Max Wisniewski
Alexander Steen

Dozent : Wolfgang Mulzer

Exercise 1

Sei n die Anzahl der verschiedenen Sammelbilder (Twighlight Bilder). Sei X Zufallsvariable für die Anzahl der benötigten Packungen Müsli, bis wir alle n Sammelbilder haben. Die Wahrscheinlichkeit für ein bestimmtes Bild ist gleichverteilt. [1]

(a)

$E[X]$ soll die Anzahl der Runden sein, bis man alle Bilder hat. $E[X_i]$ soll beschreiben, wie viele Runden man in Runde i braucht, das heißt wie viele Runden wir von $i-1$ bis i Bilder brauchen. Nun brauchen wir alle Bilderkarten $n = \sum_{i=1}^{n} i$.Die Zufallsvariable X_i beschreibt die Anzahl der Runden, das heißt für alle Runden brauchen wir $X = \sum_{i=1}^{n} X_i$.
Nach der Linearität des Erwartungswertes gilt:

$$E[X] = E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i]$$

(b)

Als erstes zeigen wir, dass wir für $E[X_i]$ eine geometrische Verteilung ist, d.h. das gilt: $Pr(X_i = k) = (1-p_i)^{k-1} \cdot p_i$

Gehen wir unseren Wahrscheinlichkeitsbaum hinunter haben wir jedesmal die möglichkeit, dass wir eine Karte bekommen, die wir schon haben oder wir bekommen eine neue. Eine neue Karte erhalten wir mit p_i, dass heißt, wenn wir keine Karte bekommen, haben wir jedesmal den Zweig mit den $1-p_i$ genommen, der Gegenwahrscheinlichkeit. Erhalten wir nun in der $k$ten Runde eine neue Karte, so müssen wir in den $k-1$ Runden vorher keine erhalten haben. $\Rightarrow Pr(X_i = k) = (1-p_i)^{k-1} \cdot p_i$.
Wir haben für $E[X_i]$ eine geometrische Verteilung.
$E[X_i] = \frac{1}{p_i}$, mit p_i Wahrscheinlichkeit ein neues Bild (das i-te Bild) zu erhalten.
Für p_i ergibt sich $p_i = \frac{n-i+1}{n}$, da wir in Runde 1 eine Wahrscheinlichkeit von $p_1 = \frac{n-1+1}{n} = 1$, in Runde 2 $p_2 = \frac{n-2+1}{n} = \frac{n-1}{n}$, etc. haben, ein neues Bild zu erhalten.
Dann ist

$$E[X_i] = \frac{1}{p_i} = \frac{1}{\frac{n-i+1}{n}} = \frac{n}{n-i+1}$$

[1] This task was copied form our solution to exercisesheet 1 exercise 2 of *Höhere Algorithmic* in WiSe 2011.

(c)

z.z.: $E[X] = O(n \log n)$

$$E[X] \overset{(a)}{=} \sum_{i=1}^{n} E[X_i] \overset{(b)}{=} \sum_{i=1}^{n} \frac{n}{n-i+1} = n \sum_{i=1}^{n} \frac{1}{n-i+1}$$

$$\overset{(*)}{=} n \sum_{i=1}^{n} \frac{1}{i} = n \cdot O(log\, n) = O(n \log n)$$

Nun ist $(*)$ eine einfache Umsortierung der Elemente der Summe, (die wir vornehmen können, weil die Summe endlich ist) indem wir die Elemente von unten nach oben aufsummieren anstatt anders herum.

Exercise 2

In the knapsack problem, we are given n items. Each item has a *weight* g_i and a *value* w_i. Furthermore , we have a maximum weight G. All inputs are positive integers.

We would like to find a set $I \subseteq \{1, ..., n\}$ of items, such that the total value $\sum_{i \in I} w_i$ is maximum, subject to the constraint that the total weight is at most G, meaning $\sum_{i \in I} g_i \leq G$.

(a) Define an appropriate decision version for the knapsack problem and show that it is NP-complete.

Solution:

We choose the canonical extension of an optimisation problem to a decision problem. Let the defined variables be as the the optimisation problem. Than take an additional positive Integer K.

The decision question is now, does there exist a set $I \subseteq \{1, ..., n\}$, such that $\sum_{i \in I} w_i \geq K$ and $\sum_{i \in I} g_i \leq G$.

Claim 1. *The decision problem for the knapsack problem is NP-complete.*

proofsketch 1.

The knapsack problem is in NP. To show that, suppose I is an optimal solution to the problem. It is linear in the input size, therefore acceptable. We now may check in polynomial time, that the weight constraint is met, and that the summed value is bigger than K.

To reduce subset-sum to knapsack we choose the following transformation. Let S be the set of positive integers for subset-sum and T a value. The question for knapsack is now, does there exist a subset $O \subseteq S$ such taht $\sum_{i \in O} i = T$.

The reduction now sets $w_i = g_i = i$ for all $i \in S$ and $G = K = T$ for the goal values. One can easily see, that the reduction can be done in lineare time, thus it can be computed in polynomial time.

Knapsack run on this input will now hold the following. If there exists a set I that satisfies the constraints $\sum_{i \in I} w_i = \sum_{i \in I} i \geq K$ and $\sum_{i \in I} g_i = \sum_{i \in I} i \leq K$. Thus if we set $O = I$ $\sum_{i \in O} i = T$ holds.
If no solution existed in the original problem, than we cannot find a solution in the reduction, because both sums will sum up to exactly the same. So one of the constraints cannot be true.

(b)

Let $W := \sum_{i=1}^{n} w_i$. Show that the knapsack problem can be solved in $O(nW)$ time. Why does this not contradict *(a)*?

Solution:

We proof this by giving a dynamic program, we used in approximation algorithms. [2] We construct an Array A of sets of tupel. A tupel $(t, w) \in A(j)$ means. There exists a subset $I \subseteq \{1, .., j\}$, such that $\sum_{i \in I} w_i = w$ and $\sum_{i \in I} g_i = t$.

Furthermore we introduce a *domination* relation. A tupel (t, w) dominates another tupel (t', w'), iff $t \leq t'$ and $w \geq w'$. (Means we can get a bigger value with less weight). Observe, that we can have at most $G + 1$ and $W + 1$ tupels in $A(j)$ for any $j \leq n$, because otherwise there is a tupel $(t', w') \in A(j)$ that is dominated by another tupel. Next we assume, that we can put any item in the bag itself. If not, we can throw them away.

```
A(1) ← {(0,0),(g_1,w_1)}
for j ← 2 to n do
    A(j) ← A(j-1)
    for each (t,w) ∈ A(j-1) do
        if t + g_j ≤ G then
            A(j) ← A(j) ∪ {(t + g_j, w + w_j)}
    Remove dominated pairs
return max_{(t,w)∈A(n)} w
```

In each iteration we make a constant number of steps for each element of the previous set of tupels. We showed, that in any set $A(j)$ we can bound the number of tupels by $\min\{G+1, W+1\}$. We iterate in the outer loop for any element in our bag.

Therefore we can write, that the runtime of our algorithm lies in $O(nW)$. (If G is smaller than W our estimation is to big, but still holds).

This solution does not contradict with (a) because W is epxonantial in the inpu size (which is $log\, W$). The algorithm is pseudo-polynomial.

[2] Exercise 1 b) / c) are done with notes to Approximation algorithm (SoSe 2012) and the Book "Design of Approximation algorithms".

(c)

For the $(1-\varepsilon)$ - approximation we also use a trick learned in approximation algorithms, based on the algorithm of (b).

The trick is to round the values of w_i, so we do not have to look at each single value.

We assign new values by the following scheme

```
M ← max_{i∈I} w_i
μ ← ε M/n
w_i' ␣← ⌊w_i/μ⌋ for all i ∈ S
```

and run the algorithm from (b) on it. With the modified Values we now get

$$W' = \sum_{i=1}^{n} w_i' = \sum_{i=1}^{n} \left\lfloor \frac{w_i}{\varepsilon M/n} \right\rfloor = O(\frac{n^2}{\varepsilon}).$$

Because now W' is polynomial in n and ε it is strictly smaller than B which is exponantial in the input size. So the runtime of our algorithm is $O(n^3 \cdot \frac{1}{\varepsilon})$, which is in $poly(n, \frac{1}{\varepsilon})$.

Leaves us to proof, that the algorithm is a $(1-\varepsilon)$ - approximation. Let I be the set which gives us the optimal solution on W and O the set, that gives use the optimal solution on W' the rounded values.

First observe, that $M \leq OPT$, because we can always can take the most valueable item. The rounding of the values gives us the estimation $\mu w_i' \leq w_i \leq \mu(w_i' + 1)$ wich leads us to $\mu w_i' \geq w_i - \mu$.

$$\begin{aligned}
\sum_{i\in O} w_i &\geq \mu \sum_{i\in O} w_i' \\
&\geq \mu \sum_{i\in I} w_i' \\
&\geq (\sum_{i\in I} w_i) - |I|\mu \\
&\geq (\sum_{i\in I} w_i) - n\mu \\
&= (\sum_{i\in I} w_i) - \varepsilon M \\
&\geq OPT - \varepsilon OPT = (1-\varepsilon)OPT
\end{aligned}$$

We can conclude, that our algorithm is an FPTAS for the knapsack problem.

Exercise 3

Blabla. Subexp.