CS513 HW4

1. **Unit Ball Graph Dominating Set (3D):** Given a set of points in 3D, a unit ball graph is defined by connecting points of distance at most 1 away (a dominating set D is a subset of vertices such that every vertex in the graph is either in D or adjacent to a vertex in D). Design a polynomial time algorithm to approximate the minimum dominating set by a constant factor.

   Solution: Basically the same as 2D geometric set cover.

   (a) **Algorithm:** Pick an arbitrary point u from the uncovered set, add it to our dominating set S, and mark u and all its neighbors (all points within distance 1) as "covered". Repeat until no uncovered points remain.

   (b) **Approximation Analysis:** To show this is a constant approximation, we use a packing argument. For every vertex v that the *optimal solution* (OPT) chooses, consider its neighborhood $N(v)$. The points in $N(v)$ are contained in a ball of radius 1 centered at v. In our greedy solution, since we pick an independent set of points to cover the space, we can choose at most a constant number of independent points (centers of unit balls) that pack into or dominate the neighborhood of v. Thus, $|S| \leq O(1) \cdot |OPT|$.

2. **Load Balancing (Greedy Makespan)**

   We are given m identical machines $M_1, \ldots, M_m$ and a set of n jobs. Each job j has a processing time $t_j > 0$. We need to assign each job to exactly one machine. Once assigned, a machine processes its jobs sequentially. Let L_i be the total load (sum of processing times) assigned to machine M_i. The objective is to minimize the **Makespan**, which is defined as the maximum load among all machines:

   $$\text{Makespan} = \max_{i=1\ldots m} L_i$$

   (Intuitively, we want to finish all jobs as early as possible, so we need to minimize the completion time of the busiest machine.)

   Now, consider the following greedy algorithm for this problem: Process the jobs in an arbitrary order $1, \ldots, n$. For each job j:

   1. Check the current load of every machine $M_1, \ldots, M_m$.
   2. Assign job j to the machine M_i that currently has the **minimum load**.
   3. Update the load of M_i: $L_i \leftarrow L_i + t_j$.

   Questions:

   (a) Show an example with $m = 2$ machines where this Greedy algorithm is not optimal.

   (b) What is the approximation ratio of this algorithm? Prove your answer.

   Solution:

   (a) **Counter-example:** Consider $m = 2$ machines and 3 jobs with processing times $\{1, 1, 2\}$.

- **Greedy Algorithm:**
  - Job 1 ($t_1 = 1$) is assigned to Machine 1. Loads: $M_1 = 1, M_2 = 0$.
  - Job 2 ($t_2 = 1$) is assigned to Machine 2 (since $0 < 1$). Loads: $M_1 = 1, M_2 = 1$.
  - Job 3 ($t_3 = 2$) is assigned to Machine 1 (tie-breaking, pick M_1). Loads: $M_1 = 1 + 2 = 3, M_2 = 1$.
  - **Greedy Makespan** $= \max(3, 1) = 3$.
- **Optimal Assignment (OPT):**
  - Assign Job 3 ($t_3 = 2$) to Machine 1.
  - Assign Job 1 and Job 2 ($t_1 = 1, t_2 = 1$) to Machine 2.
  - Loads: $M_1 = 2, M_2 = 1 + 1 = 2$.
  - **Optimal Makespan** $= 2$.

Thus, Greedy gives 3 while OPT is 2. The ratio is $3/2 = 1.5$.

(b) **Approximation Ratio:** The greedy algorithm is a $(2 - \frac{1}{m})$**-approximation**.

Proof: Let OPT be the optimal makespan. We rely on two lower bounds for OPT:

i. **Average Load:** The optimal makespan must be at least the average load across all machines: $OPT \geq \frac{1}{m} \sum_{i=1}^{n} t_i$.

ii. **Max Job:** The optimal makespan must be at least the length of the longest job: $OPT \geq \max_i t_i$.

Consider the machine M_{busy} that finishes last in the Greedy schedule. Let T_{greedy} be its load (which is the makespan). Let job j (with length t_j) be the *last* job assigned to machine M_{busy}.

When job j was assigned to M_{busy}, M_{busy} must have had the *minimum* load among all m machines (otherwise, the greedy algorithm would have picked a different machine). Let S be the load of M_{busy} just before adding job j. So, $T_{greedy} = S + t_j$.

Since S was the minimum load at that time, all other $m - 1$ machines had a load of at least S. Therefore, the total processing time of all jobs satisfies:

$$\sum_{i=1}^{n} t_i \geq m \cdot S + t_j$$

Rearranging for S:

$$S \leq \frac{1}{m} \left(\sum_{i=1}^{n} t_i - t_j \right) = \frac{1}{m} \sum_{i=1}^{n} t_i - \frac{t_j}{m}$$

Using the lower bound $OPT \geq \frac{1}{m} \sum t_i$, we get:

$$S \leq OPT - \frac{t_j}{m}$$

Now, substitute this back into the expression for the greedy makespan T_{greedy}:

$$T_{greedy} = S + t_j \leq \left(OPT - \frac{t_j}{m} \right) + t_j = OPT + \left(1 - \frac{1}{m} \right) t_j$$

Using the second lower bound $t_j \leq OPT$, we conclude:

$$T_{greedy} \leq OPT + \left(1 - \frac{1}{m} \right) OPT = \left(2 - \frac{1}{m} \right) OPT$$

Thus, the algorithm is a $(2 - \frac{1}{m})$-approximation. For large m, it approaches a factor of 2.

3. **Maximum 3D Matching**

Given disjoint sets X, Y, Z (each of size n) and a set $T \subseteq X \times Y \times Z$ of triplets. A subset $M \subseteq T$ is a *3D matching* if each element of X, Y, Z appears in at most one triplet in M. The goal is to find a 3D matching M of maximum size. Give a polynomial time algorithm to find a solution that is at least $1/3$ of the optimal solution.

Solution:

(a) **Algorithm (Greedy):** Start with $M = \emptyset$. Iterate through the triplets in T in any arbitrary order. For a triplet $t = (x, y, z) \in T$:

- If none of x, y, z are currently covered by any triplet in M, add t to M.
- Otherwise, discard t.

Return M.

(b) **Analysis (Approximation Ratio):** Let M be the solution returned by the greedy algorithm, and M^* be the optimal maximum matching. For every triplet $t^* = (x^*, y^*, z^*) \in M^*$, one of the following must be true:

- $t^* \in M$ (we picked it).
- t^* was not picked because at least one of its elements (x^*, y^*, or z^*) was already covered by some triplet $t \in M$ that we picked earlier.

Crucially, each triplet $t \in M$ consists of 3 elements, so it can "block" or conflict with at most 3 triplets in the optimal solution M^* (one conflicting on x, one on y, and one on z). Since every triplet in M^* is either in M or blocked by a triplet in M, we have:

$$|M^*| \leq 3|M| \implies |M| \geq \frac{1}{3}|M^*|$$

Thus, this is a $\frac{1}{3}$-approximation.

4. **Facility Location (Supermarket Placement)**

Given a set of n customers and a set of potential locations S for supermarkets, decide where to open the supermarkets to minimize a total cost function. The cost consists of two parts:

- **Opening Cost:** A cost f_i if we open a market at location $s_i \in S$.
- **Service Cost:** For each customer j, if they are served by a store at s_i, the cost is d_{ji}. Each customer connects to the closest open store.

Design an $O(\log n)$ approximation algorithm for this problem.

Solution:

This problem can be modeled as a **Metric Uncapacitated Facility Location** problem, which is closely related to the Set Cover problem.

Algorithm: We treat this as a set cover problem where we want to "cover" customers. However, unlike standard set cover, we can cover a subset of customers using a facility with a specific cost.

Define a "star" as a pair (s_i, C') where $s_i \in S$ is a facility and C' is a subset of uncovered customers served by s_i. The cost of this star is $f_i + \sum_{j \in C'} d_{ji}$. The "cost-effectiveness" is $\frac{f_i + \sum_{j \in C'} d_{ji}}{|C'|}$.

Greedy Algorithm:

Initialize all customers as uncovered.

While there are uncovered customers:

- Pick the facility s_i and a subset of remaining customers C' that minimizes the cost-effectiveness ratio (average cost per newly covered customer).
- Open facility s_i, assign customers in C' to it, and mark them as covered.

Analysis: The analysis follows the standard greedy set cover proof. Let n_k be the number of uncovered customers at step k. The optimal solution can cover these n_k customers with total cost OPT. Therefore, there exists some star (subset of the optimal facilities) that has an average cost of at most OPT/n_k. Since our greedy algorithm picks the *best* cost-effectiveness, the cost we pay for covering a new element is at most OPT/n_k. Summing over all elements (harmonic series):

$$Cost_{greedy} \leq \sum_{k=1}^{n} \frac{OPT}{k} = H_n \cdot OPT \approx O(\ln n) \cdot OPT$$

Thus, the algorithm achieves an $O(\log n)$ approximation ratio.