Aufgaben zum Tut am 06.02.2007

Thomas Pajor

6. Februar 2007

Aufgabe 1.

Das Problem MULTIPROCESSOR SCHEDULING (MPS) ist wie folgt definiert:

Gegeben: Eine Menge $M := \{1, \ldots, m\}$ von m identischen Maschinen oder Prozessoren und eine Menge J von n Jobs mit positiven Bearbeitungsdauern, also einer Abbildung $p : J \to \mathbb{N}$ die jedem Job eine Zeit zuordnet. Außerdem eine Deadline $D \in \mathbb{N}$.

Frage: Existiert ein Schedule s mit Abarbeitungslänge höchstens D? Also existiert eine Zuordnung der Jobs auf die Maschinen $s : J \to M$, so dass

$$\max_{1 \leq j \leq m} \sum_{i \in s^{-1}(j)} p(i) \leq D$$

Zeigen Sie: MPS ist $\mathcal{NP}$-vollständig.

Lösung.

Wir zeigen MPS $\in \mathcal{NPV}$ in zwei Schritten.

1. MPS $\in \mathcal{NP}$:

   Zu einem geratenen Lösungsvorschlag s können wir mit Aufwand $\mathcal{O}(|M| \cdot |J|)$ überprüfen ob er eine gültige Lösung ist in dem wir für jede Maschine die Laufzeiten der zugeordneten Jobs aufaddieren und prüfen ob die Summe höchstens D ist.

   $\Rightarrow$ MPS ist in $\mathcal{NP}$.

2. MPS ist $\mathcal{NP}$-hart:

   Wir reduzieren das Problem PARTITION auf MPS, also PARTITION $\leq_p$ MPS.

   Gegeben sei eine Instanz $I = (A, w)$ des Problems PARTITION. Wir konstruieren eine Instanz $I' = (M, J, p, D)$ von MPS wie folgt:

(1) $|M| := 2$
(2) $|J| := |A|$
(3) $p(i) := w(i)$
(4) $D := \frac{1}{2} \cdot \sum_{i \in A} w(i)$

Der Aufwand für (2), (3) und (4) ist jeweils in $\mathcal{O}(|A|)$. (1) hat einen Aufwand von $\mathcal{O}(1)$. Der Gesamtaufwand der Transformation ist also polynomiell.

Es bleibt noch zu zeigen:

$$I \text{ ist Lösung von PARTITION} \quad \Leftrightarrow \quad I' \text{ ist Lösung von MPS}$$

Sei I eine Lösung von PARTITION und A_1, A_2 die Teilmengen von A mit

$$\sum_{i \in A_1} w(i) = \sum_{i \in A_2} w(i)$$

dann gilt wegen der Konstruktion gerade

$$\sum_{i \in A_1} w(i) = \sum_{i \in A_2} w(i) = D$$

Also ist auch I' mit folgendem Schedule s

$$s(i) := \begin{cases} 1 & \text{falls} \quad i \in A_1 \\ 2 & \text{falls} \quad i \in A_2 \end{cases} \qquad \forall i \in J$$

eine Lösung von MPS.

Sei nun I' eine Lösung von MPS und seien J_1 und J_2 die Mengen der Jobs die durch s auf die erste bzw. zweite Maschine abgebildet wurden. Also gilt:

$$\sum_{i \in J_1} p(i) \leq D \tag{1}$$

$$\sum_{i \in J_2} p(i) \leq D \tag{2}$$

Zusammengefasst gilt nach Konstruktion von D:

$$\sum_{i \in (J_1 \cup J_2)} p(i) \leq 2D = \sum_{i \in A} w(i)$$

Wegen $p(i) = w(i)$ gilt sogar

$$\sum_{i \in (J_1 \cup J_2)} p(i) = 2D$$

Das heißt es muss gelten

$$\sum_{i \in J_1} p(i) = \sum_{i \in J_2} p(i) = D$$

da wir sonst einen Widerspruch zu den Gleichungen (1) und (2) kriegen.

Damit ist auch I eine Lösung von PARTITION.

$\Rightarrow$ MPS ist $\mathcal{NP}$-hart.

Gäbe es nun einen effizienten Algorithmus $\mathcal{A}$ der MPS lösen kann, so könnten wir $\mathcal{A}$ mit Hilfe der obigen Transformation dazu benutzen um für jede Instanz I von PARTITION zu überprüfen ob sie eine Lösung hat.

$\Rightarrow$ MPS ist in $\mathcal{NPV}$. □

Aufgabe 2.

Zeigen Sie: Das Halteproblem ist $\mathcal{NP}$–hart. Warum ist es nicht $\mathcal{NP}$–vollständig?

Lösung.

Wir reduzieren 3SAT auf das Halteproblem. ↑

Sei $\mathcal{M}$ eine Turingmaschine, die als Eingabe eine 3SAT Instanz $I := (C, P := \{P_1, \ldots, P_n\})$ erhält wobei C eine Menge von Klauseln über P ist. $\mathcal{M}$ arbeite wie folgt:

(1) Schreibe einen Bitvektor X der Länge $|P|$ auf das Band und initialisiere ihn mit lauter Nullen. $\mathcal{M}$ behandle den Bitvektor X wie folgt: An der i–ten Stelle von X steht genau dann eine 1, wenn P_i als wahr interpretiert wird.

(2) Prüfe ob X eine erfüllende Variablenbelgung für I ist. Falls ja: stoppe. Sonst überschreibe X durch seinen lexikographischen Nachfolger und prüfe erneut.

(3) Falls $X = (1, \ldots, 1)$ gehe in eine Endlossschleife (Beispielsweise indem der Kopf unendlich lange nach Rechts gefahren wird).

Betrachte nun eine 3SAT Instanz $I := (C, P)$. Wir konstruieren dazu eine Instanz $I' := (\langle \mathcal{T} \rangle, w)$ des Halteproblems wie folgt:

- $\langle \mathcal{T} \rangle := \langle \mathcal{M} \rangle$
- w ist die Menge der Klauseln sowie die aussagenlogischen Variablen in einer geeigneten Kodierung.

Die Transformation ist offensichtlich polynomiell.

Nach Konstruktion von $\mathcal{M}$ hält die Turingmaschine, wenn die als Eingabe übergebene 3SAT Instanz eine erfüllende Belegung hat, denn es werden in Schritt (2) alle möglichen Variablenbelegungen für P überprüft. Hat I keine erfüllende Belegung so geht $\mathcal{M}$ in Schritt (3) in eine Endlosschleife, und terminiert nicht. Wir konnten also zeigen dass das Halteproblem $\mathcal{NP}$–hart ist. □

Das Halteproblem ist jedoch nicht $\mathcal{NP}$–vollständig, da dafür zusätzlich verlangt wird, dass es in $\mathcal{NP}$ liegt. Dies ist aber nicht gegeben, da das Halteproblem nicht entscheidbar ist. Dies ist also ein Beispiel für ein $\mathcal{NP}$–hartes Problem, das aber *nicht* $\mathcal{NP}$–vollständig ist.

Aufgabe 3.

Gegeben sei folgender naiver Algorithmus zum überprüfen, ob eine Zahl $n \in \mathbb{N}$ eine Primzahl ist:

Algorithmus 1 : PRIMTEST

Eingabe : $n \in \mathbb{N}$
Ausgabe : Ob n prim oder nicht

```
1 für k ← 2 ... (n − 1) tue
2     wenn k teilt n dann
3         return n ist nicht prim
4 return n ist prim
```

Warum konnte dennoch erst 2004 gezeigt werden, dass das Problem PRIMES in $\mathcal{P}$ liegt[1]?

Lösung.

Die Laufzeit des Algorithmus ist tatsächlich durch ein Polynom in n beschränkt. Jedoch muss n kodiert werden, und die Laufzeit muss polynomiell in der Länge der Kodierung von n sein.

Wähle zum Beispiel eine Kodierung zur Basis $b \geq 2$ und betrachte ein Wort der Länge x. Dann lassen sich damit Zahlen zwischen 0 und $b^x - 1$ kodieren. Unser Algorithmus muss also für eine Eingabe der Länge x exponentiell viele Teiler überprüfen, und somit ist die Laufzeit des Algorithmus exponentiell in der Eingabelänge.

Anmerkung: „Polynomiell“ wäre der Algorithmus durchaus, wenn man als Basis für die Kodierung $b = 1$ wählt. Ein Algorithmus, dessen Laufzeit bei unärer Kodierung polynomiell wird, sonst aber exponentiell ist, heißt auch *pseudopolynomieller Algorithmus*.

[1] http://www.math.princeton.edu/%7Eannals/issues/2004/Sept2004/Agrawal.pdf