Greedy Approximation Algorithms.

Curs 2018

Greedy and Approximations algorithms

Many times the Greedy strategy yields a feasible solution with value which is near to the optimum solution.
In many practical cases, when finding the global optimum is hard, it is sufficient to find a good feasible solution, an approximation.

Given an optimization problem (maximization or minimization) an optimal algorithm computes the best output OPT (e) on any instance e of size n according to a unction c.
An approximation algorithm for the problem computes a close valid output.

We want to design approximation algorithms, that are fast and output solutions as close as possible to OPT (e).

Greedy and Approximations algorithms

Given an optimization problem, an α-approximation algorithm $\mathcal{A}px$ computes a solution whose cost is within an $\alpha \geq 1$ factor of the cost OPT (e):

$$\frac{1}{\alpha} \leq \frac{c(\mathcal{A}px(e))}{c(\text{OPT}\,(e))} \leq \alpha.$$

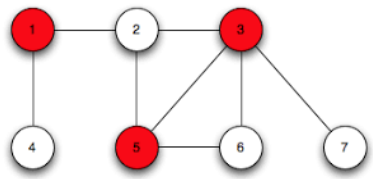
α is called the approximation ratio.

Notice, α measures the factor by which the cost output of $\mathcal{A}px$ exceeds that of OPT (e) , on any input.

The first $\leq$ works for maximization and the second $\leq$ works for minimization.

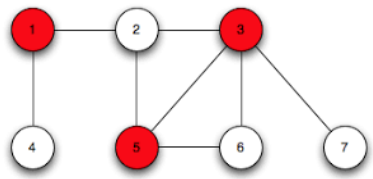
An easy example: Vertex cover

Given a graph $G = (V, E)$ with $|V| = n, |E| = m$ find the minimum set of vertices $S \subseteq V$ such that it covers every edge of G.

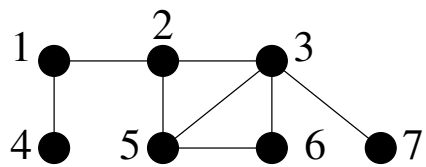


An easy example: Vertex cover

Given a graph $G = (V, E)$ with $|V| = n, |E| = m$ find the minimum set of vertices $S \subseteq V$ such that it covers every edge of G.

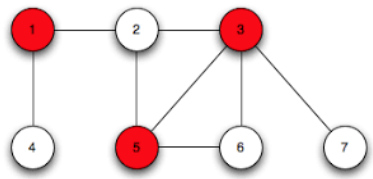


GreedyVC $G = (V, E)$
$E' = E$, $S = \emptyset$,
while $E' \neq \emptyset$ **do**
  Pick $e \in E'$, say $e = (u, v)$
  $S = S \cup \{u, v\}$,
  $E' = E' - \{(u, v) \cup \{\text{edges incident to } u, v\}\}$
end while
return S.

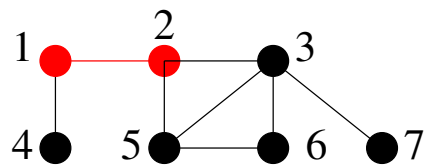


An easy example: Vertex cover

Given a graph $G = (V, E)$ with $|V| = n, |E| = m$ find the minimum set of vertices $S \subseteq V$ such that it covers every edge of G.

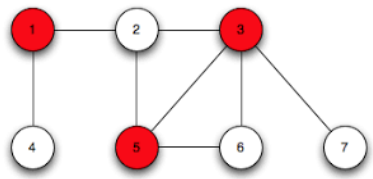


GreedyVC $G = (V, E)$
$E' = E$, $S = \emptyset$,
while $E' \neq \emptyset$ **do**
  Pick $e \in E'$, say $e = (u, v)$
  $S = S \cup \{u, v\}$,
  $E' = E' - \{(u, v) \cup \{\text{edges incident to } u, v\}\}$
end while
return S.

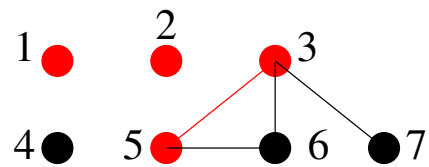


An easy example: Vertex cover

Given a graph $G = (V, E)$ with $|V| = n, |E| = m$ find the minimum set of vertices $S \subseteq V$ such that it covers every edge of G.

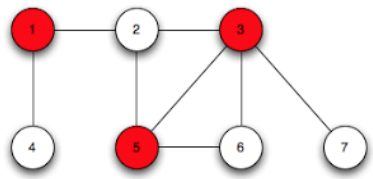


GreedyVC $G = (V, E)$
$E' = E$, $S = \emptyset$,
while $E' \neq \emptyset$ **do**
  Pick $e \in E'$, say $e = (u, v)$
  $S = S \cup \{u, v\}$,
  $E' = E' - \{(u, v) \cup \{\text{edges incident to } u, v\}\}$
end while
return S.

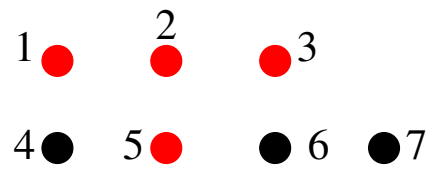


An easy example: Vertex cover

Given a graph $G = (V, E)$ with $|V| = n, |E| = m$ find the minimum set of vertices $S \subseteq V$ such that it covers every edge of G



GreedyVC $G = (V, E)$
$E' = E$, $S = \emptyset$,
while $E' \neq \emptyset$ **do**
  Pick $e \in E'$, say $e = (u, v)$
  $S = S \cup \{u, v\}$,
  $E' = E' - \{(u, v) \cup \{\text{edges incident to } u, v\}\}$
end while
return S.



An easy example: Vertex cover

Theorem

The algorithm $\mathcal{A}px$ runs in $O(m+n)$ steps. Moreover, $|\mathcal{A}px(e)| \leq 2|OPT(e)|$.

An easy example: Vertex cover

Theorem

The algorithm $\mathcal{A}px$ runs in $O(m+n)$ steps. Moreover, $|\mathcal{A}px(e)| \leq 2|OPT(e)|$.

Proof.

We use induction to prove $|\mathcal{A}px(e)| \leq 2|\mathrm{OPT}(e)|$. Notice for every $\{u, v\}$ we add to $\mathcal{A}px(e)$, either u or v are in $\mathrm{OPT}(e)$.
Base: If $V = \emptyset$ then $|\mathcal{A}px(e)| = |\mathrm{OPT}(e)| = 0$.
Hipothesis: $|\mathcal{A}px(e - \{u, v\})| \leq 2|\mathrm{OPT}(e - \{u, v\})|$. Then,

$$\begin{aligned} |\mathcal{A}px(e)| &= |\mathcal{A}px(e - \{u, v\})| + 2 \leq 2|\mathrm{OPT}(e - \{u, v\})| + 2 \\ &\leq 2(|\mathrm{OPT}(e - \{u, v\})| + 1) + 2 \leq 2|\mathrm{OPT}(e)|. \end{aligned}$$

□

The decision problem for Vertex Cover is NP-complete. Moreover, unless P=NP, vertex cover can't be approximated within a factor $\alpha \leq 1.36$

Clustering problems

Clustering: process of finding interesting structure in a set of data.
Given a collection of objects, organize them into coherent groups with respect to some (distance function $d(\cdot, \cdot)$).
This not necessarily has to be the physical (Euclidean) distance, it could be similarity distance, time to travel, but it must be a metric, i.e.
Recall if d is a metric: $d(x, x) = 0, d(x, y) > 0$ for $x \neq y$, $d(x, y) = d(y, x)$, $d(x, y) > 0$ and $d(x, y) + d(y, z) \leq d(x, z)$.

k-clustering problems: Given a set of data points $X = \{x_1, x_2, \ldots, x_n\}$ together with a distance function on X and given a $k > 0$, want to partition X into k disjoint subsets, a k-clustering, such as to optimize some function (depending on d).

The k-clustering problem

Given as input a set of $X = \{x_1, \ldots, x_n\}$, with distances $D = \{d(x_i, x_j)\}$ and a given integer k:
Find the partition X into k **clusters** $\{C_1, \ldots, C_k\}$ such as to minimize the diameter of the clusters, $\min_j \max_{x,y \in C_j} d(x, y)$.

Covering by balls

Consider a set $X = \{x_1, \ldots, x_n\}$, of points in a space in which a distance D is defined.

A ball $B(c, r)$, with center c and radius r contains the points in X within distance r of c.

Let $\mathcal{C} = \{c_1, \ldots, c_k\}$ be a set of centers.

Define $\mathcal{C}$ to be a r-cover for X if $\forall x \in X$, $\exists c_j \in C$ s.t. $d(x, c_j) \leq r$.

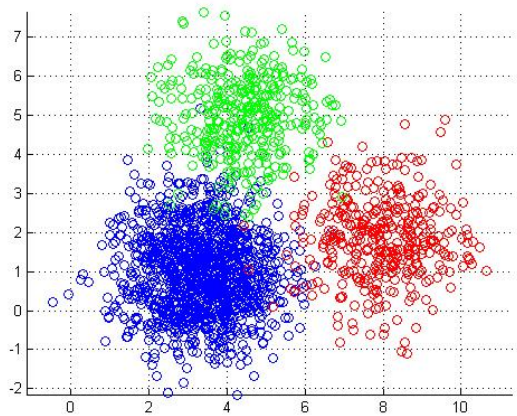
The k-center clustering problem

Given as input (X, D, k), select the centers $C = \{c_1, \ldots, c_k\} \subseteq X$, and $r = r(C)$ such that the resulting $\{C_1, \ldots, C_k\}$ is an r-cover for X, with r as small as possible.

The k-center clustering problem

Given as input (X, D, k), select the centers $C = \{c_1, \ldots, c_k\} \subseteq X$, and $r = r(C)$ such that the resulting $\{C_1, \ldots, C_k\}$ is an r-cover for X, with r as small as possible.

k-center on $\mathbb{Z}^2$: Given $X \subset \mathbb{Z}^2$ points and $k \in \mathbb{Z}$, compute the set $C = \{c_1, \ldots, c_k\}$ of centers $C \subset X$ such that if $\tilde{X} = X \backslash C$, it minimizes $\max_{x \in \tilde{X}} d(x, C)$.



The k-Center clustering problem: Complexity

For $k > 2$, the decision version of the k-center clustering problem is NP-complete.

There is a deterministic algorithm working in $O(n^k)$. (Can you design one?)

The k-Center clustering problem: A greedy algorithm

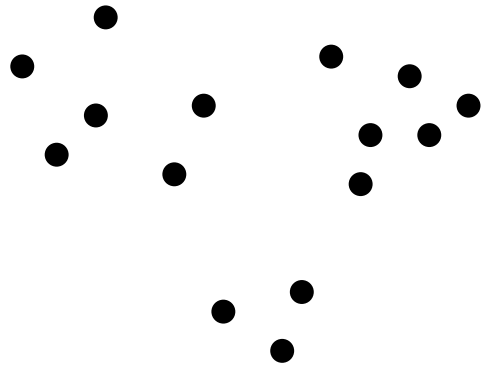
The algorithm iterates k times, at each iteration it chooses a new center, add a new cluster, and it refines the radius r_i of the cluster balls. T. Gonzalez (1985)

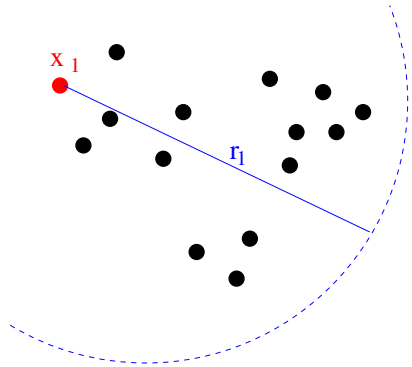
1. Choose arbitrarily x and make $c_1 = x$. Let $C_1 = \{c_1\}$
2. For all $x_i \in X$ let $d_1[x] = d(x_i, c_1)$.
3. Choose $c_2 = x_j$ s.t. $\max\min_{x \in X} d_1[x]$.
4. Let $r_1 = d(c_1, c_2)$ and $C_2 = \{c_1, c_2\}$.
5. For $i = 2$ to k
   - 5.1 At interaction $i + 1$: Let c_{i+1} be the element in $X \setminus C_i$ that maximizes the distances to C_i.
   - 5.2 Let $C_{i+1} = \{c_1, c_2, \ldots, c_{i+1}\}$ and $r_i = \max\min_{j \leq i} d(c_{i+1}, c_j)$,
6. Output the $C = \{c_1, \ldots, c_k\}$ centers and r_k.

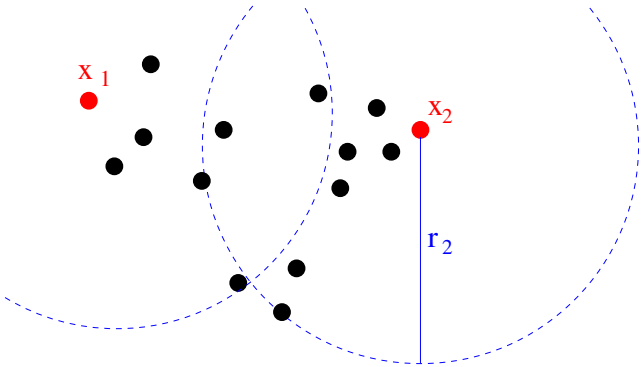
The max min means the max of the P2P distances.

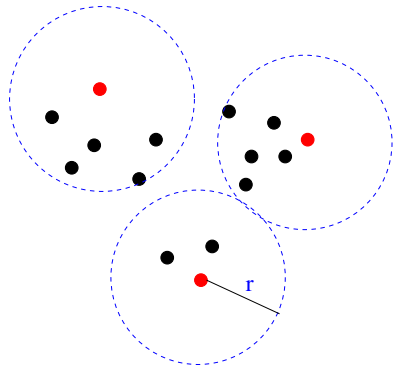
Greedy algorithm: Example

Given X, $k = 3$ and the n^2 distance vector D:









Greedy algorithm: Complexity

We have the set X of points and all their $O(n^2)$ distances so that obtaining $d(x, y)$ requires constant time.

- At each step i we have to compute the distance from all $x \in X$ to all current centers $c \in C_{i-1}$, and choose the new c_i and r_i, but
- For each $x \in X$ define $d_i[x] = d(x, C_i)$
  Observe that $d_i[x] = \min\{d_{i-1}[x], \underbrace{d(x, c_i)}\}$
- Therefore at each step, to compute r_i we need to compute the max of $d_i[x]$ for $x \notin C_{i-1}$.
- At iteration i, choosing c_i and computing r_i takes $O(n)$ steps, therefore the complexity of the greedy algorithm is $O(kn)$ steps.

Approximation to the k-center clustering problem

Theorem

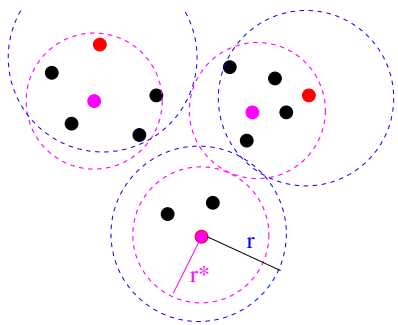
The the resulting diameter in the previous greedy algorithm is an approximation algorithm to the k-center clustering problem, with an approximation ratio of $\alpha = 2$.

(i.e. It returns a set C s.t. $r(C) \leq 2r(C^*)$ where C^* is an optimal set of k-center).

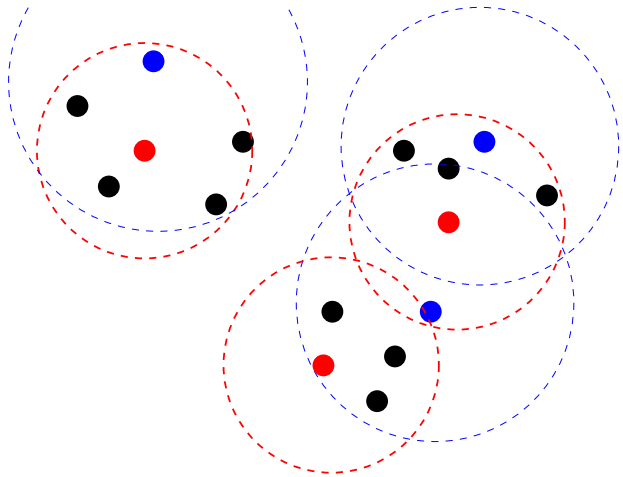
Proof

Let $C^* = \{c_i^*\}_{i=1}^k$ and r^* be the optimal values, and let $C = \{c_i\}_{i=1}^k$ and r the values returned by the algorithm.
Want to prove $r \leq 2r^*$.

Case 1: Every $C_j^* = B(c_j^*, r^*)$ covers at least one c_i. $\Rightarrow$ as $\forall x \in X$, if C_j^* covers x, then $\exists c_i \in C_j^* \Rightarrow d(x, c_i) \leq 2r^*$.



Proof cont.



Case 2: At least one C_j^* does not cover any center in C. Then, $\exists C_j^*$ covering at least c_i and $c_j \Rightarrow d(c_i, c_j) \leq 2r^*$.
We need to prove that $d(c_i, c_j) > r$. Wlog assume the algorithm chooses c_j at iteration j and that c_i has been selected as centre in a previous ith. iteration, then $d(c_i, c_j) > r_j$.
Moreover, notice than $r_1 > r_2 > \ldots > r_k = r$,
therefore $d(c_i, c_j) > r_j \geq r$ and $r \leq d(c_i, c_j) \leq 2r^*$ □

The k-clustering problem

Given as input a set of $X = \{x_1, \ldots, x_n\}$, with distances $D = \{d(x_i, x_j)\}$ and a given integer k:
Find the partition X into k **clusters** $\{C_1, \ldots, C_k\}$ such as to minimize the diameter of the clusters, $\min_j \max_{x,y \in C_j} d(x, y)$.

The k-clustering problem

Given as input a set of $X = \{x_1, \ldots, x_n\}$, with distances $D = \{d(x_i, x_j)\}$ and a given integer k:
Find the partition X into k **clusters** $\{C_1, \ldots, C_k\}$ such as to minimize the diameter of the clusters, $\min_j \max_{x,y \in C_j} d(x, y)$.
What is the difference with the k-center clustering problem?

The k-clustering problem

Given as input a set of $X = \{x_1, \ldots, x_n\}$, with distances $D = \{d(x_i, x_j)\}$ and a given integer k:
Find the partition X into k **clusters** $\{C_1, \ldots, C_k\}$ such as to minimize the diameter of the clusters, $\min_j \max_{x,y \in C_j} d(x, y)$.
What is the difference with the k-center clustering problem?
An approximation algorithm?