Applied Algorithm Design
Lecture 6

Pietro Michiardi

Institut Eurécom

Local Search Algorithms

Introduction

- Local search (LS) is a very general technique
- It describes an algorithm that “explores” the solution space in a sequential fashion
- The algorithm **moves** in each step from a current solution to a “**nearby**” one
- The main advantage of such technique is that it is very easy to design and implement; furthermore, LS addresses almost any kind of computationally hard problems

Introduction

- The main problem that derives from this flexibility is that it is often very difficult to say anything precise or provable about the quality of the solutions that a LS algorithm finds
- As a consequence, it is very difficult to make a distinction between a good LS heuristic and a bad one

- Today we will discuss a number of LS algorithms that find good, but not necessarily optimal solutions
- It is useful to proceed by relating LS to **energy minimization principles** in physics

Introduction

- Note that there are cases in which it is possible to prove properties of LS algorithms, and to bound their performance relative to an optimal solution
- In fact, we will focus on a set of problems, Facility Location problems, in which it is possible to design LS algorithms that appear trivial
- Hence, a major effort will go into:
  - Proving that these algorithms achieve a solution close to the optimal one
  - Finding a relation between **approximation** and **LS** algorithm

The Landscape of an Optimization Problem

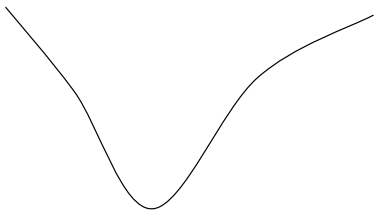
- Much of the core of local search was developed by people thinking in terms of analogies with physics
- Looking at a range of hard computational problem, they reasoned as follows
  - Physical systems are performing minimization all the time
  - They do so when they try to minimize their **potential energy**
  - What can we learn from the ways nature performs minimization?

Potential energy

Example

We now focus on a simple example, the movement of a sphere (ball)

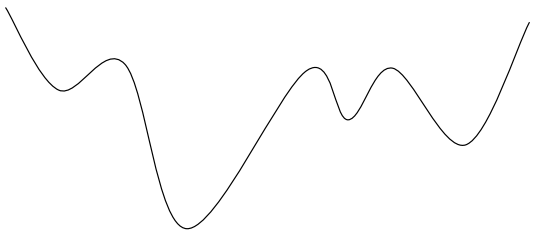
- Why do balls roll downhill?
- From Newtonian mechanics, the ball is trying to minimize its potential energy
- If a ball has a mass m and falls a distance of h, it looses an amount of potential energy proportional to mh



A funnel

Potential energy

- If we make the landscape a bit more complicated, some extra issues creep in



A jagged funnel

- There is a lower point in the figure, which is more desirable to place the ball to rest
- If we start rolling the ball from the wrong side, it will not be able to get over the barrier and settle in the lowest point

Potential energy

- We say that the ball is trapped in a **local minimum**
- A local minimum looks like the lowest point from neighbor locations
- However, if we look at the whole landscape we know that there is a **global minimum**

The Connection to Optimization

- The physical system can be in one of a large number of possible states
- Its energy is a function of its current state
- From a given state, a small **perturbation** leads to a "**neighboring**" state
- The way that these neighboring states are linked together, along with the structure of the energy function on them, defines the underlying energy landscape

The Connection to Optimization

Relation to optimization problems

- In an optimization problem, we have a large (typically exponential-size) set $\mathcal{C}$ of possible solutions
- We also have a **cost function** $c(\cdot)$ that measures the quality of each solution
- For a solution $S \in \mathcal{C}$ we write its cost as $c(S)$

The goal:

Find a solution $S^* \in \mathcal{C}$ for which $c(S^*)$ is as small as possible

The Connection to Optimization

Neighborhoods

We now move on and define precisely what do we mean by neighboring solutions

- We capture the idea that S' can be obtained by a small modification of another solution S
- We write $S' \sim S$ to denote that S' is a neighboring solution of S
- We use $N(S)$ to denote the **neighborhood** of S

$$N(S) = \{S' : S \sim S'\}$$

The Connection to Optimization

- The set $\mathcal{C}$ of possible solutions and the cost function $c(\cdot)$ are provided by the specification of the problem
- Instead, we can make any neighbor we want

Local Search Algorithm

A Local Search algorithm takes this setup, including a neighbor relation, and works according to the following scheme:

- At all times it maintains a **current solution** $S \in \mathcal{C}$
- In a given step, it chooses a neighbor S' of S, declares S' to be the new current solution and iterates
- Through the execution of the algorithm, it remembers the minimum-cost solution that it has seen so far

The Connection to Optimization

- The crux of a LS algorithm is in the choice of the neighbor relation, and in the design of the rule for choosing a neighboring solution at each step

- One can think of a neighbor relation as defining a graph on the set of **all possible solutions**
- Edges join neighboring pairs of solutions
- A LS algorithm can be viewed as **performing a walk** on this graph, trying to move toward a good solution

The Vertex Cover Problem

The Vertex Cover Problem

- You are given a graph $G = (V, E)$
- A set $S \subseteq V$ is a **vertex cover** if each edge $e \in E$ has at least one end in S

The Weighted Vertex Cover Problem

- You are given a graph $G = (V, E)$
- Each $v \in V$ has a **weight** $w_i \geq 0$
- The weight of a set S of vertices is denoted $w(S) = \sum_{i \in S} w_i$
- A set $S \subseteq V$ is a **vertex cover** if each edge $e \in E$ has at least one end in S
- Find a vertex cover S of minimum weight $w(S)$

The Vertex Cover Problem

- So, we are given a graph G
- The set $\mathcal{C}$ of possible solutions consists of all subsets S of V that form vertex covers
  - We always have that $V \in \mathcal{C}$
- The **cost** $c(S)$ of a vertex cover will simply be its size, if no weights are given, or its weight

With such a model, minimizing the cost of a vertex cover is the same as finding one of minimum size

The Vertex Cover Problem

- We focus on examples on local search algorithms that use a particularly simple neighbor relation
- We say that $S \sim S'$ if S' can be obtained from S by adding or deleting a single node
- Thus, our local search algorithm will be **walking** through the space of possible vertex covers, adding or deleting a node to their current solution in each step, and trying to find as small a vertex cover as possible

The Vertex Cover Problem

Proposition

Each vertex cover S has at most n neighboring solutions

Proof.

- Each neighboring solution of S is obtained by adding or dropping a distinct node
- There are a maximum of n nodes in the graph G

□

- A consequence of the previous proposition is that we can efficiently examine all possible neighboring solutions of S

Gradient Descent Method

Gradient descent

- Start with $S = V$
- If there is a neighbor S' that is a vertex cover and has lower cardinality, replace S with S'.

Proposition

The above algorithm terminates after at most n steps

Proof.

Each update decreases the size of the cover by one □

Gradient Descent Method

- Informally, gradient descent moves **strictly** "downhill" as long as it can
- Once this is no longer possible, it stops

Note

- The gradient descent algorithm terminates precisely at solutions that are **local minima**
- That is, solutions S such that, for all neighboring S', we have that:

$$c(S) \leq c(S')$$

Gradient Descent Method

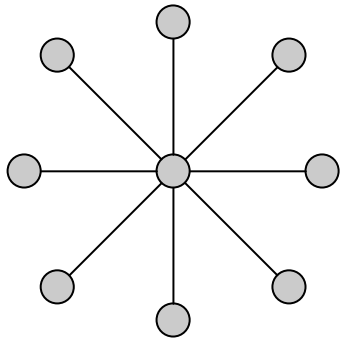
Example 1

- Think about an **empty graph**, that is a graph with no edges
- The empty set is the optimal solution (there are no edges to cover)
- Gradient descent does very well! It starts with the full vertex set V and keeps deleting nodes until there are none left

$\rightarrow$ There is a unique local minimum which is also the unique global minimum

Gradient Descent Method

Example 2



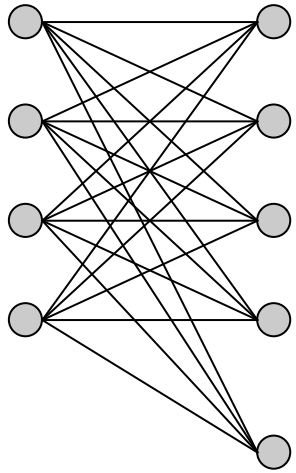
optimum = center node only
local optimum = all other nodes

Gradient Descent Method

- The minimum vertex cover for G is the singleton composed by the center node
- Gradient descent can reach this solution by deleting successively all other nodes in any order
- But if the algorithm starts by deleting the center first, then it is immediately stuck!
- It reaches a local minimum

Gradient Descent Method

Example 3



optimum = all nodes on left side
local optimum = all nodes on right side

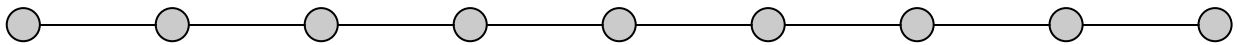
Gradient Descent Method

- If we consider the bipartite graph of this example (without the additional vertex on the right set), we notice that there are two local minima
  - One is the set of nodes to the left
  - The other is the set of nodes to the right
- Which of these local minima is discovered by the gradient descent algorithm depends on the first node that is deleted, whether it is on the left or the right

- Instead, in the example before with all the vertexes, we have that the local minimum is the right set, the global minimum is the left set

Gradient Descent Method

Example 4



optimum = even nodes
local optimum = omit every third node

Gradient Descent Method

- In the path example, there is a unique cover: all even nodes
- However, there are many local minima! E.g. a set of nodes where every third node of the path is deleted
- This is a vertex cover that is significantly larger than the optimum one, but there is nothing you can do about it, once you get stuck in such a local minima

The Metropolis Algorithm

- An idea for an improved LS algorithm considers simulating the behavior of a physical system according to principles of **statistical mechanics**
- The idea is to design an algorithm that is globally biased toward "downhill" steps, but that occasionally makes "uphill" steps to break out of local minima

The Metropolis Algorithm

Gibbs-Boltzmann function

- The probability of finding a physical system in a state with energy E is proportional to $e^{-E/(kT)}$, where $T > 0$ is temperature and k is a constant
- For any temperature $T > 0$, the function is a monotone decreasing function of energy E
- Indeed, systems are more likely to be in a lower energy state than higher one
  - T large: high and low energy states have roughly same probability
  - T small: low energy states are much more probable

The Metropolis Algorithm

Metropolis algorithm

- Given a **fixed** temperature T, maintain current state S
- Randomly perturb current state S to new state $S' \in N(S)$
- If $E(S') \leq E(S)$, update current state to S'
- Otherwise, update current state to S' with probability $e^{-\Delta E/(kT)}$, where $\Delta E = E(S') - E(S) > 0$

The Metropolis Algorithm

Theorem

- Let $f_S(t)$ be fraction of first t steps in which the algorithm is in state S
- Then, assuming some technical conditions, with probability 1:

$$\lim_{t\to\infty} f_S(t) = \frac{1}{Z} e^{\frac{E(S)}{kT}}$$

where

$$Z = \sum_{S\in N(S)} e^{\frac{E(S)}{kT}}$$

Intuition.

The algorithm spends roughly the right amount of time in each state, according to Gibbs-Boltzmann equation □

The Metropolis Algorithm

How does the Metropolis Algorithm behave with the previous examples of Vertex Cover problems?

The Metropolis Algorithm

- We can think of T as a one-dimensional knob that we're able to turn
- It controls the extent to which the algorithm is willing to accept uphill moves
  - T large: probability to accept uphill moves goes to 1 $\rightarrow$ the Metropolis algorithms behaves like a **random walk**
  - T small: the probability to accept deviations from downhill is very low

Simulated Annealing

Simulated Annealing: a physical analog

- Take solid and raise it to high temperature, we do not expect it to maintain a nice crystal structure
- Take a molten solid and freeze it very abruptly, we do not expect to get a perfect crystal either
- Annealing: cool material gradually from high temperature, allowing it to reach equilibrium at succession of intermediate lower temperatures

Simulated Annealing

- We can try to mimic the process of annealing
- Basically, we take the Metropolis algorithm and we gradually decrease the value of T over the course of each iteration
- The exact way in which T is updated, is called the **cooling schedule**
- Hence, we have $T = \tau(i)$

The exact way of choosing $T = \tau(i)$ may be difficult and varies from problem to problem

- Any ideas here?

Simulated Annealing

How does Simulated Annealing behave with the previous examples of Vertex Cover problems?

Facility Location Problems

- The goal: serve a set of demand points, typically called as **clients**, by a opening a set of access points, typically called as **facilities**, in a cost effective and efficient manner
- The distances between the clients and facilities are assumed to satisfy **metric properties**
- There is a location dependent cost for opening a facility at each location
- The sum of costs of opening the facilities is called the **facility cost**, and the sum of distances of each client to the facility it is assigned to is called as the **service cost**

Solution to FLP

A solution to a facility location problem is specified by a **set of facilities to be opened** and an **assignment** of the clients to the open facilities

Facility Location Problems

- The set of facilities and the set of clients form a part of the input for all facility location variants
- We denote the set of facilities by F and the set of clients by C
- The distance between a pair of points $i, j \in F \cup C$ is denoted by c_{ij}
  - $c_{ij} = 0 \iff i = j$
  - Symmetry property
  - Triangle inequality

We are required to open a subset of the facilities and serve the demands of clients by assigning them to one of the open facilities

Facility Location Problems

- Suppose S is a set of open facilities

$\rightarrow$ Then, $\forall j \in C$, $dist(j, S)$ denotes the **minimum distance** between j and a facility in S

Service Cost:

- If a client j is served by a facility i in a solution, then the distance c_{ij} is said to be the service cost of client j
- The sum of service costs of all clients is the service cost of the solution

Facility Cost:

- There is a cost associated with opening a facility at $i \in F$, denoted by f_i
- Certain variants of facility location may also place a limit on the number of clients a facility can serve, namely its **capacity**

k-median problem

Motivations:

This problem is motivated by scenarios in which a limited budget is available for opening the facilities and the cost of all the facilities are roughly the same

- **Input**: The set of facilities F, the set of clients C, and the distance metric. An integer k which is the **maximum** number of facilities that can be opened
- **Output**: Find a set $S \subseteq F$ such that $|S| \leq k$, and $\sum_{i \in C} dist(i, S)$ is minimized

We show that a simple local search heuristic gives a $3(1 + \epsilon)$ **approximation** for the k-median problem

Uncapacitated Facility Location (UFL)

Motivations:

There are facility location situations in which both the facility cost and service cost are incurred only once. The UFL problem models such scenarios

- **Input**: The set of facilities F, the set of clients C, and the distance metric. For each $i \in F$, the cost of opening a facility at i, denoted by f_i, is also given
- **Output**: Find a set $S \subseteq F$ such that $\sum_{i \in S} f_i + \sum_{i \in C} dist(i, S)$ is minimized

We provide a local search heuristic with a $3(1 + \epsilon)$ approximation for the UFL problem. Our algorithm considers very simple local operations.

k-Uncapacitated Facility Location (k-UFL)

Motivations:

This problem is a variant of uncapacitated facility location in which a limit is placed on the number of facilities which can be opened

- **Input**: The set of facilities F, the set of clients C, an integer k, and the distance metric. For each $i \in F$, cost of opening a facility at i, denoted by f_i, is also given
- **Output**: Find a set $S \subseteq F$ such that $|S| \leq k$ and $\sum_{i \in S} f_i + \sum_{i \in C} dist(i, S)$ is minimized

Note that, k-UFL is a generalization of previous problems. It reduces to the k-median problem if all the facility costs are zero and reduces to the UFL if k is equal to the number of the facilities in F

Facility Location with Soft Capacities (∞-CFL)

Motivations:

The uncapacitated facility location ignores the fact that the cost of a facility could depend on the number of clients it serves. This problem associates capacities with the facilities and makes the cost of facility vary linearly with respect to the number of clients it serves modulo its original capacity

- **Input**: The set of facilities F, the set of clients C, and the distance metric. For each $i \in F$, cost of opening a facility at i denoted by f_i, and a capacity u_i which is the maximum number of clients a facility at i can serve

Facility Location with Soft Capacities (∞-CFL)

- **Output**:
  - A function h from the set of facilities to the set of integers, $h : F \rightarrow \mathbf{N}$, which specifies the number of copies of a facility being opened.
  - Assignment of clients to the set of facilities, $g : C \rightarrow F$. The assignment should be such that $(s_i = |\{j \in C | g(j) = i\}| \leq h(i) \cdot u_i), \forall i \in F$

In other words, we have to open a subset of facilities (multiple copies of a facility can be allowed which increases its capacity by a factor of its original capacity) and assign clients to the open facilities such that no facility serves more than its effective capacity. The goal is to minimize $\sum_{i \in S} h(i) \cdot f_i + \sum_{j \in C} c_{jg(j)}$

Budget Constrained k-median problem

Motivations:

- We introduce the following problem motivated by the contrasting objectives of the k-median and k-center problems
- The objective function in k-median problem **minimizes the average distance** traveled by the clients
- In the k-center problem, the goal is to open k facilities such that the **maximum distance** of any client from its nearest facility is minimized
- In many situations, it is desirable to obtain a simultaneous approximation for both the problems
- We consider the problem of minimizing the total service cost of when a limit is placed on the maximum service cost that can be incurred by a client

Budget Constrained k-median problem

- **Input**: The set of clients C. The distances between clients in C satisfy metric properties. Input also consists of an integer k and a budget B. For a client $j \in C$ and a set $S \subseteq C$, $ds(j, S)$ denotes $\min_{i \in S} c_{ij}$
- **Validity**:A valid solution $S \subseteq C$ is such that, $(|S| \leq k) \wedge (\forall i \in C : ds(i, S) \leq B)$
- **Assumption**: The input has at least one valid solution
- **Output**: A valid solution S such that $\sum_{i \in C} ds(i, S)$ is minimized

Techniques for FL problems

Greedy Heuristics

Approximation algorithms based on greedy heuristics were the first to be proposed for facility location problems

- Facility location problems reduced to variants of set cover (See Lecture 5)
- Algorithms along the lines of the greedy heuristic for the set cover problem, with $O(logn)$ bounds

LP Rounding Techniques

- Approximation algorithms based on rounding the fractional optimal solution to the LP relaxation of the original integer programs (as discussed in Lecture 5)

Local Search and Locality-Gap

A generic local search algorithm

Algorithm 1: Local Search

Let S be an arbitrary feasible solution in S
while $\exists S' \in \mathcal{N}(S)$ *such that* $cost(S') < cost(S)$ **do**
  $S \leftarrow S'$
end
return S

Local Search and Locality-Gap

- Consider an optimization problem P and an instance I of the problem
- A local search algorithm $LS(P)$ produces a solution to the instance I by iteratively exploring the space of all feasible solutions to I
- Formally, the algorithm can be described by:
  - The set $\mathcal{S}$ of all feasible solutions to the input instance
  - A cost function cost: $\mathcal{S} \rightarrow \Re$
  - A neighborhood structure $\mathcal{N} : \mathcal{S} \rightarrow 2^{\mathcal{S}}$
  - An **oracle** that given any solution $S \in \mathcal{S}$, finds (if possible) a solution $S' \in \mathcal{N}(S)$ such that $cost(S') < cost(S)$

Local Search and Locality-Gap

- A solution $S \in \mathcal{S}$ is called **local optimum** if $cost(S) \leq cost(S') \forall S' \in \mathcal{N}(S)$
- The algorithm defined above returns a locally optimum solution
- The cost function and the neighborhood structure $\mathcal{N}$ will vary depending on the problem and the heuristic being employed
- The neighborhood structure usually specifies the local operations allowed at each step

In case of facility location problems, the algorithm above can be modified suitably to **run it in polynomial** time and **argue approximability**

Local Search and Locality-Gap

- Consider a minimization problem P and a local search procedure to solve P, denoted by $LS(P)$
- For an instance I of the problem P, let $\texttt{global}(I)$ denote the cost of the global optimum and $\texttt{local}(I)$ be the cost of a locally optimum solution provided by $LS(P)$

Locality Gap

We call the **supremum** of the ratio $\texttt{local}(I)/\texttt{global}(I)$, the locality gap of $LS(P)$

Converting Locality-Gap into Approximation Factor

- The generic algorithm above may not always terminate in polynomial time
- To run it in polynomial time, we modify the `while` loop of the algorithm as follows:

Algorithm 2: Local Search

Let S be an arbitrary feasible solution in S
while $\exists S' \in \mathcal{N}(S)$ *such that* $cost(S') \leq (1 - \frac{\epsilon}{Q})cost(S)$ **do**
  $S \leftarrow S'$
end
return S

Where

- $\epsilon > 0$ is a constant
- Q is a suitable integer which is polynomial in the size of the input

Converting Locality-Gap into Approximation Factor

- In the modified algorithm, the cost of the current solution decreases by a factor of at least ϵ/Q
- If O denotes an optimum solution and S_0 denotes the initial solution, then the number of steps in the algorithm is at most $\log\left(cost(S_0)/cost(O)\right)/\log\frac{1}{1-\epsilon/Q}$

- As Q, $\log\left(cost(S_0)\right)$, and $\log\left(cost(O)\right)$ are polynomial in the input size, the algorithm terminates after polynomially many local search steps
- We choose Q such that, the algorithm with the above modification continues to have a small locality gap

Converting Locality-Gap into Approximation Factor

- We now present a generic technique for proving a bound on the locality gap
- If S is a locally optimum solution then for all $S' \in \mathcal{N}(S)$:

$$cost(S') - cost(S) \geq 0$$

- The key to arguing locality gap is to identify a suitable, polynomially large (in the input size) subset $\mathcal{Q} \in \mathcal{N}(S)$ of neighboring solutions which satisfies the following property:

$$\sum_{S' \in \mathcal{Q}} (cost(S') - cost(S)) \leq \alpha \cdot (cost(O) - cost(S))$$

where:

- O is an optimum solution
- $\alpha > 1$ is a suitable constant

Converting Locality-Gap into Approximation Factor

But:

- $\sum_{S' \in \mathcal{Q}}(cost(S') - cost(S)) \geq 0$ as S is locally optimum

$\Rightarrow$ $cost(S) \leq \alpha cost(O)$
which constitutes a bound of α on the locality gap

- Let us now consider a solution S output by the modified algorithm, with $Q = |\mathcal{Q}|$
- To analyze the quality of S, we note that:

$$\forall S' \in \mathcal{Q},\ cost(S') > (1 - \epsilon/Q)cost(S)$$

Converting Locality-Gap into Approximation Factor

Hence:

$$\alpha \cdot \Big(cost(O) - cost(S) \Big) \geq \sum_{S' \in \mathcal{Q}} (cost(S') - cost(S)) \geq -\epsilon \cdot cost(S)$$

Which implies that:

$$cost(S) \leq \frac{\alpha}{(1-\epsilon)} cost(O)$$

Locality Gap - Approximation Factor

Thus our proof that a certain local search procedure has a locality gap of at most α translates into a $\alpha/(1-\epsilon)$ approximation algorithm with a running time that is polynomial in the input size and $1/\epsilon$.

The k-median problem

- Consider a facility location problem where all the facilities have to be opened inside the same city
- It is reasonable to assume that the cost of opening a facility is roughly the same in all the locations
- Suppose that there is a fixed budget for opening the facilities
- This implies that the total number of facilities that can be opened is bounded by the ratio of budget to the cost of a single facility

The k-median problem

A LS heuristic

- Consider a simple `swap` operation at each step of the algorithm
- The algorithm starts with an arbitrary subset of k facilities
- At each step, it tries to improve the solution by removing one of the facilities from current solution and adding a new facility. The algorithm terminates when the solution cannot be improved in this manner

The k-median problem: Notations

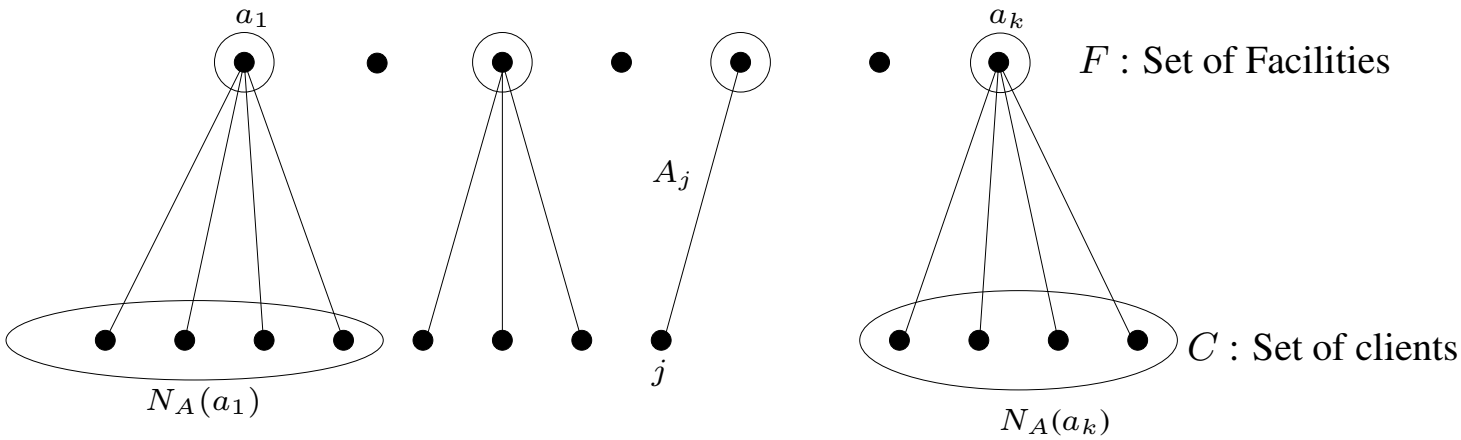
- Consider a solution to an instance of the k-median problem
- Suppose S is the set of facilities which are opened
- The most natural and in fact, **optimal assignment** of clients to the facilities in S is: assign each client to the **closest facility** in S

$\rightarrow$ The service cost of each client is well defined once the set S is specified

Assignment Function:

- A solution to a typical facility location problem is given by a set of open facilities A, and an assignment of each client to an open facility
- Let $\sigma : C \rightarrow A$ denote the assignment function

Example Instance of k-median problem



The k-median problem: Notations

Neighborhoods

- The **neighborhood** of a facility is the set of clients assigned to it. Formally, for a facility $a \in A$, the neighborhood of a, is denoted by $N_A(a) = \{j | \sigma(j) = a\}$
- For a subset of facilities $T \subseteq A$, let $N_A(T) = \bigcup_{a \in T} N_A(a)$

Service Cost

The service cost of a client j is its distance from the facility it is assigned to, and is defined as $c_{j\sigma(j)}$

The k-median problem: LS with Single Swaps

- We consider a local search whose only local operation is a single `swap` which improves the cost
- A swap is effected by closing a facility $s \in S$ and opening a facility $s' \neq s \in S$ and is denoted by $< s, s' >$
- Hence, $N(S) = \{S - \{s\} + \{s'\} | s \in S\}$
- We start with an arbitrary set of k facilities and keep improving our solution with such swaps until we reach a locally optimum solution
- Notation: we use $S - s + s'$ to denote $S - \{s\} + \{s'\}$

The k-median problem: The LS algorithm

Algorithm 3: k-median Local Search Algorithm

Let $S \subseteq F$ be such that $|S| \leq k$
while $\exists s \in S,\ s' \in F$ *s.t.* $cost(S - s + s') < cost(S)$ **do**
  $S \leftarrow S - s + s'$
end
return S

Analysis of the LS algorithm

- We now show that this local search procedure has a locality gap of $\alpha = 5$
- S denotes the locally optimum solution output by the local search procedure and O denotes an optimum solution
- From the local optimality of S, we know that:

$$cost(S - s + o) \geq cost(S) \, \forall s \in S, \, o \in O$$

- Note that even if $S \cap O \neq \emptyset$, the above inequalities hold

Analysis of the LS algorithm

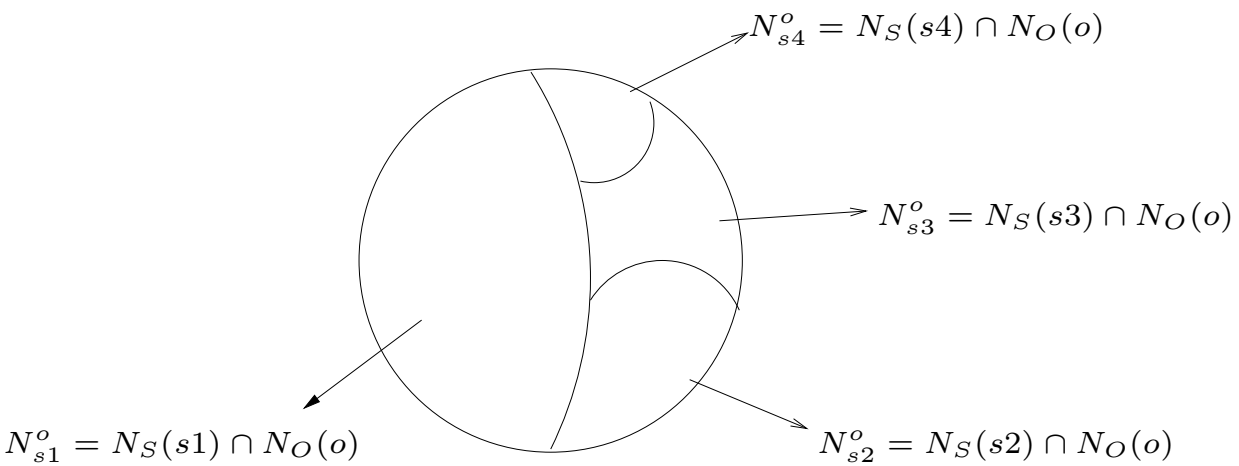
Our goal:

We combine these inequalities judiciously to show that

$$cost(S) \leq 5 \cdot cost(O)$$

- We consider the following partitioning of the neighborhood of a facility $o \in O$ with repsect to the solution S
- The partitioning of $N_O(o)$ is such that, each partition consists of all the clients in $N_O(o)$ which are served by a unique facility $s \in S$
- Formally, we partition $N_O(o)$ into subsets $N_s^o = N_O(o) \bigcap N_S(s)$ for all $s \in S$ as shown next

Analysis of the LS algorithm



Analysis of the LS algorithm

- The main idea in our proof is to consider suitable mapping between the clients in the neighborhood of each optimum facility, and use the mapping to amortize the cost of all the swaps considered
- Let us first consider a special case of our analysis to understand the main ideas
- Let us assume that, given a neighborhood $N_O(o)$, we can find a $1-1$ and onto function π such that every client in $N_O(o)$ is mapped to a client in $N_O(o)$ which belongs to a partition different from the one it belongs to

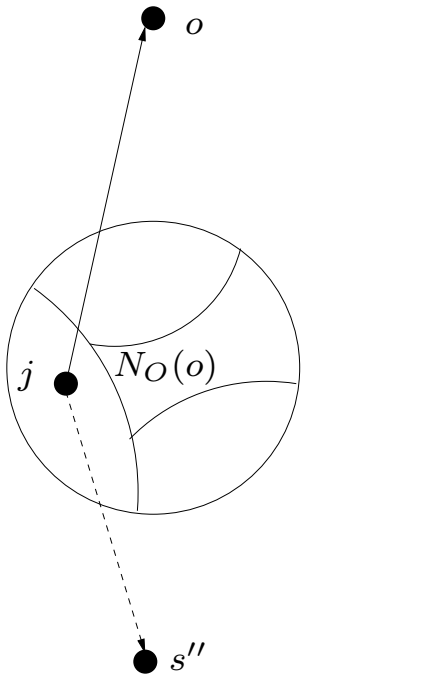
Analysis of the LS algorithm

- Formally, a client $j \in N_s^o$ is mapped to a client $j' \in N_{s'}^o$ such that $s \neq s'$
- Let us consider an arbitrary pairing of the facilities in O with the facilities in S. Specifically, let $< s_1, o_1 > \cdots , < s_k, o_k >$ be the pairings
- We consider the k swaps given by swapping s_i with o_i

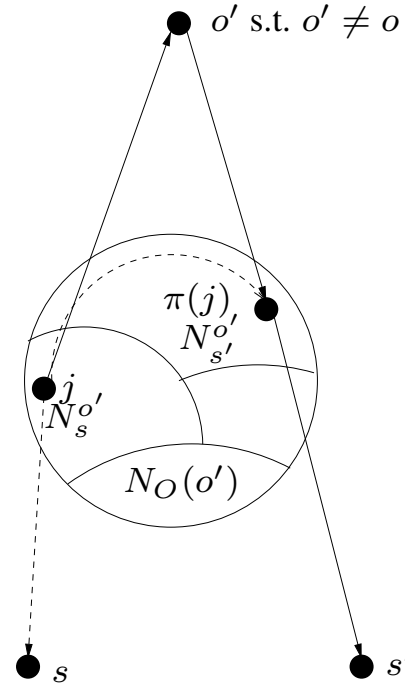
Analysis of the LS algorithm

- When a facility $s \in S$ is swapped with a facility $o \in O$, the clients in the neighborhood of s given by $N_S(s)$ **have to be reassigned**
- We use a specific reassignment to **bound the change in cost** as a result of the swap
- We use the mapping π to reassign the clients in $N_S(s) \cup N_O(o)$ as shown next

Analysis of the LS algorithm



Reassigning a client $j \in N_O(o)$



Reassigning a client $j \in N_S(s) \setminus N_s^o$

Analysis of the LS algorithm

Simple case:

- A client $j \in N_O(o)$ is reassigned to o
- The change in cost due to this reassignment is $O_j - S_j$

Complex case:

- A client $j \in N_S(s) \setminus N_s^o$ is reassigned as follows:
- Let $\pi(j) \in N_{s'}^o$
- By our assumption, $s \neq s'$
- We reassign j to s'
- By **triangle inequality**, the change in cost due to this reassignment is **bounded** by $O_j + O_{\pi(j)} + S_{\pi(j)} - S_j$
- The overall change in cost due to any of these swaps is greater than zero as S is a local optimum

Analysis of the LS algorithm

- The change in cost over all the k swaps defined above gives rise to the following equation:

$$\sum_{i \in \{1,\cdots,k\}} \left(\sum_{j \in N_O(o_i)} (O_j - S_j) + \sum_{j \in N_S(s_i) \setminus N_{s_i}^{o_i}} (O_j + O_{\pi(j)} + S_{\pi(j)} - S_j) \right) \geq 0$$

- Note 1: $\cup_{i \in \{1,\cdots,k\}} N_O(o_i) = C$
- Note 2: $O_j + O_{\pi(j)} + S_{\pi(j)} - S_j \geq 0 \, \forall j \in C$

Analysis of the LS algorithm

Hence:

- The set $N_S(s_i) \setminus N_{s_i}^{o_i}$ can be replaced by $N_S(s_i)$
- Thus:

$$\sum_{j \in C}(O_j - S_j) + \sum_{i \in \{1,\cdots,k\}} \left(\sum_{j \in N_S(s_i)} (O_j + O_{\pi(j)} + S_{\pi(j)} - S_j) \right) \geq 0$$

- The first term in the above equation is equal to $cost(O) - cost(S)$
- Also, note that $\cup_{i \in \{1,\cdots,k\}} N_S(s_i) = C$

Analysis of the LS algorithm

Thus:

$$cost(O) - cost(S) + \sum_{j \in C}(O_j + O_{\pi(j)} + S_{\pi(j)} - S_j) \geq 0$$

- But the mapping π is $1-1$ and onto
- $\sum_{j \in C}(S(\pi(j)) - S_j) = 0$
- $\sum_{j \in C}(O_j + O_{\pi(j)}) = 2 \cdot cost(O)$

Hence we have:

$$3 \cdot cost(O) \geq cost(S)$$

Analysis of the LS algorithm

- Observe that the $1-1$ and onto mapping function π **is very crucial** in amortizing the cost over all the k swaps
- The correctness of the proof is also dependent on the existence of such a mapping function, which we still din't characterize
- It is easy to see that such a mapping function does not exist if there are facilities $s \in S$, $o \in O$ such that $|N_s^o| > \frac{1}{2}|N_O(o)|$
- This leads us to categorize the facilities in S based on whether they serve more than half the clients of at least one facility in the optimal solution O

Analysis of the LS algorithm

Definition

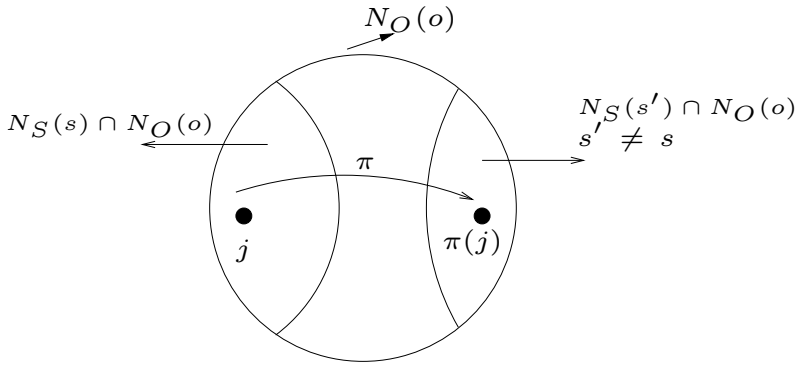
We say that a facility $s \in S$ captures a facility $o \in O$ if s serves more than half the clients served by o,that is, $|N_s^o| > \frac{1}{2}|N_O(o)|$

- It is easy to see that a facility $o \in O$ is captured by at most one facility in S
- We call a facility $s \in S$ **bad**, if it captures some facility $o \in O$, and **good** otherwise

Analysis of the LS algorithm

Property:

If s does not capture o, that is $|N_s^o| \leq \frac{1}{2}|N_O(o)|$, then $\pi(N_s^o) \cap N_s^o = \emptyset$



s does not capture o
π is a one-to-one, and onto mapping.

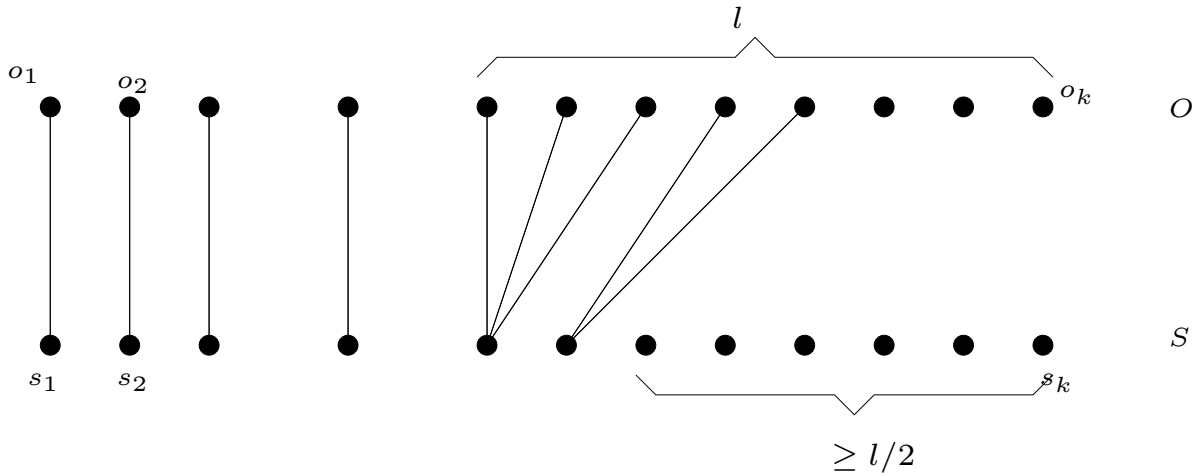
Analysis of the LS algorithm

- The notion of capture can be used to construct a bipartite graph $H = (S, O, E)$ (see next figure)
- For each facility in S, we have a vertex on the S-side and for each facility in O, we have a vertex on the O-side
- We add an edge between $s \in S$ and $o \in O$ if s captures o
- We call H the **capture graph**

Note:

It is easy to see that each vertex on the O-side has degree at most one, while vertices on the S-side can have degree up to k

Analysis of the LS algorithm



Analysis of the LS algorithm

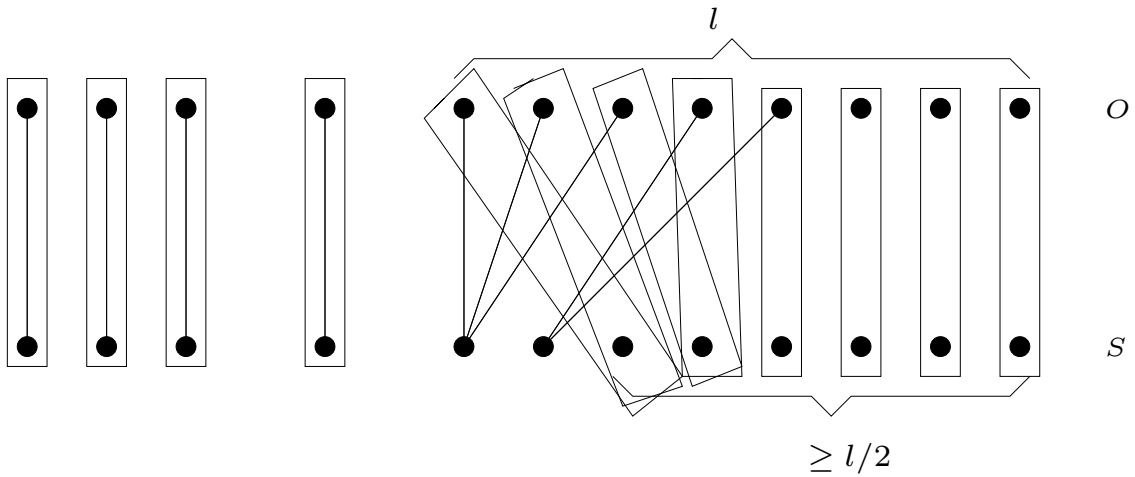
- Consider a facility $s \in S$ which captures a facility $o \in O$
- Suppose s is **swapped** with a facility $o' \neq o \in O$
- By the definition of the mapping function π, it cannot be used to reassign the clients in N_s^o
- Thus, s is constrained to be swapped out with o
- Suppose s captures two optimum facilities o_1, o_2
- Clearly, s is constrained to be swapped with o_1 **or** o_2
- When s is swapped with one of them, say o_1, the clients in $N_s^{o_2}$ cannot be reassigned using the mapping function π

Any facility in S which captures two or more facilities in the optimum **cannot be involved in any swap**

Analysis of the LS algorithm

- We now consider k swaps, one for each facility in O
- If some bad facility $s \in S$ captures exactly one facility $o \in O$ then we consider the swap $< s, o >$
- Suppose l facilities in S (and hence l facilities in O) **are not considered** in such swaps
- Each facility out of these l facilities in S is either **good** or captures **at least two facilities** in O
- Hence there are at least $l/2$ good facilities in S.
- Now, consider l swaps in which the remaining l facilities in O get swapped with the good facilities in S such that each good facility is considered in at most two swaps.
- The bad facilities which capture at least two facilities in O are not considered in any swaps

Analysis of the LS algorithm



Analysis of the LS algorithm

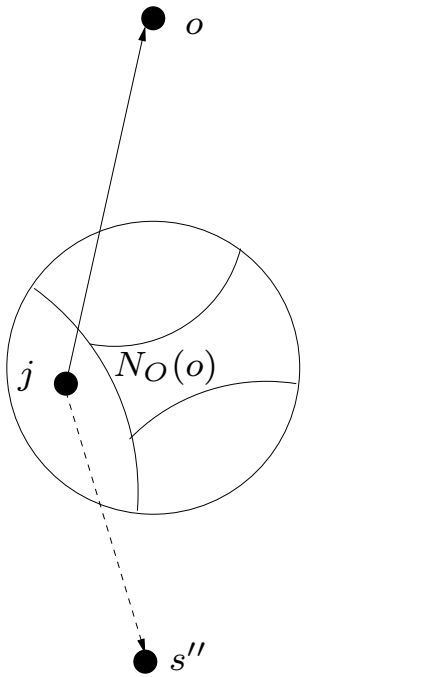
The swaps considered above satisfy the following properties:

1. Each $o \in O$ is considered in exactly one swap
2. A facility $s \in S$ which captures more than one facility in O is not considered in any swap
3. Each good facility $s \in S$ is considered in at most two swaps
4. If swap $< s, o >$ is considered then facility s does not capture any facility $o' \neq o$

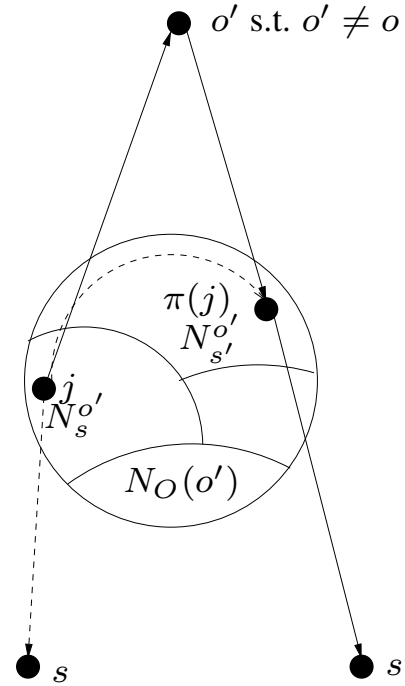
Analysis of the LS algorithm

- We now analyze each of these swaps
- Consider a swap $< s, o >$
- We place an upper bound on the increase in the cost due to this swap by reassigning the clients in $N_S(s) \cup N_O(o)$ to the facilities in $S - s + o$ as follows (see next figure)

Analysis of the LS algorithm



Reassigning a client $j \in N_O(o)$



Reassigning a client $j \in N_S(s) \setminus N_s^o$

Analysis of the LS algorithm

- The clients $j \in N_O(o)$ are now assigned to o
- Consider a client $j \in N_s^{o'}$ for $o \neq o'$
- As s does not capture o', by the property of π, we have that $\pi(j) \notin N_S(s)$
- Let $\pi(j) \in N_S(s')$
- Note that the distance that the client j travels to the nearest facility in $S - s + o$ is at most $c_{js'}$
- From the triangle inequality:
  $c_{js'} \leq c_{jo'} + c_{\pi(j)o'} + c_{\pi(j)s'} = O_j + O_{\pi(j)} + S_{\pi(j)}$
- The clients that do not belong to $N_S(s) \cup N_O(o)$ continue to be served by the same facility

Analysis of the LS algorithm

- We have that:

$$cost(S - s + o) - cost(S) \geq 0$$

- Therefore:

$$\sum_{j \in N_O(o)} O_j - S_j + \sum_{j \in N_S(s), j \notin N_O(o)} O_j + O_{\pi(j)} + S_{\pi(j)} - S_j \geq 0$$

Analysis of the LS algorithm

- As each facility $o \in O$ is considered in exactly one swap, the first term of inequality of the previous inequality added over all k swaps gives exactly

$$cost(O) - cost(S)$$

- For the second term, we will use the fact that each $s \in S$ is considered in at most two swaps
- Since S_j is the shortest distance from client j to a facility in S, using the triangle inequality we get

$$O_j + O_{\pi(j)} + S_{\pi(j)} \geq S_j$$

- Thus, the second term of the previous inequality, added over all k swaps gives:

$$2 \sum_{j \in C} O_j + O_{\pi(j)} + S_{\pi(j)} - S_j$$

Analysis of the LS algorithm

$$2\sum_{j\in C} O_j + O_{\pi(j)} + S_{\pi(j)} - S_j$$

But:

$$\sum_{j\in C} O_j = \sum_{j\in C} O_{\pi(j)} = cost(O)$$

$$\sum_{j\in C} S_{\pi(j)} - S_j = 0$$

Thus:

$$2\sum_{j\in C} O_j + O_{\pi(j)} + S_{\pi(j)} - S_j = 4 \cdot cost(O)$$

Analysis of the LS algorithm

Theorem:

A local search procedure for the metric k-median problem with the local neighborhood structure defined by, $\mathcal{N}(S) = \{S - \{s\} + \{s'\} | s \in S\}$ has a locality gap:

$$5 \cdot cost(O) \geq cost(S)$$

The Uncapacitated Facility Location Problem

Exercise:

Given what we learned for the k-median problem, can you define a LS algorithm for the UFL?