CS 577-2: Introduction to Algorithms Fall 2020

Solutions to Midterm Exam 2

Instructor: Dieter van Melkebeek

Part (a)

Intuitively, and consistent with the example, it seems best to match the k-th tallest participant with the k-th longest snowboard for each $k \in [n]$. Let us call this matching G. If indeed optimal, we can compute the minimum total discrepancy by sorting the arrays h and ℓ, and then summing the squares of the component-wise differences. For completeness, we include pseudocode.

Algorithm 1

```
sort h[1...n] in nondecreasing order
sort ℓ[1...n] in nondecreasing order
s ← 0
for k = 1 to n do
    s ← s + (h[k] − ℓ[k])^2
return s
```

Correctness For $n = 2$ the optimality of G is equivalent to the following statement.

Claim 1. *Suppose $h_1 \le h_2$ and $\ell_1 \le \ell_2$. Then*

$$(h_1 - \ell_1)^2 + (h_2 - \ell_2)^2 \le (h_1 - \ell_2)^2 + (h_2 - \ell_1)^2. \tag{1}$$

Proof of Claim 1. We have

$$\begin{aligned}(1) &\Leftrightarrow h_1^2 - 2h_1\ell_1 + \ell_1^2 + h_2^2 - 2h_2\ell_2 + \ell_2^2 \le h_1^2 - 2h_1\ell_2 + \ell_2^2 + h_2^2 - 2h_2\ell_1 + \ell_1^2 \\ &\Leftrightarrow -2h_1\ell_1 - 2h_2\ell_2 \le -2h_1\ell_2 - 2h_2\ell_1 \\ &\Leftrightarrow 2(h_2 - h_1)\ell_1 \le 2(h_2 - h_1)\ell_2 \\ &\Leftarrow \ell_1 \le \ell_2\end{aligned}$$

The first line follows by expanding the squares in (1), the second line by canceling the common terms on both sides, and the third line by rearranging terms. The implication in the fourth line holds because $h_2 - h_1 \ge 0$ by the hypothesis $h_1 \le h_2$. The right-hand side of the fourth line is the hypothesis $\ell_1 \le \ell_2$. Thus, (1) follows. □

Optimality of G for general n follows from an exchange argument based on Claim 1. Consider any matching of participants and snowboards. It can be described as a permutation $S : [n] \to [n]$, where $S(i) = j$ means that we match the i-th element in the sorted array $h[1 \ldots n]$ with the j-th element in the sorted array $\ell[1 \ldots n]$. In this notation G corresponds to the identity permutation.

Any matching S other than G has to contain an inversion, i.e., a pair of elements $i_1, i_2 \in [n]$ such that $i_1 < i_2$ and $j_1 \doteq S(i_1) > S(i_2) \doteq j_2$. The order relations between the indices imply that

$$h[i_1] \le h[i_2] \text{ and } \ell[j_1] \ge \ell[j_2]. \tag{2}$$

Consider swapping j_1 and j_2 in S, yielding a new matching S'.

- By (2), Claim 1 applies to $h_1 \doteq h[i_1]$, $h_2 \doteq h[i_2]$, $\ell_1 \doteq \ell[j_2]$ and $\ell_2 \doteq \ell[j_1]$. This shows that the swap cannot increase the joint contribution of participants i_1 and i_2 to the total discrepancy. As the discrepancy of every other participant is unaffected by the swap, this means that the total discrepancy of S' is no larger than that of S.
- The swap eliminates the inversion (i_1, i_2). It does not affect the number of inversions that a position $i \in [n]$ outside of the interval $[i_1, i_2]$ is involved in, and cannot increase the number of inversions that a position $i \in [n]$ in the interval (i_1, i_2) is involved in. Thus, the number of inversions in S' is at least one less than the number of inversions in S.

We repeatedly apply an exchange of the above type as long as the matching differs from G. Since the number of inversions decreases by at least one for each application, and cannot go below 0, we need to end up with G. As no application increases the total discrepancy, this shows that G has a total discrepancy no larger than S. As S is arbitrary, we conclude that G has the minimum possible total discrepancy.

Alternate exchange arguments Instead of picking an arbitrary inversion (i_1, i_2) in S, one can make the following more specific choices:

- Pick an inversion of the form (i_1, i_1+1). As we argued in class, such an element $i_1 \in [n]$ needs to exists. Swapping based on such an inversion always decreases the number of inversions by exactly one.
- Consider the smallest $i_1 \in [n]$ such that $j_1 \doteq S(i_1) \neq G(i_1) \doteq j_2$. Let $i_2 \in [n]$ be the position in S where j_1 appears, i.e., $S(i_2) = j_1$. Since S agrees with G on all positions before i_1, it is necessarily the case that $i_1 < i_2$, and by the construction of G, that $j_1 > j_2$. Thus (i_1, i_2) constitutes on inversion in S. The swap increases the length of the prefix on which the permutation agrees with G by at least one.

Running time Sorting the arrays h and ℓ takes time $O(n \log n)$. Aggregating the total discrepancy of G takes time $O(n)$. Thus, the overall running time is $O(n \log n)$.

Part (b)

This problem can be efficiently reduced to minimal survey design. Recall that in the survey design problem we are given:

- customers $i \in [m]$,
- products $j \in [k]$,
- a set $P_i \subseteq [k]$ for each $i \in [m]$ of products that customer i can survey,
- the maximum number c_i of products that customer i can survey for each $i \in [m]$, and
- the minimum number p_j of customers to survey product j for each $j \in [k]$.

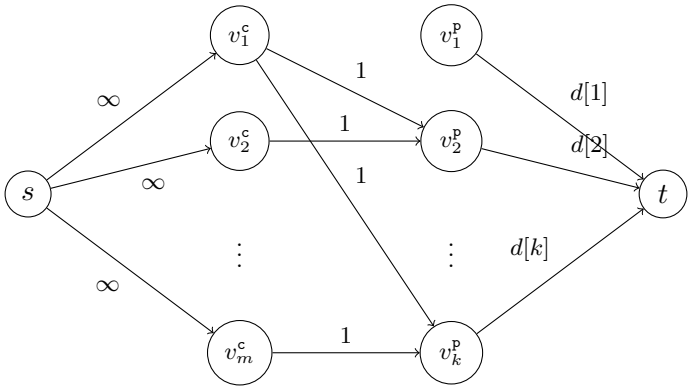
In the *minimal* survey design problem we additionally require that no product $j \in [m]$ is surveyed by more than the required number of customers p_j.

The description of the minimal survey design problem exactly matches the one of the problem we need to solve if we identify customers with instructors, products with days, set $P_i \doteq \{j \in [k] : A[i,j] = 1\}$, $c_i \doteq \infty$, and $p_j \doteq d[j]$. Thus, the algorithm from class to find a minimal survey design yields a correct solution to the given problem. As the instance of minimal survey design can be constructed in time $O(mk)$, and the algorithm from class runs in time $O((m+k) \cdot (m+k+\sum_{i\in[m]} |P_i|)) = O((m+k) \cdot mk)$, the resulting running time is $O((m+k)mk)$.

Reduction to max flow The algorithm for minimal survey design further reduces the problem to max flow. For future reference, we review the reduction. It constructs a network N with as vertices a source s, a sink t, one vertex $v_i^{\mathtt{c}}$ for each customer $i \in [m]$, and one vertex $v_j^{\mathtt{p}}$ for each product $j \in [k]$. The network has

- an edge of capacity c_i from the source s to customer vertex $v_i^{\mathtt{c}}$ for each $i \in [m]$,
- an edge of capacity 1 from customer vertex $v_i^{\mathtt{c}}$ to product vertex $v_j^{\mathtt{p}}$ for each $i \in [m]$ and $j \in P_i$, and
- an edge of capacity p_j from product vertex $v_j^{\mathtt{p}}$ to the sink t for each $j \in [k]$.

The resulting network for our instantiation is given in the figure below.



The analysis from class shows that there exists a solution S to the given problem iff the value of the maximum flow in N equals $D \doteq \sum_{j\in[k]} d[j]$, and that there is a one-to-one and onto correspondence between integral flows f of value D and solutions S to the problem. In particular, the flow f can be obtained from the solution S by routing one unit of flow from s over $v_i^{\mathtt{c}}$ and $v_j^{\mathtt{p}}$ to t for each pair $(i,j) \in S$.

Alternate more efficient solution Due to the lack of upper bound on the number of work days for any of the instructors (corresponding to $c_i = \infty$), the following simpler approach works: Start with the table A, and remove from every column (day) $j \in [k]$, excessive checked rows (instructors) above the required number $d[j]$, or report "no solution" if there are fewer than $d[j]$ checked rows in that column. This algorithm involves a constant amount of work per entry of A, thus $O(mk)$ in total. For completeness, pseudocode is provided.

Algorithm 2

```
1: S ← A
2: for j = 1 to k do
3:     r ← d[j]
4:     for i = 1 to m do
5:         if r = 0 then T[i, j] ← 0
6:         if S[i, j] = 1 then r ← r − 1
7:     if r > 0 then return "no solution"
8: return S
```

Part (c)

For each $i \in [m]$, let w_i denote the number of days that instructor i works in the given plan S. Let $W \doteq \max_{i \in [m]}(w_i)$ denote the maximum number of days that instructors work in S. A modified plan S' that accommodates the request of instructor i^* and does not increase the maximum number of days that instructors work, can be obtained as an instance of minimal survey design like in part (b) with the following modifications: We set $c_{i^*} \doteq w_{i^*} - 1$, and $c_i \doteq W$ for each $i \in [m] \setminus \{i^*\}$. The former assumes that $w_{i^*} \geq 1$; otherwise, it is certainly impossible to accommodate the request.

We can call the blackbox for minimal survey design on this instance as we did in part (b). The blackbox runs max flow on the network N' obtained from N by changing the capacities of the edges from the source s to the customer vertices as indicated. Integral flows of value $D \doteq \sum_{j \in [k]} d[j]$ in N' are in one-to-one and onto correspondence with solutions S' to the new problem. This yields an algorithm for part (c) that runs in time $O((m + k)mk)$.

In order to reduce the running time to $O(mk)$, we make use of the given plan S. Let f denote the integral flow corresponding to the given plan S in the network N. Let N'' be the modified network N' except that we set the capacity of the edge from the source s to $v^{\mathsf{c}}_{i^*}$ to w_{i^*} instead of $w_{i^*} - 1$. Note that f represents a valid flow in N'' as well.

Claim 2. *There exists a solution S' to problem (c) iff there exists a path from s to $v^{\mathsf{c}}_{i^*}$ in N''_f.*

Proof of Claim 2. $\Leftarrow$ If there exists a path from s to $v^{\mathsf{c}}_{i^*}$ in N''_f, then there exists a simple one. P, and we can extend P with the edge from $v^{\mathsf{c}}_{i^*}$ to s (which exists in N''_f since $w_{i^*} > 0$) to obtain a simple cycle P'. By path augmentation and the fact that N'' is integral, we can superimpose on f a flow of one unit along P'. The resulting flow f' is a valid integral flow in N'' of the same value as f. Moreover, since the edge from s to $v^{\mathsf{c}}_{i^*}$ is used by f at full capacity in N'', this edge is not present in N''_f and therefore not in P nor P'. It follows that f' sends at least one unit of flow less than f from s to $v^{\mathsf{c}}_{i^*}$. Thus, f' is a valid flow in N' of value $D \doteq \sum_{j \in [k]} d[j]$, and corresponds to a solution S' to the given instance of problem (c).

$\Rightarrow$ We argue the contrapositive. If there is no path from s to $v^{\mathsf{c}}_{i^*}$ in N''_f, then there exists a min cut in N'' where $v^{\mathsf{c}}_{i^*}$ belongs to the sink side of the cut. This means that every flow of maximum value in N'' needs to use the edge from s to $v^{\mathsf{c}}_{i^*}$ at full capacity w_{i^*}, and therefore that in every solution to problem (b) where the maximum number of days that instructors work does not exceed W, i^* needs to work at least w_{i^*} days. Thus, there is no solution to the given instance of problem (c). □

Claim 2 shows the correctness of the following algorithm.

Algorithm 3

1: **if** S assigns no days to i^* **then return** "no"
2: construct the network N''
3: construct the integral flow f in N'' corresponding to S
4: compute the residual network N''_f
5: **return** whether there is a path from s to $v^{\mathsf{c}}_{i^*}$ in N''_f

Each step in the above algorithm runs in time linear in the number of vertices and edges of N'', which is $O(mk)$. It follows that the total running time is $O(mk)$.