E0 249: Approximation Algorithms, Spring 2022 **April 19**

Week 13

Primal-dual algorithm for Facility Location &
Local search for k-median

Instructors: Arindam Khan, Anand Louis *Scribe: Aditya Lonkar*

1 Introduction

In the previous lecture, we showed how to obtain a deterministic polynomial time 4-approximation and a randomized polynomial time 3-approximation for the FACILITY LOCATION problem using an LP-based algorithm. In this lecture, we will first look at a deterministic primal-dual algorithm for the FACILITY LOCATION problem which yields a 3-approximation. Further, we will look at how to use Local search to get a $(5+\epsilon)$-approximation for the k-MEDIAN problem running in polynomial time as long as ϵ is a constant.

2 Primal-Dual method

This section is based on Section 7.3 and 7.6 in [WS11]. We have already seen the Primal-Dual method in the context of set cover. The main idea in this method is that we do not need to explicitly construct a dual solution. For a minimization problem, initially, the primal is infeasible and we start with a feasible dual with let's say all dual variables set to 0. Then, we increase dual variables in a certain fashion until the dual constraints get tight. We do this until no more dual variable can be increased without violating a constraint. Based on the constraints in the dual which got tight, we correspondingly choose the primal variable to be set to 1 to be included in our solution. We first demonstrate the primal-dual method to get an exact solution to the Shorted s-t path problem and then in the next subsection, we show how to obtain a 3-approximation for the UNCAPACITATED FACILITY LOCATION location problem using the primal-dual method.

2.1 Shortest s-t path

Recall that in the Shortest s-t path problem, we are given a graph $G = (V, E)$, two vertices s and t, and a non-negative weight function for all the edges in E. The goal is to find a minimum weight s-t path in G. There is a well known greedy algorithm called *Dijkstra's algorithm*. We will demonstrate using the primal-dual method an algorithm which runs in polynomial time and behaves in the same way as *Dijkstra's algorithm*.
First, we look at the primal and its dual LP formulations for this problem. Let $C_{s,t}$ denote the set of s-t cuts in G. The LP is as follows:

$$\min \sum_{c \in E} c_e x_e$$

such that

$$\sum_{e \in \delta(S)} x_e \geq 1 \ \ \forall S \in C_{s,t}$$

$$x_e \geq 0 \ \ \forall e \in E$$

The dual has variables y_S corresponding to each $S \in C_{s,t}$. We state the dual now:

$$\max \sum_{S \in C_{s,t}} y_S$$

such that

$$\sum_{S \in C_{s,t}: e \in \delta(S)} y_S \leq c_e \ \ \forall e \in E$$

$$y_s \geq 0 \ \ \forall S \in C_{s,t}$$

Now, we state the primal-dual algorithm.

Algorithm 1: Primal-Dual algorithm for Shortest s-t path:

Initialize $y = 0$, $F = \emptyset$.
while *there is no s-t path in* (V, F) **do**
  Let C be the connected component in (V, F) containing s. Increase y_C until there is an edge $e \in \delta(C)$ such that its corresponding dual constraint is tight.
  Set $F := F \cup \{e\}$
If P be an s-t path in (V, F), output P.

Lemma 2.1. *At any point in the algorithm, F forms a tree containing s.*

Proof. Proof is by induction on the size of the connected component containing s. Base case is true when only s is in the connected component. When we add an edge e to a connected component C of (V, F) containing s, e has one endpoint inside C and one endpoint outside by definition of the algorithm. Hence, it cannot form a cycle. ■

Therefore, algorithm outputs an s-t path and for each edge $e \in P$, $c_e = \sum_{S: e \in \delta(S)} y_S$ due to the definition of our algorithm. Thus, we have that

$$\sum_{e \in P} c_e = \sum_{e \in P} \sum_{S: e \in \delta(S)} y_S = \sum_{S \in C_{s,t}} y_S |P \cap \delta(S)|$$

Lemma 2.2. *For $S \in C_{s,t}$, if $y_S > 0$, then $|P \cap \delta(S)| = 1$.*

Proof. We will prove the statement of the lemma by contradiction. Hence, assume that for $S \in C_{s,t}$, $y_S > 0$ and $|P \cap \delta(S)| > 1$. Then, there must be a subpath P' of P such that only the start and end vertices on P' are in S. But since y_S ended up increasing to be strictly larger than 0, we can infer from Lemma 2.1 that F at that point formed a tree spanning the vertices of S. Thus, $F \cup P'$ forms a cycle as there exist two paths between the end vertices of the endpoints of P'. This is a contradiction since by Lemma 2.1 we know that even when the entire path was constructed by the algorithm, $P \in F$ at that point and F was a tree. ■

The above lemma implies that $\sum_{e \in P} c_e = \sum_{S \in C_{s,t}} y_S \leq \text{OPT}$ by weak duality. Since no s-t path can have length $< \text{OPT}$, the algorithm finds a minimum weight path from s to t, i.e., with weight OPT.

2.2 Uncapacitated Facility location

First, we state the primal LP and the dual LP for Facility Location . For the primal,

$$\min \sum_{i \in \mathcal{F}} f_i y_i + \sum_{i \in \mathcal{F}, j \in \mathcal{C}} d_{ij} x_{ij}$$

such that

$$\sum_{i \in \mathcal{F}} x_{ij} = 1 \ \ \forall j \in \mathcal{C}$$

$$x_{ij} \leq y_i \ \ \ \forall i \in \mathcal{F}, j \in \mathcal{C}$$

$$x_{ij} \geq 0 \quad \forall i \in \mathcal{F}, j \in \mathcal{C}$$
$$y_i \geq 0 \ \ \forall i \in \mathcal{F}$$

For the dual,

$$\max \sum_{j \in \mathcal{C}} v_j$$

such that

$$v_j - w_{ij} \leq d_{ij} \quad \forall i \in \mathcal{F}, j \in \mathcal{C}$$
$$\sum_{j \in \mathcal{C}} w_{ij} \leq f_i \quad \forall i \in \mathcal{F}$$
$$w_{ij} \geq 0 \quad \forall i \in \mathcal{F}, j \in \mathcal{C}$$
$$v_j \geq 0 \ \ \forall j \in \mathcal{C}$$

In such problems, it is often a good idea to interpret what the dual means. Hence, w_{ij} can be viewed as the contribution of client j towards opening facility i. v_j as mentioned in the last lecture is the cost of client j paid towards the total cost of the dual. Now, unlike in the case of shortest s-t path, here we have two types of dual variables we can increase as according to the blueprint of a primal-dual algorithm.

Now, for $i \in \mathcal{F}$, define $N(i) = \{j \in \mathcal{C} : v_j \geq d_{ij}\}$.
Symmetrically, for $j \in \mathcal{C}$, define $N(j) = \{i \in \mathcal{F} : v_j \geq d_{ij}\}$. In our algorithm, we will consider a maximal dual feasible solution (v^*, w^*), in the sense that we cannot increase the variables v_j any further without violating a constraint. From this, we obtain a set of values for the w_{ij} variables. This dual solution has a nice structure as we explain now.
We will say that a client j contributes to a facility i if $w_{ij} > 0$. Now, observe that given a feasible set of values for the dual variables v^*, we can derive w^* by setting $w^*_{ij} = \max\{0, v^*_j - d_{ij}\}$. If we derive the w^* values, in this way, if a client j contributes to facility i, then $w^*_{ij} > 0$ which implies that $j \in N(i)$. Also, if $j \in N(i)$, then $v^*_j = w^*_{ij} + d_{ij}$.
We now define T to be the set of facilities such that for $i \in T$, $\sum_{j \in \mathcal{C}} w_{ij} = f_i$, i.e., their corresponding constraint is tight or equivalently, the entire cost for their opening is paid for by different clients. Then, we make the following claim.

Claim 2.3. *In a maximal dual solution (v^*, w^*), every client should neighbour some facility in T.*

Proof. First we claim that in a maximal dual solution, $v^*_j = \min_{i \in \mathcal{F}}(d_{ij} + w^*_{ij})$ and some facility $i \in \mathcal{F}$ attaining this minimum lies in T. Then, $v^*_j \geq d_{ij}$ which implies j neighbours $i \in T$. To see the claim, if $v^*_j < \min_{i \in \mathcal{F}}(d_{ij} + w^*_{ij})$, we can feasibly increase v^*_j without violating any constraints and hence, the solution would not have been maximal. If $v^*_j = \min_{i \in \mathcal{F}}(d_{ij} + w^*_{ij})$ and the facility i attaining this minimum would not have been in T, then we can again increase v^*_j and w^*_{ij} simultaneously by the same amount until $i \in T$ and again, the solution would not have been maximal. ■

The next obvious intuition is to assign every client in j to some facility in T. That is, if we assign every client j to a facility i to which it contributes, then, for a facility $i \in T$, the cost of assigning all $j \in N(i)$ to i plus the cost of opening i is as follows:

$$f_i + \sum_{j \in N(i)} d_{ij} = \sum_{j \in N(i)} (w^*_{ij} + d_{ij}) = \sum_{j \in N(i)} v^*_j$$

The first equality follows from the fact that since $i \in T$, $f_i = \sum_{j \in N(i)} w^*_{ij}$. The clients which contribute to the cost of opening facility i are exactly the ones which lie in its neighbourhood. Second equality follows from the way we increased our dual variables. But the problem with assigning clients in this way is that a client j might contribute to multiple facilities in T. Hence, the primal-dual algorithm which we present now

makes sure that a client $j \in \mathcal{C}$ is assigned to exactly 1 facility in T, which need not be its neighbour and should not be too far from it.
We now present the algorithm: Generate a maximal dual solution by increasing the dual variables v_j. Let S be the set of clients whose duals we are increasing, and let T be the set of facilities whose dual inequality is tight. Initially $S = \mathcal{C}$ and $T = \emptyset$. We increase v_j uniformly for all $j \in \mathcal{C}$. Once, $v_j = d_{ij}$ for some i, we increase w_{ij} uniformly with v_j. Then, two things can happen w.r.t j: either at some point j becomes a neighbour of some tight facility $i \in T$ or a dual inequality becomes tight for some facility i. In the first case, we remove j from S, and in the second case, we add i to T and remove all neighbouring clients $N(i)$ from S. Once S is empty and very client neighbours some facility in T, we iteratively start selecting some facilities $T' \subseteq T$ to be included in our final solution. The way we do that is we start with some arbitrary facility $i \in T$, add i to T' and delete all facilities $i' \in T$ such that i' has a common neighbour j with i, which contributes to both of them. We do this until T becomes empty, which defines our set T'. Finally assign every client j to the nearest facility in T'.

Algorithm 2: Primal-Dual algorithm for Uncapacitated Facility Location

$v \leftarrow 0$, $w \leftarrow 0$
$S \leftarrow D$
$T \leftarrow \emptyset$
while $S \neq \emptyset$ **do**
  Increase v_j for all $j \in S$ and w_{ij} for all $i \in N(j), j \in S$ uniformly until some $j \in S$ neighbours some $i \in T$ or some $i \notin T$ has a tight dual inequality
  if *some* $j \in S$ *neighbours some* $i \in T$ **then**
    $S \leftarrow S \setminus \{j\}$
  if $i \notin T$ *has a tight dual inequality* **then**
    $T \leftarrow T \cup \{N(i)\}$
$T' \leftarrow \emptyset$
while $T \neq \emptyset$ **do**
  Pick $i \in T : T' \leftarrow T' \cup \{i\}$
  $T \leftarrow T \setminus \{h \in T : \exists j \in D, w_{ij} > 0 \text{ and } w_{hj} > 0\}$

Observe that the clients $j \in \mathcal{C}$ which get assigned in our algorithm to a neighbouring facility contribute very little to the total cost as we saw in the analysis above. We just need to worry about clients which had to be assigned to some facility in T' which was not their neighbour.
We will now prove that for a client j which did not get assigned to any facility in T' in the algorithm, there exists some facility $i \in T'$ such that $d_{ij} \leq 3v_j$. The intuition is that if a client j does not have a neighbour in T', then it must have neighboured some tight facility $h \in T$ such that some other client k contributed both to h and another facility $i \in T'$. We can now apply triangle inequality to get the required bound.

Lemma 2.4. *If a client j does not have a neighbour in T', then there exists a facility in $i \in T'$ such that $d_{ij} \leq 3v_j$.*

Proof. Consider j to be a client which does not neighbour any facility in T'. During the course of the algorithm, we stopped increasing v_j because j neighboured some facility $h \in T$. h cannot be in T' due to our assumption. Then, the facility h must have been removed from T because it neighboured some other client k which neighboured another facility $i \in T'$. We will show now that the cost of assigning j to i is at most $3v_j$. This cost, by triangle inequality, is at most the sum of the three terms $d_{hj} + d_{hk} + d_{ik}$ and we will show that all of these terms individually are no more than v_j.
We know that $d_{hj} \leq v_j$ by the fact that j neighbours h. Consider the point in the algorithm when we stopped increasing v_j. Due to our choice of h, either k already neighboured h or did it at the exact point in time when j started to neighbour h. That is, Because the dual variables are increased uniformly, we have

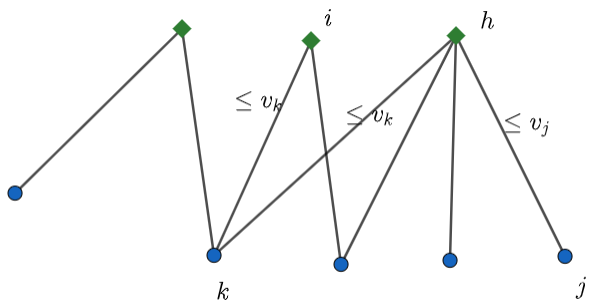

Figure 1: The figure shows client j which does not neighbour any facility in T' in the algorithm. However, it neighbours h and there exists a client k which contributes to another facility $i \in T'$ and h. And $d_{ij} \leq 3v_j$ by triangle inequality.

that $v_k \leq v_j$. Also, since the client k contributes to both i and h, we have that $d_{hk} \leq v_k$ and $d_{hi} \leq v_j$, which finishes the proof. ■

We now conclude by showing a bound on the cost of the solution produced by the primal-dual algorithm. For any client that contributes to a facility in T', we assign it to this facility. Note by the construction of the algorithm, any client contributes to at most one facility in T'. Let $A(i) \subseteq N(i)$ be the neighbouring clients assigned to a facility $i \in T'$. As mentioned before, the cost of opening the facility plus the cost of assigning clients to it is

$$\sum_{i \in T'} \left(f_i + \sum_{j \in A(i)} d_{ij} \right) = \sum_{i \in T'} \sum_{j \in A(i)} (w_{ij} + d_{ij}) = \sum_{i \in T'} \sum_{j \in A(i)} v_j$$

Let Z be the set of clients not neighbouring a facility in T', so that $Z = \mathcal{C} \setminus \bigcup_{i \in T'} A(i)$. By the previous Lemma, we know that there exists some facility $i \in T'$ such that the cost of assigning j to i is at most $3v_j$. Then, the total cost of our solution is

$$\sum_{i \in T'} \sum_{j \in A(i)} v_j + \sum_{j \in Z} 3v_j \leq 3 \sum_{j \in \mathcal{C}} v_j \leq 3\text{OPT}$$

where the last inequality follows from weak duality.

3 Local search

This section is based on Section 9.2 in [WS11]. First, we will start with the basics of Local search and show how to get a 1/2-approximation algorithm for Max-Cut. Then, we will show how to design a local search algorithm for k-Median. The basic idea in local search is to start with an arbitrary solution. Then, try a set of predefined local operations; if one of them works (gives a better value for the objective function), we change our solution to the one defined by this local move. Once it is not possible to execute any such local moves, we stop.

3.1 Max-Cut

In this subsection, we present a local search algorithm for Max-Cut which gives a 1/2-approximate solution. We already know how to get an easy randomized 1/2-approximation algorithm. This algorithm is essentially same in the way it tries out the local moves to the randomized algorithm. Recall the Max-Cut problem:

We are given a graph $G = (V, E)$ and we want to partition of the vertex set into S and $V \setminus S$ such that the number of edges crossing this partition is maximized. First, we start with an arbitrary partition $(S, V \setminus S)$. We then employ the following operations in our local search algorithm:

1. $M_s(u)$: Update $S := S \setminus \{u\}$.
2. $M_{V \setminus S}(u)$: Update $S := S \cup \{u\}$.

As mentioned before, we execute the above two moves until no move leads to a better solution than the local solution at that point of time. Thus,

1. For any $u \in S$, $cost(S) - cost(S \setminus \{u\}) \geq 0$.
2. For any $u \in V \setminus S$, $cost(S) - cost(S \cup \{u\}) \geq 0$.

Here, $cost(S)$ refers to the Max-Cut value when $(S, V \setminus S)$ is the partition of the vertex set V.
Now, we prove that such a locally optimum solution w.r.t the local moves mentioned above is a 1/2-approximate solution. In fact, we will prove something stronger that the cost of this solution is at least $m/2$, where $m = |E|$. The idea in the analysis is to use all of the above inequalities which arise from the fact that the solution is locally optimum, for each vertex $u \in V$.
The above inequality for any vertex $u \in S$ boils down to:

$$cost(S) - cost(S \setminus \{u\}) = |N(u) \cap (V \setminus S)| - |N(u) \cap S| \geq 0$$

where $N(u)$ refers to the neighbourhood of vertex u in $V(G)$. Therefore,

$$2|N(u) \cap (V \setminus S)| \geq |N(u) \cap (V \setminus S)| + |N(u) \cap S| = |N(u)|$$

Similarly, for any $u \in V \setminus S$,

$$2|N(u) \cap S| \geq |N(u) \cap (V \setminus S)| + |N(u) \cap S| = |N(u)|$$

Now, we add all of the inequalities, i.e., for each vertex $u \in V$.

$$\sum_{u \in S} |N(u) \cap (V \setminus S)| + \sum_{u \in V \setminus S} |N(u) \cap S| \geq \frac{1}{2}(\sum_{u \in S} |N(u)| + \sum_{u \in V \setminus S} |N(u)|) = \frac{1}{2} \cdot 2m = m$$

The terms on the LHS of the first inequality above count each edge $\{u, v\}$ crossing the cut twice. This is because assume w.l.o.g that $u \in S$ and $v \in V \setminus S$. Then, the $\{u, v\}$ adds to the count by 1 in the first term of the LHS and adds to the count by 1 again in the second term of the LHS since $v \in V \setminus S$. Hence, we get that the Max-Cut value is at least $m/2$, which implies a $\frac{1}{2}$-approximation. As for the running time, each local search move leads to an increase in the Max-Cut value by at least 1 for it to be a feasible move. Since, the Max-Cut value can be at most m, the algorithm runs in polynomial time.

3.2 *k*-median

In this subsection, we demonstrate a $(5 + \epsilon)$-approximation for the k-MEDIAN problem using local search which runs in polynomial time as long as $\epsilon > 0$ is a constant. k-MEDIAN can be thought of as a variant of FACILITY LOCATION. In k-MEDIAN , we can open at most k facilities so as to minimize the service cost. That is, we are given a set of facilities $\mathcal{F}$ as before and a set of client $\mathcal{C}$. The cost of opening a facility $f_i = 0$, but we can open at most k facilities so as to minimize the service cost. Also given is a set of distances. Facility i and client j have a distance d_{ij} between them. These distances form a metric.
To describe the local search algorithm, we start with an arbitrary set $S \subseteq \mathcal{F}$ of at most k facilities. We will use $c(S)$ to denote the cost of S. That is, the sum of distances over all clients between a client and assigned facility pair when the facilities chosen form the set S. We check if a feasible swap of size 1 exists. That is,

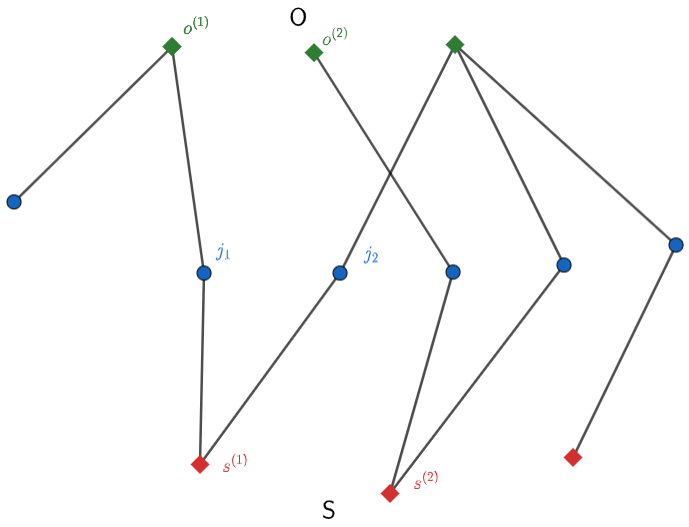

Figure 2: The figure shows the swap operation between $s^{(1)}$ and $o^{(1)}$.

if there exists a facility $f_1 \in S$ and another facility $f_2 \in \mathcal{F} \setminus S$ such that $c((S \setminus \{f_1\}) \cup \{f_2\}) - c(S) < 0$, then we discard f_1 from our solution and add f_2 to it. This is called a feasible swap. Finally, if no swap is feasible, we simply output S.

Let S denote a locally optimal solution and let O denote an optimal solution. For $j \in \mathcal{C}$, let c_j denote its service cost and let o_j denote the service cost of j in O. Assume w.l.o.g that S has k facilities and O has k facilities. This is because one can trivially add more facilities to a solution without incurring any extra cost. Let the facilities in S be $s^{(1)}, s^{(2)}, ..., s^{(k)}$ and the facilities in O be $o^{(1)}, o^{(2)}, ..., o^{(k)}$.

Since the solution is locally optimal, we know that $c((S \setminus s^{(1)}) \cup \{o^{(1)}\}) \geq 0$. Here, let $s^{(1)}$ be the nearest facility in S to $s^{(1)}$. We want to somehow bound the increase in service cost by discarding $s^{(1)}$ and including $o^{(1)}$ in our solution. Hence, instead of considering the optimal assignment of clients after swapping $s^{(1)}$ and $o^{(1)}$, we will choose a slightly different assignment to get a more amenable expression in terms of the bound on the change in cost which will anyway be an upper bound on the optimal assignment of clients to the facilities. To that end, we now consider an assignment of clients to facilities after swapping $s^{(1)}$ and $o^{(1)}$. Then, all the clients not in the neighbourhood of $s^{(1)}$ remained assigned as previously. For the clients $j \in N(s^{(1)}) \cap N(o^{(1)})$, we assign them to $o^{(1)}$. For the clients $j \in N(s^{(1)}) \setminus N(o^{(1)})$, let's say $o^{(2)}$ is the facility assigned to j in O. Now assume that a facility $s^{(2)}$ exists which is closest to $o^{(2)}$ in S such that $s^{(2)} \neq s^{(1)}$. Then, in this case, we assign j to $s^{(2)}$ after swapping.

So, after swapping $s^{(1)}$ with $o^{(1)}$, in the first case when a client $j_1 \in N(s^{(1)}) \cap N(o^{(1)})$, the change in cost is $-c_{j_1} + o_{j_1}$. For the second case, when a client $j_2 \in N(s^{(1)}) \setminus N(o^{(1)})$, the change in cost is

$$\text{change in cost } \leq -c_{j_2} + d(j_2, o^{(2)}) + d(o^{(2)}, s^{(2)})$$

The above expression follows from triangle inequality. Since, we know that $s^{(2)}$ is the closest to $o^{(2)}$ in S,

$$\text{change in cost } \leq -c_{j_2} + d(j_2, o^{(2)}) + d(o^{(2)}, s^{(1)})$$

$$\text{change in cost } \leq -c_{j_2} + o_{j_2} + o_{j_2} + c_{j_2} = 2o_{j_2}$$

But, note that here we crucially use that $s^{(2)} \neq s^{(1)}$, otherwise the analysis fails since we are swapping $s^{(1)}$ out of the solution to begin with. Hence, this leads to the insight for choosing our swaps carefully.

For a locally optimal solution, the analysis will only consider a specific set of swaps which will help us bound the change in cost. Hence, without even considering a subset of the possible swaps, we will be able to show a $(5 + \epsilon)$-approximation for a locally optimal solution.

3.2.1 Choosing the swaps

We consider a function η which maps each facility $i \in O$ to its nearest facility in S. Hence, for a facility $i \in O$, let $\eta(i)$ denote the closest facility to it in S. Define $R_0, R_1, R_{\geq 2} \subseteq S$ to be the set of those facilities such that the number of facilities mapped by η to them is $0, 1$ and 2, respectively. We now construct the set of swaps $\mathcal{S}$:

1. For each $i \in R_1$, add $(i, \eta^{-1}(i))$ to $\mathcal{S}$. Note that η^-1i is well defined here for such an i. Let $O_1 \subseteq O$ be the set of facilities matched in this step.
2. We prove in a claim below that $|O \setminus O_1| \leq 2R_0$. Out of $R_0 \times (O \setminus O_1)$, we then choose a set of swaps such that each facility $i \in R_0$ appears at most twice. Add all these swaps to $\mathcal{S}$.
3. We do not consider any facility in $R_{\geq 2}$ in any swap in $\mathcal{S}$.

Claim 3.1.

$$|O \setminus O_1| \leq 2R_0$$

Proof. First, note that $|S \setminus R_1| = |O \setminus O_1|$ by definition of η. Since, the facilities in $R_{\geq 2} \subseteq (S \setminus R_1)$ are matched to at least 2 facilities in $O \setminus O_1$, $|R_{\geq 2}| \leq \frac{|O \setminus O_1|}{2}$. Also, $|R_{\geq 2}| + |R_0| = |S \setminus R_1|$. Hence, $|R_0| \geq \frac{|O \setminus O_1|}{2}$, which implies the claim. ■

For the swaps in $\mathcal{S}$, our assignment is going to be slightly different than what we showed before. For a swap $(s, o) \in \mathcal{S}$, for each $j \in N(o)$, assign client j to o. And for $j \in N(s) \setminus N(o)$, assign j to $\eta(f_j^0)$, where f_j^0 is the facility serving j in O. We have the following lemma to bound the change in the cost for swaps in $\mathcal{S}$.

Lemma 3.2. *For each swap $(s, o) \in \mathcal{S}$, we have that*

$$0 \leq c((S \setminus \{s\}) \cup \{o\}) \leq \sum_{j \in N(o)} + \sum_{j \in N(s)} 2o_j$$

Proof. Recall in the analysis that we showed before when we swapped $s^{(1)}$ with $o^{(1)}$, the problem arose when $\eta(f_j^0)$ was $s^{(1)}$ again for some facility j assigned to $s^{(1)}$ before the swap. But, the swaps in $\mathcal{S}$ ensure that this does not happen. In our new assignment mentioned in the paragraph above, we can claim that $\eta(f_j^0) \neq s$ for the swap (s, o). This is because no swaps were included in $\mathcal{S}$ involving any facility in $R_{\geq 2}$. As mentioned before, if $j \in N(o)$, the change in cost is $-c_j + o_j$. If $j \in N(s) \setminus N(o)$, then the change in cost C' is

$$C' \leq -c_j + d(j, f^0(j)) + d(j, \eta(f^0(j)))$$
$$\implies C' \leq -c_j + o_j + (c_j + o_j) = 2o_j$$

■

Summing over all swaps in $\mathcal{S}$, we get that

$$0 \leq \sum_{(s,o) \in \mathcal{S}} \left(\sum_{j \in N(o)} (c_j - o_j) + \sum_{j \in N(s)} 2o_j \right)$$

$$0 \leq \sum_{(o \in O)} \left(\sum_{j \in N(o)} c_j \right) - \sum_{(o \in O)} \left(\sum_{j \in N(o)} o_j \right) + \sum_{(s,o) \in \mathcal{S}} \sum_{j \in N(s)} 2o_j$$

This follows from the fact that every $o \in O$ appears in exactly one swap in $\mathcal{S}$. Now using the fact that a client j is uniquely mapped to a facility in O,

$$0 \leq c(O) - c(S) + 2 \cdot 2 \sum_{s \in S} \left(\sum_{j \in N(S)} o_j \right)$$

The above inequality follows from the fact that a facility in S appears in at most two swaps in $\mathcal{S}$. Further, we now use that clients are uniquely mapped to facilities in S which gives

$$0 \leq c(O) - c(S) + 4c(O)$$

$$\implies c(S) \leq 5c(O)$$

Therefore, it is a 5-approximation algorithm.

To analyze the running time, checking if a swap exists can be done in polynomial time since the swaps are of size 1. We have to just check $O(nk)$ number of possible swaps and further assign the clients to the nearest facility. But, local search algorithms need not always terminate. In the presented algorithm, it is unclear if the algorithm will terminate in polynomial time since there can be $\Theta(n^k)$ values for the local search solution. Hence, for an arbitrarily large k the algorithm may need super polynomial time. But we can slightly compromise on the approximation factor by only allowing swaps which decrease the local search solution's value by a factor of at least $(1 - \epsilon)$ for sufficiently small $\epsilon > 0$.

By this approach, in the above equation the LHS will be $-\epsilon \cdot c(S)$ and the RHS remains the same. Hence, the inequality turns out to be $c(S) \leq \frac{5c(O)}{(1-\epsilon)} \leq 5(1 + \epsilon)$. Thus, choosing ϵ to be sufficiently small we can get an algorithm with approximation ratio arbitrarily close to 5. Since now feasible swaps in the local search algorithm necessitate that the cost of the solution decreases by a factor of $(1 - \epsilon)$, the number of swaps will be bounded by $\frac{\log C}{\log(1+\epsilon)}$, where C is the maximum cost of any feasible solution to the instance. Using the Taylor series expansion of $\log(1 + x)$, it is easy to see that $\log(1 + \epsilon) = \Theta(\epsilon)$ for small $\epsilon > 0$. Since $\log C$ can be bounded by the input size, the running time of this algorithm is polynomial in the input and $(1/\epsilon)$.

The swap operation of size 1 can be generalized to swaps of size at most t, by which one can achieve an approximation of $(3 + \frac{2}{t})$, but the running time of this local search algorithm is $n^{O(t)}$. Hence, if we choose $t = 2/\epsilon$, we can achieve an approximation of $(3 + \epsilon)$ and the running time of such an algorithm would be $n^{O(1/\epsilon)}$.

The current best known algorithm (in terms of approximation) for k-MEDIAN is due to Byrka et al. [Byr+14] which yields a 2.675 approximation. Note that this algorithm is not a local search based algorithm which uses a technique called *dependent rounding* and its running time is $n^{O((1/\epsilon)\log(1/\epsilon))}$.

References

[WS11] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[Byr+14] Jarosław Byrka et al. "An improved approximation for k-median, and positive correlation in budgeted optimization". In: *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*. SIAM. 2014, pp. 737–756.