The Theory of Computation
Bernard M.E. Moret
Addison-Wesley-Longman, 1998

Solution Manual
Chapter 8, Additional Problem 8.2

Since the associated decision problem is (trivially) strongly NP-complete, there can be no fully polynomial-time approximation scheme for the optimization problem. One can also prove that no polynomial-time approximation algorithm for *Decision Tree* can exist unless P = NP.

In order to "multiply" the problem here, we need to do more than merely copy the problem over and over, since the result would not even have an identification tree of any kind. We want more classes, because we want to feed to the claimed approximation algorithm a suitably larger instance. But we need all new classes tied to existing ones somehow, since all will be in the same tree. In particular, we need tests to distinguish not only between new classes in a "copy," but also tests to distinguish between "copies." Assume that the claimed algorithm guarantees a distance K approximation and let k be the smallest power of 2 such that $2^p = k > K$. We can solve an arbitrary instance C, $\mathcal{T}$ (where C is the set of classes and $\mathcal{T}$ the set of tests) exactly by using $\mathcal{A}$ as follows.

We create a new instance C', $\mathcal{T}'$. C' is the disjoint union of $2^p = k$ copies of C; if $C = \{a, b \ldots, z\}$, then $C' = \{a_1, a_2, \ldots, a_k, b_1, \ldots, b_k, \ldots, z_1, \ldots, z_k\}$. $\mathcal{T}'$ is a single copy of $\mathcal{T}$ where each class has been replaced by its k copies, plus p additional tests, call them $T_1, \ldots, T_p$, as follows.

$$T_1 = \{a_1, \ldots, a_{k/2}, b_1, \ldots, b_{k/2}, \ldots, z_1, \ldots, z_{k/2}\}$$

$$T_2 = \{a_1, a_2, ..., a_{k/4}, a_{1+k/2}, a_{2+k/2}, ..., a_{3k/4}, b_1, b_2, ..., b_{k/4}, b_{1+k/2}, b_{2+k/2}, ..., b_{3k/4}, \ldots, z_1, ..., z_{3k/4}\}$$

$$\ldots$$

$$T_p = \{a_1, a_3, a_5, \ldots, a_{k-1}, b_1, b_3, b_5, \ldots, b_{k-1}, \ldots, z_1, z_3, \ldots, z_{k-1}\}$$

Note that these p tests among themselves distinguish any copy from any other copy, but cannot distinguish any two elements in the same copy. We claim that an optimal identification tree for this new instance uses these p tests in the first p levels (all complete), then has $2^p = k$ copies of the optimal identification tree for the original instance hanging from level p; if the total path length of the optimal identification tree for the original instance with n objects was m, the total path length of this new tree is $M = km + knp$. (Equivalently, we can use the optimal identification tree for the original instance and replace each of its leaves by a complete tree of p levels using tests $T_1, \ldots, T_p$ to distinguish between the 2^p copies of each class associated with that leaf. Yet other optimal trees exist, but they all share the property that all p tests, $T_1, \ldots, T_p$, appear on every path from the root.) A simple, but tedious, induction proof verifies that claim. Now we have the desired effect. Applying $\mathcal{A}$ to the new instance produces a tree with total path length not exceeding $M + K$; since every class exists in 2^p copies and since only the p tests T_i can distinguish between copies, all of them must appear on every path, so that we can assume that they make up the top p levels of the tree. (If they do not, we can move them up the tree until they do.) Now, each of the k subtrees hanging from level p is a copy of the original instance and its optimal solution requires a subtree of total path length m. Since the total path length of the tree created by $\mathcal{A}$ does not exceed $M + K = km + knp + K$ and since the top p levels account for the knp term, the k copies have a total cost of $km + K$, with $K < k$; since each copy costs at least m, one copy must cost exactly m and thus has been solved optimally.

Clearly, the algorithm given cannot be optimal: we can run it in time $O(n^2 \cdot \log n)$ easily, so that its optimality would imply $\mathrm{P} = NP$, an unlikely outcome. One can fairly easily construct examples where the heuristic could construct trees with an external path length arbitrarily larger than optimal, in absolute values; it is much harder to construct an example where the ratio of approximate to optimal does not asymptotically reach 1; much harder yet to construct an example where the ratio can grow arbitrarily large. However, the latter is the correct worst-case behavior of the algorithm ($O(\log n / \log \log n)$), as proved in Garey & Graham (*Acta Informatica* **3**, 74, pp. 347–355). It can nevertheless be shown that the algorithm is in the mean asymptotically optimal (see Moret, *ACM Computing Surveys* **14**, 4, pp. 595–623, for details).

Whether a polynomial-time approximation algorithm with a constant ratio guarantee exists remains an open problem.