Lecture 12: A new set of techniques
(Facility Location Problems)

So far we've seen:
- Basic relax and round (LP, SDP)
- Greedy algo.
- Region growing (Leighton Las) & Embedding
- Spectral Techniques

Now let us consider a new collection of ideas:

- "Primal Dual"
- "Local Search"
- And some more, especially beyond basic random round and greedy


For such problems, basic setup:
- Metric space \((V, d)\), usually finite \(|V| = n\).
- A client set \(C \subseteq V\).
- Facility costs \(f_i \in C\).

FacLoc: want to open some set \(F \subseteq V\)

\[
\text{st. } \min \sum_{c \in C} d(c, F) + \sum_{i \in F} f_i
\]

K median: open \(|F| = k\) that

\[
\text{minimize } \sum_{c \in C} d(c, F)
\]

- Problems seem related (and they are!), more on that later
- Also, if \(d(c, F) \) replaced by \([d(c, F)]^2\), then \(k\)-means (among others)
Today, we focus on k-median and local search.

Also: start with any "rearrangeable" solution.

While there is a move that improves the solution, if \( \text{cost(new)} < \text{cost(old)} \), take it (or take the best such move).

What are moves? Given a solution \( FSV \):

\[
\begin{align*}
\text{\texttt{Swap}}(u,v) & \quad u \in F, \ v \in F \\
\text{move to solution } F' = F - u + v, \text{ if it has lower cost.}
\end{align*}
\]

We can also consider p-swaps:

\[
\begin{align*}
u \in F, \ v \notin F \quad & \quad |U| = |V| = p, \text{ then move to } (F \cup u) \cup v
\end{align*}
\]

Care about values of \( p = O(1) \).

**Thm:**

Any local optimum with 1 swaps has cost at most \( 3 \cdot \text{OPT} \).

2 p-swaps has cost at most \( (3 + 2p) \cdot \text{OPT} \).

**Thm:**

\( \exists \) 20 instances where the locality gap is at least \( 3 - \varepsilon \). Not any p-swaps.

i.e., local optimum whose cost \( \geq (\text{3 OPT} + \varepsilon) \).
This theorem does not say anything about convergence.

However, by starting at a "reasonable" solution and performing the best swap (or at least, any swap that makes "large" progress),
can give $3^{1+\epsilon}$ approximation in $O(k^2)$ time. [Thm]

Will see this given time (or will see an exercise).

Let's first see how to get the $5apx$ and the lower bound $\geq 3$.

**Lower bound.**

<table>
<thead>
<tr>
<th>Alg</th>
<th>OPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Distances given by shortest path distances.

(Imagine a complete bip. graph, on $OPT \times Alg$, and put a client at distance $x$ from $Alg$, 1 from $OPT$.)

Sps close Alg, open OPT.

type a: old cost $x$ new cost 1

type b: $x$ new cost 1

type c: $x$ $x+2$.

so $k$ clients give improvement of $(x-1)$

$(k-1)$ loss of $(x+2) - x = 2$

$\Rightarrow$ total improvement in $k(x-1) - (k-1) 2$

$= k(x-3) + 2$

$\Rightarrow$ if $x = 3 - \frac{2}{k}$ this is a local optimum. [Thm]

$\forall \epsilon, \exists k$ large enough s.t. local opt w/ $1$-swaps has cost $\geq 3 - \epsilon$. (similarly can set \text{not $p$-swaps also opt $\geq 3 - \epsilon$})
Now for the algorithm's performance

Suppose Alg is a local OPT, what is the cost?

All swaps are non-improving.

So show a set of swaps, that, because of non-improvement, give bounds on the cost of Alg vs OPT.

"Simpler" bijective case

For each facade be OPT, let \( r(b) \) be closest Alg facility to \( b \).

\[
\begin{array}{c}
O & \rightarrow & A \\
\downarrow & & \downarrow \\
O & \rightarrow & A \\
\downarrow & & \downarrow \\
O & \rightarrow & A \\
\end{array}
\]

Bijective case where this map \( r : \text{OPT} \rightarrow \text{Alg} \) is a bijection.

Consider the swaps where \( a \) we open some \( b \) be OPT and close \( r(b) = a \in \text{Alg} \).

What does non-improvement tell us?

Where do these clients go? To \( b \)? To someone else?

[Diagram with nodes and arrows indicating changes in connections]
All clients assigned to b in OPT, let's assign them to b.

Notation: \( C^*(b) \): assigned to b in OPT's soln.

\( C(a) \): clients assigned to a \( \in \) AllG's soln

\( O_j \): cost of client j in OPT

\( A_j \): \( \in \) AllG

\( \Delta \text{Cost} = (O_j - A_j) \)

\( \Rightarrow \) All clients in \( C^*(b) \), assign them to b.

All clients in \( C(a) \setminus C^*(b) \) where to assign them?

\\( \text{Sps. } j \in \text{ and } j \in \text{ } C^*(b') \).

So assign to \( a' \) (which is open, by bijective)

\( = \pi(b') \neq \pi(b) \)

\( \Delta \text{Cost} = d(j, a') - A_j \)

\( \leq d(j', b') + d(b', \pi(b')) - A_j \)

\( \leq d(j, b') + d(b', a) - A_j \) \( \because \pi(b') \) is closest to \( b' \) \( \in \) AllG

\( \leq d(j, b') + d(b', j) + d(j, a) - A_j \) \( \Delta \)-ineq.

\( = 2O_j + A_j' - A_j \)

\( = 2O_j \)

\( \Rightarrow \Delta \text{Cost} = \sum_j (O_j - A_j) + \sum_{j \notin C^*(b)} 2O_j \)

\( 0 \leq \sum_j (O_j - A_j) + \sum_{j \notin C^*(b)} 2O_j \)

\( = \sum_j (O_j - A_j) + \sum_{j \in C(a) \setminus C^*(b)} 2O_j \)

And this must be non-negative.
Now sum over all \((a,b)\) swaps: (and use that \(\pi(e) \in C((b))\) for some \(b\) and \(e \in C((r(b)))\) for some \(b\))

\[
\sum_i (a_j - A_j) + \sum_j 20j > 0.
\]

\[\Rightarrow 30PT - A[Q] > 0.\]

But used bijective property. How to handle case where \(\pi(b) = \pi(c)\) for some \(b \neq b'\)

- Draw a graph on \(OPT \times A[Q] \) of pairs \((b, b')\)
  - has \(k\) edges.
  - every node on left has degree 1. (good)
  - so same nodes on right have degree 1.
  - but # of nodes on right with degree 0 (empty)
  \[\Rightarrow \#\text{ nodes on right with degree } \geq 2. \text{ (dangerous)}\]

So, in this example,

- pair each \(b\) to \(\pi(b)\) if \(\pi(b)\) good
- pair all other \(b\) to some empty ofs facility
- each empty used in 2 pairs
For each swap above, open b, close a

\( \forall j \in C(b) \), change in \( O_j - A_j \)

\( \forall j \in C(a) \), assign to \( b \) which is never

say \( j \in C(b) \) \( O_j(b) \) which by our construction

is never a.

and so is open.

Again same argument says change in \( (20j + A_j) - A_j \leq 20j \)

\[
\sum \left( \sum_{j \in C(b)} (O_j - A_j) + \sum_{j \in C(a) \cup C(b)} 20j \right) \geq 0.
\]

\( \Rightarrow \sum_{j \in C(b)} (O_j - A_j) + \sum_{j \in C(b)} 20j \geq 0. \)

but each b opened once exactly

each a closed \( \leq \) twice

\( \Rightarrow \sum_{j \in C} (O_j - A_j) + 2 \sum_{j \in C} 20j \geq 0. \)

\( \Rightarrow \text{Alg} = \sum_{j \in C} A_j \leq 5 \sum_{j \in C} O_j = 5\text{opt} \)
How to get poly time algorithm: 2 pieces

1) make big steps when possible
2) start at reasonable sol'n.

Same argument says:

Spr. Alg \geq (5+\varepsilon)OPT

\Rightarrow at least one of the k swaps we show have

\text{Improvement} \geq \frac{\varepsilon}{k} \text{OPT}.

If we take best swap:

\Rightarrow improve by at least \left(\frac{\text{Alg} - 5\text{OPT}}{k}\right)

\Rightarrow \text{Alg(new)} \leq \text{Alg(old)} - \left(\frac{\text{Alg} - 5\text{OPT}}{k}\right)

Hence decrease "distance to 5OPT" by a constant factor in \(k\) steps.

and hence get

\text{Alg after } k\text{ steps} \leq 5\text{OPT} + \varepsilon(\text{Alg initial} - 5\text{OPT})(1 - \frac{1}{k})^k

\leq 5\text{OPT} + \text{Alg initial} \left(1 - \frac{1}{k}\right)^k

\text{Reasonable solution:}

Pick some solution with cost \(\text{Alg initial} \leq n \cdot \text{OPT}\)

then get \((5+\varepsilon)\text{OPT}\) after \(k\)\(=\frac{1}{\varepsilon}\) steps at most.

\text{How?} \begin{cases} \text{Pick any } \text{facility } f_0 \text{ to start: } F_1 = \{f_0\} \text{?} \\ \text{for } n = 2 \ldots k \\ \text{choose } f_i \text{ at largest distance from } F_{i-1} \end{cases}

Exercise: \(n\) approximation. Other, better starting solutions exist. (Exercise)
Recap:

**Local search**

- Simple idea
- Define set of moves such that
  (a) can find improving move in polytime
  (b) local optima not this set of moves is good.

Widely used in practice

- Often no guarantee of performance of local optimia
- Often convergence time may be unbounded /exponential
  (even to set to near-optimum solns).

In theoretical analyses

- Show some subset of moves that help compare Alg to Opt.
- Several nice applications
  - Even for exponentially large swap sets. (capacitated fac.loc)
    (labeling problems in vision)
  - Fast algo for bounded-degree spanning tree, kmedian w/o outliers.
- Clean analyses for clean algorithms.