Lecture 12

Gidon Rosalki

2024-12-08

1 Dynamic algorithms

Example 1 (The complete knapsack problem)**.** *Input: W - the maximal weight that the knapsack can take, and n pairs of numbers $(v_1, w_1), \ldots, (v_n, w_n)$. v_i - the value of the ith item, w_i - the weight of the ith item. We may not split up the items.*
Output: $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$, and such that $\sum_{i \in S} v_i$ is maximal given this.

The number of possible solutions is very large, can be up to 2^n.

Solution - Greedy. Can we use a greedy algorithm? Always taking the most expensive item, or always taking the largest specific value (dividing by weight or something). □

Solution - Dynamic. Symbols: For $1 \leq i \leq n$ and $0 \leq u \leq W$, we will write $k[i, u]$ to be the optimal solution of the complete knapsack problem with the items $\{i, i+1, \ldots, n\}$, and the remaining weight in the knapsack is u

First recursive formula: $k[1, W] = \max\{v_1 + k[2, W - w_1], k[2, W]\}$

General recursive formula: $k[i, u] = \begin{cases} v_n, & \text{if } i = n, w_n \leq u \\ 0, & \text{if } i = n, w_n > u \\ k[i+1, u], & \text{if } i < n, w_i > u \\ \max\{v_i + k[i+1, u - w_i], k[i+1, u]\}, & \text{if } i < n, w_i \leq u \end{cases}$

Number of different sub problems: The sub problems of the complete knapsack with the values $\{i+1, \ldots, n\}$ are $0 \leq u \leq W$. The set of weights are the numbers of the style $\left\{W - \sum_{i \in S} w_i\right\}_{S \subseteq [n]}$, in the assumption that these numbers are non negative.

Facilitating assumption: we will assume that W and all the weights $w_1, \ldots, w_n$ are natural numbers. So all the weights u in the sub problem will be whole numbers between 0 and W, and in total there will be $W + 1$ numbers such as this.

Given the assumption that W, and all the weights $w_1, \ldots, w_n$ are natural numebrs, the number of different sub problems are at most the number of pairs $1 \leq i \leq n : (i, u)$. u is a natural numbers between 0 and W, which is $n \cdot (W + 1)$

Algorithm: We will define the table T woth n rows, and $W + 1$ columns, and we want $T[i, u] = k[i, u]$.

1. **Initialisation:** We will define $T[n, u] = \begin{cases} 0, & \text{if } u < w_n \\ v_n, & \text{if } u \geq w_n \end{cases}$
2. **Iteration:** We will fill in the rest of the table in $n - 1$ iterations, where in the $1 \leq k \leq n - 1$, we will fill in the row $i = n - k$ according to the following formula: $T[i, u] = \begin{cases} T[i+1, u], & \text{if } w_i > u \\ \max\{v_i + T[i+1, u - w_i], T[i+1, u]\}, & \text{if } w_i \leq u \end{cases}$
3. End: We return $T[1, W]$

Runtime: $O(nW)$ □

2 Approximation algorithms

Example 2 (Load balancing/scheduling)**.** ***Input:*** *A number k of identical machines, and n jobs of length $t_1, \ldots, t_n$ which need to occur (on the machines).*
Objective: *to find a schedule $S : [n] \to [k]$ which has the minimal finish time for the most busy machine.*

$$L_j(S) = \sum_{i=1 \wedge S(i)=j}^{n} t_i \wedge q(S) = \max_{j \in [k]} L_j(S)$$

In other words, we want to find S^* which brings $q(S^*)$ to the minimum value. Example: 2 machines, length of tasks: $\{1, 5, 1, 2\}$. So we assign 5 to the first machine, and $\{1, 1, 2\}$ to the second machine, and thus $q(S^*) = 5$

This problem has no solution in polynomial time. Even for $k = 2$ the problem is NP-hard.

***New objective:** To find an efficient algorithm (polynomial) that brings a solution that is "close enough". The algorithm A approaches-α (for $\alpha > 1$) if $\frac{q(A)}{q(S^*)} < \alpha$ for every input k and $t_1, \ldots, t_n$*

Solution - Greedy algorithm. We shall order the tasks arbitrarily, and at every step we will put the task on the machine which is the least busy at the given step.

Theorem 1. *The greedy algorithm approaches $\left(2 - \frac{1}{k}\right)$ for the load balancing problem.*

Proof (relying on the two lemmas). We will write J^* to be the most busy machine in G. We shall assume that it takes the final task in the iteration $1 \leq l \leq n$, (we will assume that the tasks are sorted $t_1, \ldots, t_n$). We will define for every machine j $L'_i(G)$ to be the business of the jth machine until the iteration $l - 1$, so: $L'_j(G) = \sum_{i=1:G(i)=j}^{l-1} t_i$. $L_{jk}(G) - L_{jk}(G)t_l$.

In the step where we chose where to put l:

$$\begin{aligned}
q(G) &= \max_{j \in [k]} (G) \\
&= L_{J^*}(G) \\
&= L'_{J^*}(G) + t_l \\
&= \min_{j \in [k]} L'_j(G) + t_l \\
&\leq \frac{1}{k} \sum_{j=1}^{k} L'_j(G) + t_l \\
&= \frac{1}{k} \sum_{j=1}^{k} \sum_{i=1}^{l-1} t_i + t_l \\
&= \frac{1}{k} \sum_{i=1}^{l} t_i + \left(1 - \frac{1}{k}\right) t_l \\
&= \frac{1}{k} \sum_{i=1}^{n} t_i + \left(1 - \frac{1}{k}\right) t_{max} \\
&\leq \left(2 - \frac{1}{k}\right) q(S^*)
\end{aligned}$$

□

Theorem 2 (Lemma 1)**.** *Let there be t_{max} the longest task. $q(S^*) \geq t_{max}$*

Proof . Let there be S^*, the optimal schedule, $q(S^*) = \max_{j \in [k]} L_J(S^*) \geq L_{S^*(t_{max}}(S^*) \geq t_{max}$ □

Theorem 3 (Lemma 2)**.** $q(S^*) \geq \frac{1}{k} \sum_{i=1}^{n} t_i$

Proof .

$$\begin{aligned}
q(S^*) &= \max_{j \in [k]} (L_j(S^*)) \\
&\geq \frac{1}{k} \sum_{j=1}^{k} L_j(S^*) \\
&= \frac{1}{k} \sum_{j=1}^{k} \sum_{i=1:L(i)=j}^{n} t_i \\
&= \frac{1}{k} \sum_{i=1}^{n} t_i
\end{aligned}$$

□

$\square$