Online metric bipartite matching

You have a metric space (such as the Euclidean plane), n red points, n blue points, and the goal is to construct a perfect matching between the red points and the blue points (associate exactly one red point to each blue point in a one-to-one fashion) so as to minimize the sum of the lengths of the n edges used, i.e. the distances between the matched pairs.

In the online setting, the red points are known a priori, but the blue points arrive in an online fashion. Every time a blue point arrives you have to decide who to match it to, among the red points that have not yet been matched; your decision is irreversible.

1. Give an example with $n = 2$, (with an ad hoc metric space – for example, a graph with distances equal to the shortest path metric on the graph) proving that no deterministic online algorithm can have competitive ratio less than 3.

   Solution: Take a line graph with points at locations $-1-\epsilon, -1, 0, 1, 1+\epsilon$. Points 1 and -1 are red. The adversary gives 0 as the first blue point. If the algorithm matches it to 1 (at cost 1), then the adversary gives $1+\epsilon$ as the next blue point, forcing the algorithm to match it to -1 (at cost $2+\epsilon$). If the algorithm matches it to -1 (at cost 1), then the adversary gives $-1-\epsilon$ as the next blue point, forcing the algorithm to match it to 1 (at cost $2+\epsilon$). Either way, the algorithm pays $3+\epsilon$ total, when there exists a way to match the two red points to the two blue points at cost $1+\epsilon$. The ratio tends to 3 as ϵ tends to 0.

2. Consider the greedy algorithm: match the newly arrived blue point to the closest red point that is not yet matched. Prove that the greedy algorithm cannot have constant competitive ratio. You may give an example that simply uses the one-dimensional Euclidean space (that is, the line) to place points.

   Solution: Place red points at positions $0, 1, 2, 4, 8, 16, \dots$ on the line. Place the first blue point at position $1/2+\epsilon$, then blue points at positions $1+\epsilon, 2+\epsilon, 4+\epsilon, 8+\epsilon, \dots$. The algorithm matches them to $1, 2, 4, 8, \dots,$, and pays $\theta(2^n)$. Instead, the optimal matching pays $1/2+n\epsilon$, so the ratio is $\theta(2^n)$ as ϵ goes to 0.

3. Prove: no deterministic algorithm can have competitive ratio less than a constant times n.

   Solution: Make a star graph with a point in the center at distance 1 from n red points $r_1, r_2, \dots, r_n$. Put the first blue point in the center. The algorithm matches it to one of the red points, say r_1. Then place a blue point at location r_1. The algorithm matches it to one of the unmatched red points, say r_2. Then place a blue point at location r_2. The algorithm matches it to one of the unmatched red points, say r_3, and so on. After $b_1, \dots, b_n$ have arrived, the algorithm has paid $1+2(n-1)$, whereas the optimal matching has cost 1.

4. Going back to the example from question 1, prove that no randomized algorithm can have competitive ratio better than 1.01.

   Solution: We use Yao's principle and give a distribution over possible inputs such that for every deterministic algorithm, the average cost of the algorithm on a random input is at least 1.01 times the average OPT. We take the same graph as in question 1, but the adversary *sequence is $0, 1+\epsilon$ with probability $1/2$ and $0, -1-\epsilon$ with probability $1/2$. Consider a deterministic algorithm. With probability $1/2$, it will create a matching of cost $1+\epsilon$, and with probability $1/2$, a matching with cost $3+\epsilon$: on average, $2+\epsilon$, whereas the optimum is $1+\epsilon$. Thus no randomized algorithm can have competitive ratio better than 2.*

Assume that your metric space is a hierarchically separated tree: a rooted tree such that the length of an edge from the root (level 0) to one of its children (level 1) is 1, the distance from a child of the root (level 1) to each of *its* children (level 2) is $\delta < 1$, ..., the distance from a node at level i to a node at level $i+1$ is δ^i. Now, on such a metric space we will analyze the randomized greedy algorithm defined as follows: match a blue node to the closest red node, breaking ties at random.

5. Let m_u be the absolute difference between the number of blue nodes and the number of red nodes in the subtree T_u rooted at node u, and $M_1 = \sum_{u:\text{level}(u)=1} m_u$. Prove that $\text{OPT} = \Omega(M_1 + \sum_{u:\text{level}(u)=1} \text{OPT}(S_u^*))$, where S_u^* is the way to choose all but m_u nodes in T_u, half red and half blue, so that the subproblem S_u inside T_u has minimum cost.

   Solution: Consider the OPT solution. If it connects a red node inside T_u to a blue node outside T_u, and a blue node inside T_u to a red node outside T_u, then we can rearrange the connecting paths at no extra cost so that the red and blue nodes inside T_u are connected to one another. So we can assume that in OPT all nodes of T_u connected to the outside are of the same color. Then their number is exactly m_u and their total cost is at least M_1 since each has to cross the edge from u to the root at cost 1. The remainder of the solution is a subsolution inside T_u which by definition of S_u^* has cost at least OPT(S_u^*).

6. Let m'_u be the number of blue nodes in the subtree rooted at u which are matched outside the subtree by the algorithm. Prove that the cost of the algorithm is at most $O(\sum_{u:\text{level}(u)=1}(m'_u/(1-\delta) + \text{Alg}(S'_u))$, where S'_u are the nodes in T_u matched by the algorithm inside T_u.

   Solution: m'_u nodes are matched outside T_u, so their matching edge costs at least 1 (the cost of the edge to get from u to the root) and at most $2/(1-\delta)$ (the cost of any leaf-to-root-to-leaf path in the tree, i.e. the diameter). The rest is just the restriction of the Algorithm to the subtree T_u, the red nodes matched within, and the sequence of blue nodes matched within.

7. Assume that we can prove that $\sum_{u:\text{level}(u)=1} m'_u = O(\sum_{u:\text{level}(u)=1} m_u \log n)$. (This can be proved by a potential function argument). Use this to complete the analysis of the algorithm.

   Solution: by the previous question the cost of the algorithm is at most $O(\sum_{u:level(u)=1}(m'_u/(1-\delta) + Alg(S'_u))$. By induction $Alg(S'_u) \leq (\log n)/(1-\delta) OPT(S'_u)$. This is at most $(\log n)/(1-\delta) OPT(S_u^*)$ since S'_u has at most as many matching pairs as S_u^* and by definition of S_u^*. By the assumption the first term is at most $\sum m_u \log n/(1-\delta)$. Adding and using the lower bound on OPT from question 5 gives the bound of $\log n/(1-\delta)$ approximation.