Esercizi su Greedy

Exercise 1.1 Let $S = \{[a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]\}$ be a set of closed intervals on the real line. We say that $C \subseteq S$ is a *covering subset* for S if, for any interval $[a, b] \in S$, there exists an interval $[a', b'] \in C$ such that $[a, b] \subseteq [a', b']$ (that is, $a \geq a'$ and $b \leq b'$).

(a) Write a greedy $O(n \log n)$ algorithm that, on input S, returns a covering subset C^* of minimum size.

(b) Prove the greedy choice property for the above algorithm.

(c) Prove the optimal substructure property for the problem.

Answer:

(a) We first sort the intervals in nondecreasing order of their left endpoint. Ties are broken as follows: if $a_i = a_j$ then $[a_i, b_i]$ precedes $[a_j, b_j]$ in the sorted sequence if $b_i > b_j$. Otherwise, if $b_i < b_j$, $[a_j, b_j]$ precedes $[a_i, b_i]$ (note that it cannot be the case that $b_i = b_j$ or the two intervals would be the same). In other words, ties are broken in favor of the larger interval. Once the intervals are so sorted, we perform a linear scan of the intervals. The greedy choice is to select the first interval. At the i-th iteration, let $[a_k, b_k]$ denote the last interval that was selected. Interval $[a_i, b_i]$ is discarded if it is contained in $[a_k, b_k]$, and is selected otherwise.

We assume that the intervals are given in input stored into an array of records S, with $S[i]$.left $= a_i$ and $S[i]$.right $= b_i$. In the code, procedure SORT(S) performs the required sorting of the intervals, so that, on its termination, $S[i]$ stores the i-th interval of the sorted sequence. Note that a trivial adaptation of MERGE_SORT suffices to yield an $O(n \log n)$ running time for SORT. The algorithm follows:

```
MIN_COVERING(S)
n ← length(S)
SORT(S)
C[1] ← S[1]
j ← 2
{ j stores the index of the first empty location in vector C }
    k ← 1
    { k stores the index of the last selected interval }
    for i ← 2 to n do
        if S[i].right > S[k].right
          then C[j] ← S[i]
          j ← j + 1
          k ← i
          { S[i] is now the last selected interval }
    return C
```

The running time $T_{\text{M\_C}}(n)$ of MIN_COVERING(S) is clearly dominated by the time required by the initial sort, hence $T_{\text{M\_C}}(n) = O(n \log n)$.

(b) MIN_COVERING clearly exhibits the greedy choice property. In fact, *every* covering subset must contain $S[1]$, since by construction $S[1]$ is not contained in any other interval in S.

(c) Let $S = \{[a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]\}$ and assume that the intervals in S have been renumbered so to comply with the ordering enforced by procedure SORT above. Let $C^\star = \{[a_1, b_1]\} \cup \overline{C}^\star$ be a covering subset of minimum size for S. The optimal substructure property that the algorithm MIN_COVERING exploits is the following: letting k be the maximum index such that $b_j \leq b_1$, for $1 \leq j \leq k$ (note that $a_j \geq a_1$ also holds, because of the ordering), then $\overline{C}^\star$ must be an optimal solution to $\overline{S} = S - \{[a_j, b_j] : 1 \leq j \leq k\}$ (observe that $\overline{S}$ could be empty). To prove this, let us first prove that $\overline{C}^\star$ is indeed a covering subset for $\overline{S}$. If $\overline{S} = \emptyset$, then $\overline{C}^\star$ is clearly empty, since $\{[a_1, b_1]\}$ is a covering subset for S. If $\overline{S}$ is not empty, then $\overline{C}^\star$ must be a covering subset for all intervals in S not contained in $[a_1, b_1]$, since $C^\star = \{[a_1, b_1]\} \cup \overline{C}^\star$ is a covering subset. Therefore, $\overline{C}^\star$ must contain $[a_{k+1}, b_{k+1}]$, which is not covered by any other interval in S. Note, however, that $\overline{S}$ might also contain intervals which are contained in $[a_1, b_1]$. Let $[a_s, b_s]$ be one of such intervals. First note that it must be $s > k + 1$, since the interval is in $\overline{S}$. Then, the

following chain of inequalities is easily established:

$$a_1 \leq a_{k+1} \leq a_s \leq b_s \leq b_1 < b_{k+1},$$

hence $[a_s, b_s]$ is covered by $[a_{k+1}, b_{k+1}] \in \overline{C}^\star$. To finish the proof it suffices to observe that if $\overline{C}^\star$ were not a covering subset of $\overline{S}$ of minimum size, then $C^\star$ would not be a covering subset of minimum size for S. □

Exercise 1.2 Let $C = \{1, 2, \ldots, n\}$ denote a set of n rectangular frames. Frame i has base $d[i].b$ and height $d[i].h$. We say that Frame i *encapsulates* Frame j if $d[i].b \geq d[j].b$ and $d[i].h \geq d[j].h$. (Note that a frame encapsulates itself). An *encapsulating subset* $C' \subseteq C$ has the property that for each $j \in C$ there exists $i \in C'$ such that i encapsulates j.

(a) Design and analyze a greedy algorithm that, on input the vector $\boldsymbol{d}$ of sizes of a set C of n frames, returns a *minimum size* encapsulating subset $C' \subseteq C$ in $O(n \log n)$ time.

(b) Prove the greedy choice property.

(c) Prove the optimal substructure property.

Answer: We first sort the entries of vector $\boldsymbol{d}$ in nonincreasing order of their $d[\cdot].b$ field. Ties are broken as follows: if $d[i].b = d[j].b$ then $d[i]$ precedes $d[j]$ in the sorted sequence if $d[i].h \geq d[j].h$. In other words, ties are broken in favor of the higher frame. Once the frames are so sorted, we perform a linear scan starting from the first frame in the sorted order. The greedy choice is to select the first frame. At the beginning of the i-th iteration, $i \geq 2$, let $j < i$ denote the last frame that was selected. Frame i is discarded if it is encapsulated by frame j, and is selected otherwise.

In the code below, procedure SORT($\boldsymbol{d}$) performs the required sorting of the frames, so that, on its termination, $d[i]$ stores the i-th frame of the sorted sequence. Note that a trivial adaptation of MERGE_SORT suffices to yield an $O(n \log n)$ running time for SORT. The algorithm follows:

```
MIN_ENCAPSULATING_SUBSET(d)
n ← length(d)
SORT(d)
C'[1] ← d[1]
k ← 2
{ k stores the index of the first empty location in vector C' }
j ← 1
{ j stores the index of the last selected frame }
for i ← 2 to n do
    if d[i].h > d[j].h
      then C'[k] ← d[i]
      k ← k + 1
      j ← i
      { i is now the last selected frame }
return C'
```

The running time $T_{\text{M\_E\_S}}(n)$ of MIN_ENCAPSULATING_SUBSET($\boldsymbol{d}$) is clearly dominated by the time required by the initial sort, hence $T_{\text{M\_E\_S}}(n) = O(n \log n)$.

In the following, we assume that the frames in C have been renumbered so to comply with the ordering enforced by procedure SORT.

(b) Greedy Choice Consider an arbitrary optimal solution $C^\star \subseteq C$. Such subset must contain a frame t which encapsulates Frame 1. Then, either $t = 1$, in which case $C^\star$ contains the greedy choice, or $t > 1$ and (1) $d[t].b \geq d[1].b$ and (2) $d[t].h \geq d[1].h$. In the latter case, due to the sorting, both (1) and (2) must be equalities, hence all frames encapsulated by Frame t are also encapsulated by Frame 1. Therefore $(C^\star - \{t\}) \cup \{1\}$ is an optimal solution containing the greedy choice.

(c) Optimal Substructure Let $C^\star = \{1\} \cup \overline{C}^\star$ be an encapsulating subset of minimum size for C containing the greedy choice. The optimal substructure property featured by the problem is the following: let $j \leq n+1$ be the maximum value such that $d[i].h \leq d[1].h$ for $1 \leq i < j$. Then $\overline{C}^\star$ must be an optimal solution to $\overline{C} = C - \{i : 1 \leq i < j\}$ (observe that $\overline{C}$ could be empty). To prove this, let us first prove that $\overline{C}^\star$ is indeed an encapsulating subset for $\overline{C}$. If $\overline{C} = \emptyset$, then $\overline{C}^\star$ is clearly an encapsulating subset. If $\overline{C}$ is not empty, then $\overline{C}^\star$ must be an encapsulating subset for all frames in C which are not encapsulated by Frame 1, since $C^\star = \{1\} \cup \overline{C}^\star$ is an encapsulating subset for C. Therefore, $\overline{C}^\star$ must

contain Frame j, which is not encapsulated by any other frame in C (indeed, $\overline{C}^\star$ could contain a frame j' with the same base and height as j, but then we can substitute j' with j in $\overline{C}^\star$ and proceed with the argument). Note, however, that $\overline{C}$ might also contain frames which are encapsulated by Frame 1. Let s be one of such frames. First note that it must be $s \geq j$, since the frame is in $\overline{C}$. Then, $d[j].b \geq d[s].b$ (by the ordering) and

$$d[j].h > d[1].h \geq d[s].h,$$

hence Frame s is also encapsulated by $j \in \overline{C}^\star$.

To finish the proof it suffices to observe that if $\overline{C}^\star$ were not an encapsulating subset of $\overline{C}$ of minimum size, then $C^\star$ would not be an encapsulating subset of C of minimum size. □

Exercise 1.3 Let $n > 0$. Given a set of positive real numbers $S = \{a_1, a_2, \ldots, a_n\}$, with $0 < a_i < a_j < 1$ for $1 \leq i < j \leq n$, a *(2,1)-boxing* of S of size k is a partition $\mathcal{P} = \{S_1, S_2, \ldots, S_k\}$ of S into k disjoint subsets (that is, $\bigcup_{j=1}^{k} S_j = S$ and $S_r \cap S_t = \emptyset, 1 \leq r \neq t \leq k$) which satisfies the following constraints:

$$|S_j| \leq 2 \quad \text{and} \quad \sum_{a \in S_j} a \leq 1, \;\; 1 \leq j \leq k.$$

A practical instance of $(2,1)$-boxing could be the following: the s_i's may correspond to the weights of different items to be boxed, where a box cannot contain more than two items whose combined weight cannot exceed 1 unit of weight. For instance, if $S = \{0.2, 0.6, 0.7\}$, an optimal 2-boxing is $\{\, \{0.2, 0.7\}, \{0.6\} \,\}$. We want to determine a $(2,1)$-boxing of minimum size.

Point 1 Give the pseudocode of a greedy algorithm which returns an optimal $(2,1)$-boxing in linear time.

Point 2 Prove that the algorithm has the greedy choice property.

Point 3 Prove that the algorithm has the optimal substructure property.

Answer: Point 1 The greedy strategy works as follows: we try to pair the smallest (a_1) and the largest (a_n) element in S (only if $n > 1$). In case their sum is at most one, we create $S_1 = \{a_1, a_n\}$. In all other cases, we create $S_1 = \{a_n\}$, since no other element, if any, could possibly be paired with a_n legally. We then proceed on the residual problem. The algorithm follows.

```
2-1_BOXING(S)
n ← |S|
⋆ Let S = {a_1, a_2, ..., a_n} ⋆
P ← ∅; first ← 1; last ← n
while (first ≤ last)
    if ((first < last) and (a_first + a_last ≤ 1))
       then P ← P ∪ { {a_first, a_last} }; first ← first + 1;
       else  P ← P ∪ { {a_last} }
    last ← last − 1
return P
```

The above algorithm scans each element once, hence it requires linear time in n.

Point 2 Observe that the greedy choice is $\{a_1, a_n\}$ if $n > 1$ and $a_1 + a_n \leq 1$, and $\{a_n\}$ otherwise. We have to prove that there is always an optimal solution containing the greedy choice. The proof is trivial for $n = 1$, or for $n > 1$ and $a_1 + a_n > 1$, since in these cases any feasible solution must contain the subset $\{a_n\}$. Therefore let us assume that $n > 1$ and that the greedy choice is $\{a_1, a_n\}$. Consider an arbitrary optimal solution, and assume that a_1 and a_n are not paired together. Then, there exist two subsets S_1 and S_2, with $|S_1|, |S_2| \leq 2$ and $S_1 \cap S_2 = \emptyset$, such that $a_1 \in S_1$ and $a_n \in S_2$. We can substitute these two sets with $S'_1 = \{a_1, a_n\}$ (this is the greedy choice, hence it is a legal subset), and $S'_2 = S_1 \cup S_2 - \{a_1, a_n\}$. Indeed, $|S'_2| \leq 2 + 2 - 2 = 2$, and whenever $|S'_2| = 2$, then $S'_2 = \{a_s, a_t\}$ with $a_s \in S_1$ and $a_t \in S_2$, but then $a_s < a_n$ (since $s < n$) and $a_t \leq 1 - a_n$ (since a_t was legally paired with a_n in S_2), hence $a_s + a_t < 1$. Therefore the new solution which includes these two new subsets is feasible and still optimal.

Point 3 Any optimal solution $\mathcal{P}^\star$ which contains the greedy choice, also contains a (2,1)-boxing $\mathcal{P}'$ of the residual set of values (either $\{a_1, \ldots, a_{n-1}\}$ or $\{a_2, \ldots, a_{n-1}\}$). If $\mathcal{P}'$ were not optimal, we could improve on the optimal solution for the original problem by considering a (2,1)-boxing obtained by adding the greedy choice to the optimal (2,1)-boxing $\mathcal{P}'^\star$ for the residual problem, a contradiction. □

Exercise 1.4 Given n programs $P_1, P_2, \ldots, P_n$, assume that the running time of P_i on a given machine $\mathcal{M}$ be t_i, with $1 \leq i \leq n$. An *execution order* of the n programs on $\mathcal{M}$ is a permutation of the indices $\pi : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$, prescribing that the programs are to be executed on $\mathcal{M}$ one after the other, according to the sequence $\langle P_{\pi(1)}, P_{\pi(2)}, \ldots, P_{\pi(n)} \rangle$. For a given execution order π, the *wait time* of the i-th executed program in the order, for $i \geq 2$, is clearly $\tau_i = \sum_{j=1}^{i-1} t_{\pi(j)}$, and the *cumulative wait time* π

is $A_\pi = \sum_{i=2}^{n} \tau_i$. We want to determine the execution order $\pi^\star$ associated to the minimum cumulative wait time $A_{\pi^\star}$.

Prove the following greedy choice property: *every* optimal execution order $\pi^\star$ is such that $t_{\pi^\star(1)} = \min\{t_i : 1 \leq i \leq n\}$. In other words the execution order which minimizes the cumulative wait time must schedule the n programs by nondecreasing running time.

Answer: For a given execution order π, observe that

$$A_\pi = \sum_{i=2}^{n} \tau_i = \sum_{i=2}^{n} \sum_{j=1}^{i-1} t_{\pi(j)}.$$

In the above double summation, the term $t_{\pi(1)}$ (the running time of the first job executed in the order) appears $n-1$ times, the term $t_{\pi(2)}$ appears $n-2$ times, and so forth. In general, for $1 \leq k \leq n-1$, the term $t_{\pi(k)}$ appears $n-k$ times. Hence, we can rewrite A_π as follows:

$$A_\pi = \sum_{k=1}^{n-1} (n-k) t_{\pi(k)}.$$

Let $\pi^\star$ be an optimal execution order and set $i_{\min}$ such that $t_{i_{\min}} = \min\{t_i : 1 \leq i \leq n\}$. Assume now, for the sake of contradiction, that $t_{\pi^\star(1)} \neq \min\{t_i : 1 \leq i \leq n\}$. Therefore, $\pi^\star(1) = i_1$, with $t_{i_1} > t_{i_{\min}}$ and there is a value $h > 1$ such that $\pi^\star(h) = i_{\min}$. We can then obtain a new execution order $\hat{\pi}$ as follows:

$$\hat{\pi}(k) = \begin{cases} i_{\min} & k = 1 \\ i_1 & k = h \\ \pi^\star(k) & \text{otherwise} \end{cases}$$

In other words, in $\hat{\pi}$ we swap the order of execution of $P_{i_{\min}}$ (which now becomes the first job to be executed) and P_{i_1} (which becomes the h-th job to be executed). Then we have:

$$\begin{aligned} A_{\pi^\star} - A_{\hat{\pi}} &= \sum_{k=1}^{n} (n-k) t_{\pi^\star}(k) - \sum_{k=1}^{n} (n-k) t_{\hat{\pi}(k)} \\ &= (n-1) t_{i_1} + (n-h) t_{i_{\min}} - (n-1) t_{i_{\min}} - (n-h) t_{i_1} \\ &= (n-1)(t_{i_1} - t_{i_{\min}}) - (n-h)(t_{i_1} - t_{i_{\min}}) \\ &= (h-1)(t_{i_1} - t_{i_{\min}}) \\ &> 0, \end{aligned}$$

since $h > 1$ and $t_{i_1} > t_{i_{\min}}$. This is a contradiction, since we were assuming that $\pi^\star$ was an optimal execution order, which should be associated with the minimum cumulative wait

time $A_{\pi^\star}$. □

Exercise 1.5 Given a set $S = \{s_1, s_2, \ldots, s_n\} \subseteq \boldsymbol{R}^+$, with $s_1 < s_2 < \ldots < s_n$, and a positive real value $\delta < 1$, a δ-*trimming* of S is a subset $T \subseteq S$ such that

$$\forall s \in S \; \exists t \in T : s(1-\delta) \leq t \leq s.$$

In other words, for each element $s \in S$ there is an element $t \in T$ that approximates s from below with relative error $(s-t)/s \leq \delta$. Trimming is a useful primitive to obtain small, accurate summaries of large datasets. Given S and δ, we want to determine δ-trimming of minimum-cardinality.

Point 1 Give the pseudocode of a greedy algorithm which returns such a δ-trimming in linear time.

Point 2 Prove that the algorithm has the greedy choice property.

Point 3 Prove that the algorithm has the optimal substructure property.

Answer: Point 1 In the trimming problem the greedy choice is simply to pick the first value $s_1 \in S$. In fact, this is somehow a forced choice, since the minimum element in the set S can only be approximated from below by itself! Once the greedy choice is made, the clean-up simply removes all subsequent points that are well approximated by s_1, that is, points s for which it holds that $s(1-\delta) \leq s_1$ (observe that the second condition $s_1 \leq s$ is guaranteed by the initial ordering). The first value s_i, if any, for which $s_i(1-\delta) > s_1$ will mark the start of the residual subproblem $S_r = \{s_i, s_{i+1}, \ldots, s_n\}$ (or otherwise the residual subproblem is the empty set). Observe that s_i will become the next greedy choice. The algorithm follows immediately.

```
GREEDY_TRIM(s, δ)
n ← s.len
g ← 1; T ← {s_g}
for i ← 2 to n do
    if (s_i(1 − δ) > s_g) then
        g ← i; T ← T ∪ {s_g}
return T
```

The running time of the above algorithm is clearly linear in n.

Point 2 As we mentioned above, we can prove a very strong greedy choice property: every optimal solution must contain the greedy choice. This is true because the minimum of S, s_1, is the only element in S less than or equal to itself, hence s_1 can only be approximated from below by s_1 itself.

Point 3 The optimal substructure property that we prove, in contrast, is weaker but sufficient to prove that the algorithm is correct. We will prove that *there exists* an optimal solution which, together with the greedy choice s_1, contains an optimal solution to the residual subproblem $S_r = \{s_i, s_{i+1}, \ldots, s_n\}$. Indeed, for this problem, not all optimal solutions containing the greedy choice also contain an optimal solution to the residual subproblem! However, this weaker property is sufficient by itself to guarantee correctness of the algorithm using an inductive argument. Let $\hat{T}$ be a generic optimal solution. As argued in Point 2, $\hat{T}$ must contain the greedy choice s_1, hence $\hat{T} = \{s_1\} \cup T'$. We have two cases: if $T' = \emptyset$ then all points in S are well approximated from below by s_1, which implies that $S_r = \emptyset$, whose (unique, hence optimal) solution is indeed $T' = \emptyset$. If $T' \neq \emptyset$, then T' must contain values which approximate all points in S_r, since none of this values is well approximated by s_1. If $T' \subseteq S_r$ we are done, since clearly T' must be optimal for S_r or we could obtain a better solution than $\hat{T}$ for the whole set. However, it may be the case that $T' \not\subseteq S_r$ beacuse it contains a single value s_k that was removed during the clean-up, since $s_k(1-\delta) \leq s_1$, but which is used to approximate a point $s_j \in S_r$, since $s_j(1-\delta) \leq s_k \leq s_j$. However, in this case we can substitute s_k with the greedy choice s_i since $s_j(1-\delta) \leq s_k \leq s_i \leq s_j$ (recall that s_k was eliminated in the clean-up, hence $k \leq i$, and also s_i is the minimum value in S_r hence $s_i \leq s_j$). Thus, $T' - \{s_k\} \cup \{s_i\} \subseteq S_r$ has the same cardinality of T' and is thus an optimal solution for S_r. □

Exercise 1.6 Consider a variant of the *Activity Selection Problem*, where the input set of intervals $S = \{[s_1, f_1), ..., [s_n, f_n)\}$ is sorted by *non increasing* values of the s_i's, that is, $s_1 \geq s_2 ... \geq s_n$. As in the original problem, we want to determine a maximum set of pairwise disjoint activities.

Point 1 Design an $O(n)$ algorithm for the above problem.

Point 2 Prove that the greedy choice property holds.

Point 3 Prove that the optimal substructure property holds.

Answer: Point 1 As for the variant seen in class, the greedy choice is to pick the first activity, that is, the one with the latest starting time, which guarantees to leave the largest possible single interval for allocating further compatible activities. The ensuing clean-up

consists in removing all immediately subsequent activities which are not compatible with the greedy choice a_1. Under the given ordering, these are those activities a_j, for $j > 1$, whose finish time is greater that s_1. The first activity a_i compatible with the current greedy choice, if any, will mark the start of the residual subproblem (or otherwise the residual subproblem is the empty set). Observe that a_i will become the next greedy choice. The algorithm follows immediately.

GREEDY_SEL($\boldsymbol{s}, \boldsymbol{f}$)
$n \leftarrow \boldsymbol{s}.len$
$g \leftarrow 1$; $A \leftarrow \{a_g\}$
for $i \leftarrow 2$ **to** n **do**
  if $(f_i \leq s_g)$ **then**
    $g \leftarrow i$; $A \leftarrow A \cup \{a_g\}$
return A

The running time of the above algorithm is clearly linear in n.

Point 2 To prove that there is an optimal solution which contain the greedy choice, let us start from a generic optimal solution $A^\star$. If $a_1 \in A^\star$ then we are done, otherwise, let $\hat{i} = \min\{j : a_j \in A^\star\} > 1$ and consider $\hat{A} = A^\star - \{a_{\hat{i}}\} \cup \{a_1\}$. We have that $|A^\star| = |\hat{A}|$ and for each index $j \neq 1$ such that $a_j \in \hat{A}$, we have that

$$f_j \leq s_{\hat{i}} \leq s_1,$$

since a_j was compatible with $a_{\hat{i}}$, and the starting time of a_1 is the latest among all starting times. This means that a_j is compatible with a_1, which proves that $\hat{A}$ is a feasible solution. Thus $\hat{A}$ is an optimal solution which contains the greedy choice.

Point 3 Consider now an optimal solution $\hat{A}$ containing the greedy choice and let $A' = \hat{A} - \{a_1\}$. We have two cases: if $A' = \emptyset$ then there can be no further activity compatible with a_1, which implies that the residual problem is the empty set, whose (unique, hence optimal) solution is indeed $A' = \emptyset$. If $A' \neq \emptyset$, then let $\hat{i}$ be the minimum index of an activity compatible with a_1, and observe that the residual problem is the set of activities $\{a_{\hat{i}}, a_{\hat{i}+1} \ldots, a_n\}$. Clearly A' contains only activities from the residual problem, since the clean-up only eliminates activities which are not compatible with a_1. Suppose that there is a larger solution $\bar{A}$ for the residual problem, and pick the one containing the greedy choice (for the residual problem), which is $a_{\hat{i}}$. Then we have that a_1 is compatible with $a_{\hat{i}}$ (by the definition of residual problem), and is also compatible with all other activities $a_j \in \bar{A}$, since these are compatible with $a_{\hat{i}}$, hence it must be $f_j \leq s_{\hat{i}} \leq s_1$. Then $\bar{A} \cup \{a_1\}$ is a

set of mutually compatible activities which yields a larger solution than $\hat{A}$ for the original problem, a contradiction. $\square$