Q1 Interval scheduling and partitions revisited

Given $m$ machines and a set of intervals $S$ with start times $s_i$ and finishing times $f_i$, output a feasible mapping $\sigma : S \rightarrow \{0, 1, 2, \ldots, m\}$ where $\sigma(I) = k > 0$ means that interval $I$ is schedule on machine $k$ and $\sigma(I) = 0$ means that interval $I$ is not scheduled. The mapping is feasible if $\sigma(I_i) = \sigma(I_j) = k > 0$ implies $I_i$ and $I_j$ do not intersect (the intervals are compatible). The objective is to maximize $|I : \sigma(I) > 0|$.

**Algorithm:** Sort the intervals such that $f_1 \leq f_2 \leq \ldots \leq f_n$. For $i = 1, \ldots, n$ we determine $\sigma(I_i)$ as follows. For machine $k \in [m]$ let $J_{ik}$ be the most recent interval that has been assigned to machine $k$ so far, if such an interval exists. More formally, $J_{ik} = I_\ell$ for the maximum $\ell < i$ such that $\sigma(I_\ell) = k$, if such an $\ell$ exists. If $J_{ik} = I_\ell$ is well defined then let $a_{ik} = f_\ell$ ("a" for "available"), i.e. $a_{ik}$ is the finishing time of the most recent interval assigned to machine $k$. Otherwise let $a_{ik} = 0$. Finally, if $a_{ik} > s_i$ for all $k$ then let $\sigma(I_i) = 0$; otherwise let $\sigma(I_i) = k$ for some $k$ that maximizes $a_{ik}$ subject to $a_{ik} \leq s_i$.

**Correctness:** Let $G_i$ be the greedy schedule after the first $i$ intervals have been processed. We prove by induction on $i = 0, \ldots, n$ that there exists an optimal solution $OPT_i$ that extends $G_i$ (in other words, $OPT_i$ and $G_i$ assign the first $i$ intervals in the same way). When $i = n$ there are no more intervals to schedule so $G_n$ must be optimal.

The base case is $i = 0$ and $G_0 = \emptyset$, so clearly there exists an optimal solution $OPT_0$ that extends $G_0$. Assume the inductive hypothesis holds for $i - 1$. We prove it for $i$ by considering the following cases:

- $I_i$ is not scheduled on any machine in $G_i$. This implies that no legal extension of $G_{i-1}$ can schedule $I_i$ on any machine. In particular, $OPT_{i-1}$ does not schedule $I_i$ on any machine. Since $OPT_{i-1}$ extends $G_{i-1}$, it follows that $OPT_{i-1}$ extends $G$ as well.

- $I_i$ is scheduled on the same machine in $G_i$ and $OPT_{i-1}$. Then clearly $OPT_{i-1}$ extends $G_i$.

- $I_i$ is scheduled on some machine $k$ in $G_i$, but on no machine in $OPT_{i-1}$. Let $I_j$ be the next interval scheduled on machine $k$ in $OPT_{i-1}$, if such an interval exists.\(^1\) Define $OPT_i$ by replacing $I_j$ with $I_i$ in $OPT_{i-1}$. This is possible because of the following two observations:
  - At the start time $s_i$ of $I_i$, machine $k$ is free in $OPT_{i-1}$. This is because $OPT_{i-1}$ extends $G_{i-1}$, and by the construction of $G_i$, at time $s_i$ machine $k$ is free in $G_{i-1}$.

\(^1\) Such an $I_j$ must exist, because otherwise we could add $I_i$ to $OPT_{i-1}$, contradicting the optimality of $OPT_{i-1}$.
At the finish time $f_i$ of $I_i$, machine $k$ is free in $OPT_{i-1}$ if $I_j$ is removed. This is because $f_i \leq f_j$.

- $I_i$ is scheduled on some machine $k$ in $G_i$, and some different machine $k'$ in $OPT_i$. Construct $OPT_i$ as follows: start with $OPT_{i-1}$, and for all $j \geq i$, if $I_j$ is scheduled on machine $k$ (resp. $k'$) then move it to machine $k'$ (resp. $k$). This results in a valid ordering because $a_{ik'} \leq a_{ik} \leq s_i$, by the nature of the greedy algorithm.

Q2 Making change

Consider the problem of making change for some amount of money $n$, given denominations $1 = C[1] < C[2] < ... < C[k]$ (for example, Canadian coins come in denominations $C[1] = 1, C[2] = 5, C[3] = 10, C[4] = 25, C[5] = 100, C[6] = 200$). Say that we want the output to be a sequence of coins $S = u_1, u_2, ..., u_m$ (where each $u_i = C[j]$ for some $j$) such that we use as few coins as possible (i.e., $n = u_1 + u_2 + ... + u_m$ and $m$ is minimal).

Give a greedy algorithm that makes change for any amount $n \geq 0$ using as few coins as possible, for the Canadian denominations. What is the runtime of your algorithm? Prove that your algorithm always returns an optimal answer.


solution

Algorithm: For $j$ from $k$ down to 1, add $\lfloor n/C[j] \rfloor$ copies of $C[j]$ to the set of coins to be output, and update $n$ to $(n \mod C[j])$.

Runtime: $O(k)$, assuming that arithmetic takes constant time, and that we’re allowed to express the output as an array $A[1...k]$ where $A[j]$ is the number of copies of $C[j]$ in the sequence. If $k$ is constant and we’re required to write the sequence $S = u_1, u_2, ..., u_m$ explicitly, then the runtime is $O(n)$.

Correctness: It will be more convenient to analyze the following algorithm, which is equivalent to the above but has a different runtime. For $j$ from $k$ down to 1, while $C[j] \leq n$, add a copy of $C[j]$ to the output and update $n$ to $n - C[j]$.

Observe the following:

- No optimal solution has more than four 1-cent coins, because five 1-cent coins can be replaced with a 5-cent coin.
- No optimal solution has more than one 5-cent coin, because two 5-cent coins can be replaced with a 10-cent coin.
• No optimal solution has more than two (5 or 10)-cent coins, because three 10-cent coins can be replaced with a 25-cent coin and a 5-cent coin, and two 10-cent coins and a 5-cent coin can be replaced with a 25-cent coin.

• No optimal solution has more than three 25-cent coins, because four 25-cent coins can be replaced with a 100-cent coin.

• No optimal solution has more than two 100-cent coins, because two of them can be replaced with a 200-cent coin.

Now assume that this algorithm does not always produce an optimal solution. This implies that there exist \( n \) and \( j \) such that \( C[j] \leq n \) and \( C[j + 1] > n \), but the optimal change for \( n \) does not have any copy of \( C[j] \). Consider the following cases:

• \( C[j] = 1 \). Then \( 1 \leq n \leq 4 \), so clearly we need 1-cent coins.

• \( C[j] = 5 \). Then \( 5 \leq n < 10 \), so an optimal solution needs at least five 1-cent coins, which is a contradiction.

• \( C[j] = 10 \). Then \( 10 \leq n < 25 \). But in an optimal solution, there are at most four 1-cent coins and at most one 5-cent coin, for a total of \( 4 \times 1 + 1 \times 5 = 9 < 10 \leq n \) cents, a contradiction.

• \( C[j] = 25 \). Then \( 25 \leq n < 100 \). But in an optimal solution, there are at most two (5 or 10)-cent coins and at most four 1-cent coins, for a total of \( 4 \times 1 + 2 \times 10 = 24 < 25 \leq n \) cents, a contradiction.

• \( C[j] = 100 \). Then \( 100 \leq n < 200 \). But in an optimal solution, there are at most three 25-cent coins, two (5 or 10)-cent coins, and four 1-cent coins, for a total of \( 3 \times 25 + 2 \times 10 + 4 \times 1 = 99 < 100 \leq n \) cents, a contradiction.

• \( C[j] = 200 \). Then \( 200 \leq n \). An optimal solution has at most one 100-cent coin. By similar reasoning as above, an optimal solution has at most 199 cents, a contradiction.

If we add a 75 cents coin, then greedy does not work for \( n = 150 \), because the optimal solution is 75 + 75 but greedy gives 100 + 25 + 25.

If instead we add a 30 cents coin, then greedy does not work for \( n = 50 \), because the optimal solution is 25 + 25 but greedy gives 30 + 10 + 10.

Q3 MSTs

(a) Prove or disprove: If \( e \) is a minimum-weight edge in connected graph \( G \) (where not all edge weights are necessarily distinct), then every minimum spanning tree of \( G \) contains \( e \).

(b) Does your answer change if \( w(e) \) is unique (no other edge in \( G \) has the same weight as \( e \), but \( w(e) \) is still the smallest)? Again, prove your answer.

solution
(a) False. Let $G$ be a cycle in which all edges have the same weight. Removing any one edge of $G$ creates an MST.

(b) Yes, every MST contains $e$ in this case. This follows from the correctness of Kruskal’s algorithm. For an alternate proof: let $T$ be a spanning tree without $e$. Adding $e$ to $T$ creates a cycle in $T$, and removing any other edge from that cycle creates a spanning tree with lower weight.

Q4 Cops and robbers

Given an array of size $n$ that has the following specifications: Each element in the array contains either a cop or a robber. Each cop can catch only one robber. A cop cannot catch a robber who is more than $K$ units away from the cop. Write an algorithm to find the maximum number of robbers that can be caught.

solution

Algorithm: Initialize $c$ to the first cop in the array, and $r$ to the first robber in the array. While $c$ and $r$ are not null, do the following. If $|c - r| \leq K$ then assign $c$ to catch $r$, and increment $c$ and $r$ to the next cop and robber respectively. Else, if $c < r$ (resp. $r < c$) then increment $c$ to the next cop (resp. $r$ to the next robber).

Correctness: Let $G_i$ be the greedy assignment after the $i$’th iteration of the greedy algorithm. We prove by induction on $i$ that there exists an optimal assignment $OPT_i$ that extends $G_i$. When $i$ is the last iteration there are no more robbers to assign or no more cops to assign, so this solution must be optimal.

The base case is $i = 0$ and $G_0 = \emptyset$, so clearly there exists an optimal solution $OPT_0$ that extends $G_0$. Assume the inductive hypothesis holds for $i - 1$. Let $c$ and $r$ be the cop and robber considered in the $i$’th iteration. We prove the inductive hypothesis for $i$ by considering the following cases:

- $G_i$ does not match $r$ with $c$. Then $|r - c| > K$, so $OPT_{i-1}$ doesn’t match $r$ with $c$ either. Since $OPT_{i-1}$ extends $G_{i-1}$, it follows that $OPT_{i-1}$ extends $G_i$ as well, so we can let $OPT_i = OPT_{i-1}$.
- $G_i$ matches $r$ with $c$, and
  - $OPT_{i-1}$ doesn’t match $r$, or $OPT_{i-1}$ doesn’t match $c$. Form $OPT_i$ from $OPT_{i-1}$ by unmatching $r$ or $c$, and then matching $r$ with $c$. This doesn’t decrease the total number of matchings, so $OPT_i$ must be optimal because $OPT_{i-1}$ is optimal. Clearly $OPT_i$ extends $G_i$.
  - $OPT_{i-1}$ matches $c$ with $r$. Then let $OPT_i = OPT_{i-1}$.
  - $OPT_{i-1}$ matches $c$ with $r'$, and $r$ with $c'$, where $r \neq r'$ and $c \neq c'$. Since $r < r'$ and $c < c'$, the largest-indexed of \{r, r', c, c'\} must be either $r'$ or $c'$. Assume without loss of generality that it’s $c'$. Then since $OPT_{i-1}$ matches $c'$ and $r$, it follows that
\[ K \geq |c' - r| = c' - r > c' - r' = |c' - r'|. \] Therefore we can define \( OPT_i \) by starting with \( OPT_{i-1} \), and replacing the matchings \((r, c')\) and \((r', c)\) with \((r, c)\) and \((r', c')\).