{1}{3} - \frac{1}{3m}$$

Dies ist ein Widerspruch dazu, daß das Theorem für weniger als n Aufgaben gilt.

Behauptung 2: In einem optimalen Plan für I werden nicht mehr als zwei Aufgaben von einem Prozessor erledigt.

Wegen Behauptung 1 wird die Aufgabe n zum Zeitpunkt $F(I) - t_n$ im LPT-Schedule gestartet. Bis zu diesem Zeitpunkt müssen alle Prozessoren beschäftigt sein, denn sonst wäre Aufgabe n früher zugeordnet worden. Daraus folgt:

$$F(I) - t_n \leq \frac{1}{m} \sum_{n=1}^{n-1} t_i$$

$$F(I) \leq \frac{1}{m} \sum_{n=1}^{n} t_i + \frac{m-1}{m} t_n$$

Wir haben die untere Schranke $F^*(I) \geq \frac{1}{m} \sum_{i=1}^{n} t_i$.

Behauptung 2: In einem optimalen Plan für I werden nicht mehr als zwei Aufgaben von einem Prozessor erledigt.

Daraus folgt $F(I) - F^*(I) \leq \frac{m-1}{m} t_n$ oder

$$\frac{F(I) - F^*(I)}{F^*(I)} \leq \frac{m-1}{m} \frac{t_n}{F^*(I)}.$$

Da das Theorem für I nicht gilt, erhalten wir

$$\frac{1}{3} - \frac{1}{3m} < \frac{m-1}{m} \frac{t_n}{F^*(I)},$$

woraus $F^*(I) < 3t_n$ folgt.

Damit ist Behauptung 2 bewiesen.

Wir sind im Beweis des Theorems nun so weit:

Falls das Theorem für eine Instanz I versagt, dann werden in einem optimalen Plan für I jedem Prozessor höchstens zwei Aufgaben zugeordnet.

Dies passiert aber nie, denn es gilt:

Behauptung 3: Falls ein optimaler Plan für I jedem Prozessor höchstens zwei Aufgaben zuordnet, dann ist ein LPT-Schedule für I ebenfalls optimal.

Beweis von Behauptung 3: Übungsaufgabe!

□

Ein PTAS für Scheduling

Folgendes Verfahren führt zu einem PTAS:

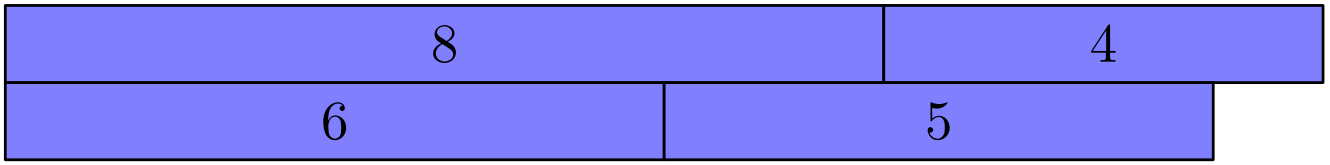
1. Wähle ein festes k (unabhängig von der Eingabe).
2. Berechne einen optimalen Plan für die k Aufgaben, mit der längsten Bearbeitungszeit.
3. Verplane die verbleibenden $n - k$ Aufgaben mit der LPT-Regel.

Warum sollte dieses Verfahren intuitiv gut sein?

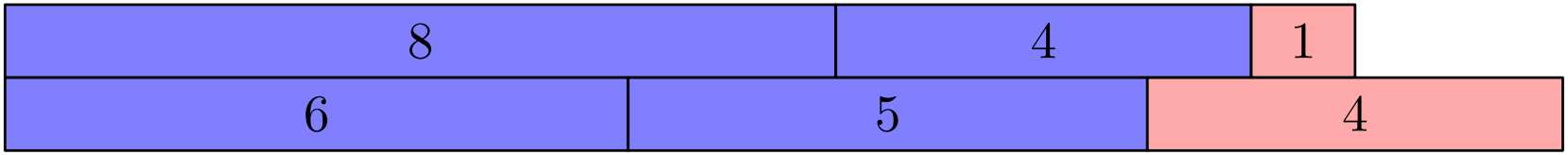
Beispiel

Sei $m = 2$, $n = 6$, $(t_1, \ldots, t_6) = (8, 6, 5, 4, 4, 1)$. Wir wählten $k = 4$.

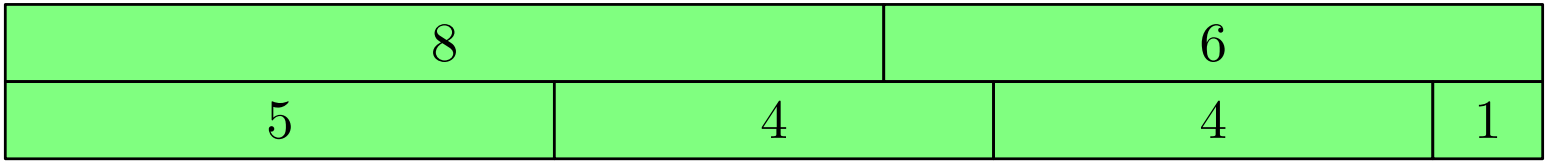
Schritt 1: Optimaler Plan für die 4 längsten Aufgaben.



Schritt 1: LPT-Plan für die verbleibenden 2 Aufgaben.



Die optimale Lösung ist:



Ein PTAS für Scheduling

Theorem

Sei $F(I)$ die Schlußzeit eines Plans, der durch das angebene Verfahrens erzeugt wird und $F^*(I)$ die optimale Schlußzeit von I.

Dann gilt

$$\frac{F(I) - F^*(I)}{F^*(I)} \leq \frac{1 - 1/m}{1 + \lfloor k/m \rfloor}.$$