Sublinear-time Algorithms

Gouleakis Themistoklis

May 27, 2010

Outline

Definition of sublinear algorithms

- We are familiar with some algorithms that have sublinear running time: e.g Binary search.

Definition of sublinear algorithms

- We are familiar with some algorithms that have sublinear running time: e.g Binary search.
- However, algorithms that need preprossesing (in $\Omega(n)$ time) in order to run in sublinear time are now called "pseudo-sublinear time" algorithms.

Definition of sublinear algorithms

- We are familiar with some algorithms that have sublinear running time: e.g Binary search.
- However, algorithms that need preprossesing (in Ω(n) time) in order to run in sublinear time are now called "pseudo-sublinear time" algorithms.

Definition:

- Algorithms which run in o(n) time without preprossesing of the input are called Sublinear - time Algorithms.
- Note that such algorithms do not read the entire input but only an infinitesimal part of it!

Example 1: Searching in a sorted list

- Our goal is to find if x is one of the n elements given in the input.

Example 1: Searching in a sorted list

- Our goal is to find if x is one of the n elements given in the input.
- We assume that the n elements are stored in a doubly-linked, each list element has access to the next and preceding element in the list, and the list is sorted.

Example 1: Searching in a sorted list

- Our goal is to find if x is one of the n elements given in the input.
- We assume that the n elements are stored in a doubly-linked, each list element has access to the next and preceding element in the list, and the list is sorted.
- We also assume that we have access to all elements in the list
  - All n list elements are stored in an array (but the array is not sorted and we do not impose any order for the array elements).

Example 1: Searching in a sorted list

- Our goal is to find if x is one of the n elements given in the input.
- We assume that the n elements are stored in a doubly-linked, each list element has access to the next and preceding element in the list, and the list is sorted.
- We also assume that we have access to all elements in the list
  - All n list elements are stored in an array (but the array is not sorted and we do not impose any order for the array elements).
- We can easily see that it is impossible to do the search in o(n) time using a deterministic algorithm.
  - However, if we allow randomization, then we can complete the search in $O(\sqrt{n})$ expected time

Randomized algorithm

- Sample uniformly at random a set S of $\Theta(\sqrt{n})$ elements from the input.
- Scan all the elements in S and in $O(\sqrt{n})$ time we can find the max $p \in S$ and the min $q \in S$ such that $p \leq x \leq q$.
- Traverse the input list starting at p until we find either the sought key x or we find element q.

Lemma 1

The algorithm above completes the search in expected $O(\sqrt{n})$ time.

Lemma 1

The algorithm above completes the search in expected $O(\sqrt{n})$ time.

Proof

The running time of the algorithm if equal to $O(\sqrt{n})$ plus the number of the input elements between p and q. Since S contains $\Theta(\sqrt{n})$ elements, the expected number of input elements between p and q is $O(n/|S|) = O(\sqrt{n})$. This implies that the expected running time of the algorithm is $O(\sqrt{n})$.

Example 2:Intersection of 2 polygons

Problem

Given two convex polygons A and B in $\mathbb{R}^2$, each with n vertices, determine if they intersect, and if so, then find a point in their intersection.

Example 2:Intersection of 2 polygons

Problem

Given two convex polygons A and B in $\mathbb{R}^2$, each with n vertices, determine if they intersect, and if so, then find a point in their intersection.

- This problem can be solved in O(n) time, for example, by observing that it can be described as a linear programming instance in 2-dimensions.

Example 2:Intersection of 2 polygons

Problem

Given two convex polygons A and B in $\mathbb{R}^2$, each with n vertices, determine if they intersect, and if so, then find a point in their intersection.

- This problem can be solved in O(n) time, for example, by observing that it can be described as a linear programming instance in 2-dimensions.
- In fact, within the same time one can either find a point that is in the intersection of A and B, or find a line L that separates A from B.

Example 2:Intersection of 2 polygons

Problem

Given two convex polygons A and B in $\mathbb{R}^2$, each with n vertices, determine if they intersect, and if so, then find a point in their intersection.

- This problem can be solved in O(n) time, for example, by observing that it can be described as a linear programming instance in 2-dimensions.
- In fact, within the same time one can either find a point that is in the intersection of A and B, or find a line L that separates A from B.
- Can we obtain a better running time?

$O(\sqrt{n})$ algorithm

We assume that A and B are given by their doubly-linked lists of vertices such that each vertex has as its successor the next vertex of the polygon in the clockwise order.

$O(\sqrt{n})$ algorithm

We assume that A and B are given by their doubly-linked lists of vertices such that each vertex has as its successor the next vertex of the polygon in the clockwise order.

Algorithm

- Sample uniformly at random $\Theta(\sqrt{n})$ vertices from each A and B, and let C_A and C_B be the convex hulls of the sample point sets for the polygons A and B, respectively.
- In $O(\sqrt{n})$ time we can check if C_A and C_B intersects.

- If they don't,let
  - L: the bitangent separating line returned by the algorithm.
  - a,b: The points in L that belong to A and B, respectively.
  - a_1, a_2:the two vertices adjacent to a in A.
  - P_A: We define polygon P_A by walking from a to a1 and then continue walking along the boundary of A until we cross L again (expected size: $O(\sqrt{n})$).

To be continued on the whiteboard...

- If they don't,let
  - L: the bitangent separating line returned by the algorithm.
  - a,b: The points in L that belong to A and B, respectively.
  - a_1, a_2:the two vertices adjacent to a in A.
  - P_A: We define polygon P_A by walking from a to a1 and then continue walking along the boundary of A until we cross L again (expected size: $O(\sqrt{n})$).

To be continued on the whiteboard...

Lemma 2

The problem of determining whether two convex n-gons intersect can be solved in $O(\sqrt{n})$ expected time, which is asymptotically optimal.

Approximating the Average Degree

Assume we have access to the degree distribution of the vertices of an undirected connected graph $G = (V,E)$, i.e., for any vertex $v \in V$ we can query for its degree.

Approximating the Average Degree

Assume we have access to the degree distribution of the vertices of an undirected connected graph G = (V,E), i.e., for any vertex $v \in V$ we can query for its degree.

Problem

Can we achieve a good approximation of the average degree in G by looking at a sublinear number of vertices?

Approximating the Average Degree

Assume we have access to the degree distribution of the vertices of an undirected connected graph G = (V,E), i.e., for any vertex v $\in$ V we can query for its degree.

Problem

Can we achieve a good approximation of the average degree in G by looking at a sublinear number of vertices?

- It seems that approximating the average degree is equivalent to approximating the average of a set of n numbers with values between 1 and n - 1, which is not possible in sublinear time.

Approximating the Average Degree

Assume we have access to the degree distribution of the vertices of an undirected connected graph G = (V,E), i.e., for any vertex $v \in V$ we can query for its degree.

Problem

Can we achieve a good approximation of the average degree in G by looking at a sublinear number of vertices?

- It seems that approximating the average degree is equivalent to approximating the average of a set of n numbers with values between 1 and n - 1, which is not possible in sublinear time.
- But our problem is much easier because the degrees of the vertices we do not sample depends on the degrees of the vertices we do sample!

Proposition

Let d denote the average degree in $G = (V, E)$ and let d_S denote the random variable for the average degree of a set S of s vertices chosen uniformly at random from V . We will show that if we set $s \geq \beta\sqrt{n}/\epsilon^{O(1)}$ for an appropriate constant β, then $d_S \geq (\frac{1}{2} - \epsilon) * d$ with probability at least $1 - \frac{\epsilon}{64}$.

Approximating the Average Degree

- By using Markov inequality we get: $\mathrm{d_S} \leq (1+\epsilon) * \mathrm{d}$ with probability at least $1 - \frac{1}{1+\epsilon} \geq \frac{\epsilon}{2}$.

Approximating the Average Degree

- By using Markov inequality we get: $\mathrm{d_S} \leq (1+\epsilon) * \mathrm{d}$ with probability at least $1 - \frac{1}{1+\epsilon} \geq \frac{\epsilon}{2}$.

Algorithm

Pick $8/\epsilon$ sets $\mathrm{S_i}$ uniformly at random, each of size s, and output the set with the smallest average degree.

Approximating the Average Degree

- By using Markov inequality we get: $\mathrm{d_S} \leq (1+\epsilon) * \mathrm{d}$ with probability at least $1 - \frac{1}{1+\epsilon} \geq \frac{\epsilon}{2}$.

Algorithm

Pick $8/\epsilon$ sets $\mathrm{S_i}$ uniformly at random, each of size s, and output the set with the smallest average degree.

- Hence, the probability that all of the sets $\mathrm{S_i}$ have too high average degree is at most $(1 - \frac{\epsilon}{2})^{8/\epsilon} \leq \frac{1}{8}$.

Approximating the Average Degree

- By using Markov inequality we get: $\mathrm{d_S} \leq (1+\epsilon) * \mathrm{d}$ with probability at least $1 - \frac{1}{1+\epsilon} \geq \frac{\epsilon}{2}$.

Algorithm

Pick $8/\epsilon$ sets $\mathrm{S_i}$ uniformly at random, each of size s, and output the set with the smallest average degree.

- Hence, the probability that all of the sets $\mathrm{S_i}$ have too high average degree is at most $(1 - \frac{\epsilon}{2})^{8/\epsilon} \leq \frac{1}{8}$.
- The probability that one of them has too small average degree is at most $\frac{8}{\epsilon} * \frac{\epsilon}{64} = \frac{1}{8}$.

Approximating the Average Degree

- By using Markov inequality we get: $d_S \leq (1+\epsilon) * d$ with probability at least $1 - \frac{1}{1+\epsilon} \geq \frac{\epsilon}{2}$.

Algorithm

Pick $8/\epsilon$ sets S_i uniformly at random, each of size s, and output the set with the smallest average degree.

- Hence, the probability that all of the sets S_i have too high average degree is at most $(1 - \frac{\epsilon}{2})^{8/\epsilon} \leq \frac{1}{8}$.
- The probability that one of them has too small average degree is at most $\frac{8}{\epsilon} * \frac{\epsilon}{64} = \frac{1}{8}$.
- Hence, the output value will satisfy both inequalities with probability at least $3/4$.

Approximating the Average Degree

- By using Markov inequality we get: $\mathrm{d_S} \leq (1+\epsilon) * \mathrm{d}$ with probability at least $1 - \frac{1}{1+\epsilon} \geq \frac{\epsilon}{2}$.

Algorithm

Pick $8/\epsilon$ sets $\mathrm{S_i}$ uniformly at random, each of size s, and output the set with the smallest average degree.

- Hence, the probability that all of the sets $\mathrm{S_i}$ have too high average degree is at most $(1 - \frac{\epsilon}{2})^{8/\epsilon} \leq \frac{1}{8}$.
- The probability that one of them has too small average degree is at most $\frac{8}{\epsilon} * \frac{\epsilon}{64} = \frac{1}{8}$.
- Hence, the output value will satisfy both inequalities with probability at least $3/4$.
- This gives us a $(2+\epsilon)$-approximation algorithm.

Lower bound

- Let H be the set of the $\sqrt{\epsilon n}$ vertices with highest degree in G and let $L = V \setminus H$ be the set of the remaining vertices.

Lower bound

- Let H be the set of the $\sqrt{\epsilon n}$ vertices with highest degree in G and let $L = V \setminus H$ be the set of the remaining vertices.

Proposition

The sum of the degrees of the vertices in L is at least $(\frac{1}{2} - \epsilon)$ times the sum of the degrees of all vertices.

Lower bound

- Let H be the set of the $\sqrt{\epsilon \mathrm{n}}$ vertices with highest degree in G and let $\mathrm{L} = \mathrm{V} \setminus \mathrm{H}$ be the set of the remaining vertices.

Proposition

The sum of the degrees of the vertices in L is at least $(\frac{1}{2} - \epsilon)$ times the sum of the degrees of all vertices.

- Let $\mathrm{d_H}$ be the degree of a vertex with the smallest degree in H.

Lower bound

- Let H be the set of the $\sqrt{\epsilon n}$ vertices with highest degree in G and let $L = V \setminus H$ be the set of the remaining vertices.

Proposition

The sum of the degrees of the vertices in L is at least $(\frac{1}{2} - \epsilon)$ times the sum of the degrees of all vertices.

- Let d_H be the degree of a vertex with the smallest degree in H.
- We assume that all sampled vertices come from the set L.

Lower bound

- Let H be the set of the $\sqrt{\epsilon n}$ vertices with highest degree in G and let $L = V \setminus H$ be the set of the remaining vertices.

Proposition

The sum of the degrees of the vertices in L is at least $(\frac{1}{2} - \epsilon)$ times the sum of the degrees of all vertices.

- Let d_H be the degree of a vertex with the smallest degree in H.
- We assume that all sampled vertices come from the set L.
- Let X_i, $1 \leq i \leq s$, be the random variable for the degree of the ith vertex from S.

Lower bound

- From Hoeffding bounds it follows that:
$$\Pr[\sum_{i=1}^{s} x_i \leq (1-\epsilon)E[\sum_{i=1}^{s} X_i]] \leq e^{-\frac{E[\sum_{i=1}^{s} X_i]\epsilon^2}{d_H}}$$

Lower bound

- From Hoeffding bounds it follows that:
$$\Pr[\sum_{i=1}^{s} x_i \leq (1-\epsilon)E[\sum_{i=1}^{s} X_i]] \leq e^{-\frac{E[\sum_{i=1}^{s} X_i]\epsilon^2}{d_H}}$$
- We know that: $d \geq d_H * |H|/n$

Lower bound

- From Hoeffding bounds it follows that:
$$\Pr[\sum_{i=1}^{s} x_i \leq (1-\epsilon)E[\sum_{i=1}^{s} X_i]] \leq e^{-\frac{E[\sum_{i=1}^{s} X_i]\epsilon^2}{d_H}}$$
- We know that: $d \geq d_H * |H|/n$
- So, $E[X_i] \geq (\frac{1}{2}-\epsilon) * d_H * |H|/n$ and by linearity of expectation: $E[\sum_{i=1}^{s} X_i] \geq s * (\frac{1}{2}-\epsilon) * d_H * |H|/n$

Lower bound

- From Hoeffding bounds it follows that:
$$\Pr[\sum_{i=1}^{s} x_i \leq (1-\epsilon)E[\sum_{i=1}^{s} X_i]] \leq e^{-\frac{E[\sum_{i=1}^{s} X_i]\epsilon^2}{d_H}}$$
- We know that: $d \geq d_H * |H|/n$
- So, $E[X_i] \geq (\frac{1}{2} - \epsilon) * d_H * |H|/n$ and by linearity of expectation: $E[\sum_{i=1}^{s} X_i] \geq s * (\frac{1}{2} - \epsilon) * d_H * |H|/n$
- By choosing s appropriately we can have $d_S \geq (1-\epsilon) * d$ with high probability (depending on s).

Minimum spanning trees

- Let G = (V,E) be an undirected connected weighted graph with maximum degree D and integer edge weights from 1, . . . ,W.

Minimum spanning trees

- Let G = (V,E) be an undirected connected weighted graph with maximum degree D and integer edge weights from 1, . . . ,W.
- We assume that the graph is given in adjacency list representation, i.e., for every vertex v there is a list of its at most D neighbors, which can be accessed from v.

Minimum spanning trees

- Let G = (V,E) be an undirected connected weighted graph with maximum degree D and integer edge weights from 1, . . . ,W.
- We assume that the graph is given in adjacency list representation, i.e., for every vertex v there is a list of its at most D neighbors, which can be accessed from v.
- It is possible to select a vertex uniformly at random.

MST cost approximation algorithm

Main Idea

Express the cost of a minimum spanning tree as the number of connected components in certain auxiliary subgraphs of G.

MST cost approximation algorithm

Main Idea

Express the cost of a minimum spanning tree as the number of connected components in certain auxiliary subgraphs of G.

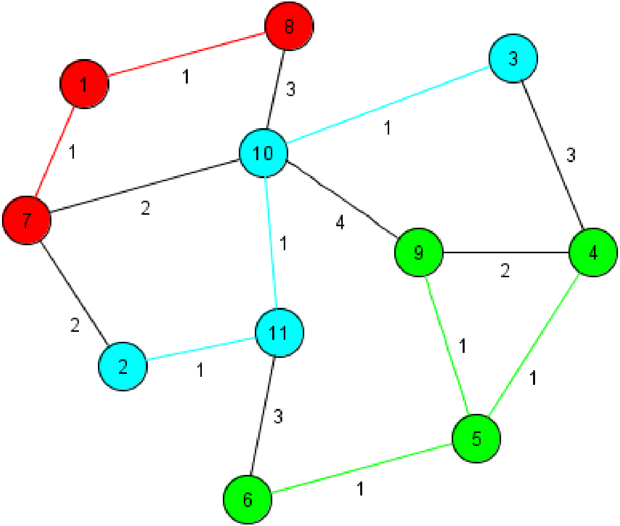
- It can be shown that: $\mathrm{MST} = \mathrm{n} - \mathrm{W} + \sum_{\mathrm{i}=1}^{\mathrm{W}-1} \mathrm{c}^{(\mathrm{i})}$

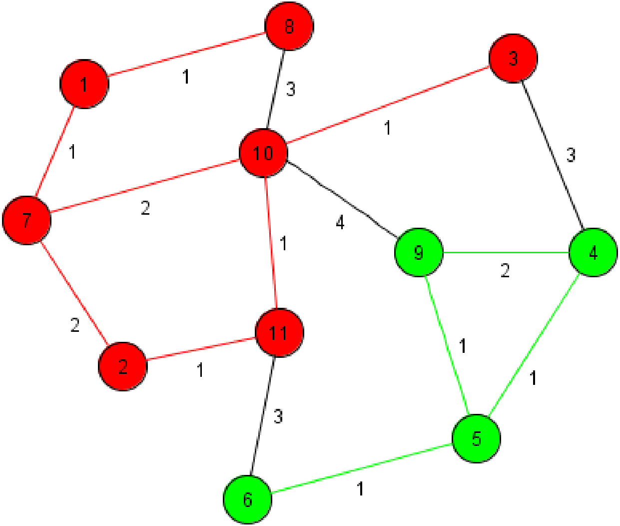
MST cost approximation algorithm

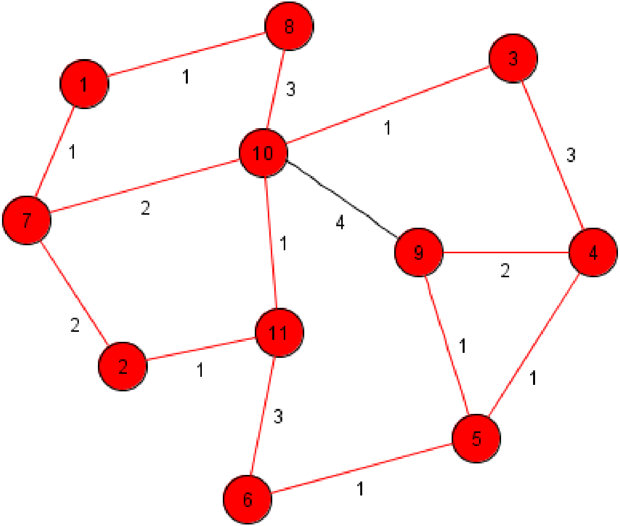
Main Idea

Express the cost of a minimum spanning tree as the number of connected components in certain auxiliary subgraphs of G.

- It can be shown that: $\mathrm{MST} = \mathrm{n} - \mathrm{W} + \sum_{\mathrm{i}=1}^{\mathrm{W}-1} \mathrm{c}^{(\mathrm{i})}$
- So there is a simple algorithm for the approximation of MST weight.


8
1
3
1
3
1
10
1
3
2
7
4
1
9
2
4
2
11
1
2
1
1
3
5
6
1


8
1
3
1
3
1
10
1
3
7
2
4
1
9
2
4
2
11
1
2
1
1
3
5
6
1


8
1
3
1
3
10
1
3
1
2
7
4
1
9
2
4
2
11
1
2
1
1
3
5
1
6

Algorithm

APPROXMSTWEIGHT(G,ε) for i = 1 to W 1 Compute estimator $\underline{c^{(i)}}$ for $c^{(i)}$ output $\underline{\text{MST}} = n - W + \sum_{i=1}^{W-1} \underline{c^{(i)}}$

Approximation algorithm for connected components

Algorithm

APPROXCONNECTEDCOMPS(G, s) Input: an arbitrary undirected graph G
Output: $\underline{c}$: an estimation of the number of connected components of G
Choose s vertices $u_1, \ldots, u_s$ uniformly at random.
for i = 1 to s do choose X according to $\Pr[X \geq k] = 1/k$
run breadth-fist-search (BFS) starting at u_i until either (1) the whole connected component containing $u_i$has been explored, or (2) X vertices have been explored if BFS stopped in case (1) then bi = 1 else bi = 0
output $\underline{c} = \frac{n}{s}\sum_{i=1}^{s} b_i$

Analysis of the algorithm

- Fix an arbitrary connected component C and let |C| denote the number of vertices in the connected component.

Analysis of the algorithm

- Fix an arbitrary connected component C and let |C| denote the number of vertices in the connected component.
- Let c denote the number of connected components in G.

Analysis of the algorithm

- Fix an arbitrary connected component C and let $|C|$ denote the number of vertices in the connected component.
- Let c denote the number of connected components in G.
- $E[b_i] = \sum_{connected components} Pr[u_i \in C] * Pr[X \geq |C|] =$
  $\sum \frac{|C|}{n} * \frac{1}{|C|} = \frac{c}{n}$

Analysis of the algorithm

- Fix an arbitrary connected component C and let |C| denote the number of vertices in the connected component.
- Let c denote the number of connected components in G.
- $E[b_i] = \sum_{connected components} Pr[u_i \in C] * Pr[X \geq |C|] =$
  $\sum \frac{|C|}{n} * \frac{1}{|C|} = \frac{c}{n}$
- By linearity of expectation: $E[\underline{c}] = c$.

Analysis of the algorithm

- Fix an arbitrary connected component C and let |C| denote the number of vertices in the connected component.
- Let c denote the number of connected components in G.
- $\mathrm{E}[\mathrm{b_i}] = \sum_{\mathrm{connectedcomponents}} \mathrm{Pr}[\mathrm{u_i} \in \mathrm{C}] * \mathrm{Pr}[\mathrm{X} \geq |\mathrm{C}|] = \sum \frac{|\mathrm{C}|}{\mathrm{n}} * \frac{1}{|\mathrm{C}|} = \frac{\mathrm{c}}{\mathrm{n}}$
- By linearity of expectation: $\mathrm{E}[\underline{\mathrm{c}}] = \mathrm{c}$.
- $\mathrm{Var}[\mathrm{b_i}] = \mathrm{E}[\mathrm{b_i^2}] - \mathrm{E}[\mathrm{b_i}]^2 \leq \mathrm{E}[\mathrm{b_i^2}] = \mathrm{E}[\mathrm{b_i}] = \frac{\mathrm{c}}{\mathrm{n}}$

Analysis of the algorithm

- Fix an arbitrary connected component C and let $|C|$ denote the number of vertices in the connected component.
- Let c denote the number of connected components in G.
- $E[b_i] = \sum_{connectedcomponents} Pr[u_i \in C] * Pr[X \geq |C|] =$
$$\sum \frac{|C|}{n} * \frac{1}{|C|} = \frac{c}{n}$$
- By linearity of expectation: $E[\underline{c}] = c$.
- $Var[b_i] = E[b_i^2] - E[b_i]^2 \leq E[b_i^2] = E[b_i] = \frac{c}{n}$
- The b_i are mutually independent and so we have

Analysis of the algorithm

- Fix an arbitrary connected component C and let |C| denote the number of vertices in the connected component.
- Let c denote the number of connected components in G.
- $\mathrm{E}[\mathrm{b_i}] = \sum_{\mathrm{connectedcomponents}} \mathrm{Pr}[\mathrm{u_i} \in \mathrm{C}] * \mathrm{Pr}[\mathrm{X} \geq |\mathrm{C}|] =$
  $\sum \frac{|\mathrm{C}|}{\mathrm{n}} * \frac{1}{|\mathrm{C}|} = \frac{\mathrm{c}}{\mathrm{n}}$
- By linearity of expectation: $\mathrm{E}[\underline{\mathrm{c}}] = \mathrm{c}$.
- $\mathrm{Var}[\mathrm{b_i}] = \mathrm{E}[\mathrm{b_i^2}] - \mathrm{E}[\mathrm{b_i}]^2 \leq \mathrm{E}[\mathrm{b_i^2}] = \mathrm{E}[\mathrm{b_i}] = \frac{\mathrm{c}}{\mathrm{n}}$
- The $\mathrm{b_i}$ are mutually independent and so we have
- $\mathrm{Pr}[|\underline{\mathrm{c}} - \mathrm{E}[\underline{\mathrm{c}}]| \geq \lambda \mathrm{n}] \leq \frac{\mathrm{n*c}}{\mathrm{s}*\lambda^2*\mathrm{n}^2} \leq \frac{1}{\lambda^2 * \mathrm{s}}$

From this, it follows that one can approximate the number of connected components within additive error of $\lambda * \mathrm{n}$ in a graph with maximum degree D in $\mathrm{O}(\mathrm{s} * \mathrm{D} * \mathrm{logn}) = \mathrm{O}(\frac{\mathrm{D} * \mathrm{logn}}{\lambda^2 * \varrho})$ time and with probability $1 - \varrho$.

Metric Steiner tree

Steiner tree Problem definition

Given an undirected graph G=(V,E) with nonnegative edge costs and whose vertices are partitioned into two sets, R(equired) and S(teiner) find a minimum cost tree in G that contains all the required vertices and any subset of Steiner vertices.

Metric Steiner tree

Steiner tree Problem definition

Given an undirected graph G=(V,E) with nonnegative edge costs and whose vertices are partitioned into two sets, R(equired) and S(teiner) find a minimum cost tree in G that contains all the required vertices and any subset of Steiner vertices.

Metric Steiner tree

If the edge costs satisfy the triangle inequality $(\forall \mathrm{u}, \mathrm{v}, \mathrm{w} : \mathrm{cost}(\mathrm{u}, \mathrm{v}) \leq \mathrm{cost}(\mathrm{u}, \mathrm{w}) + \mathrm{cost}(\mathrm{w}, \mathrm{v}))$ we call that : Metric Steiner tree problem.

Theorem 3.2

There is an approximation factor preserving reduction from the Steiner tree to the metric Steiner tree problem.

Theorem 3.2

There is an approximation factor preserving reduction from the Steiner tree to the metric Steiner tree problem.

Proof:

Theorem 3.2

There is an approximation factor preserving reduction from the Steiner tree to the metric Steiner tree problem.

Proof:

Construction

- Let G' be $K_{|V|}$
- Define: cost(u,v) (in G')= cost of the shortest path from u to v in G (G' is the metric closure of G).
- The sets R,S remain the same.
- Also, $OPT' \leq OPT$.

Proof

- Let T' be a Steiner tree in G'.
- Replace each edge of T' with the corresponding path of equal cost in T.
- Delete some edges to obtain a tree T.
- As we can see,$\forall \mathrm{T}' \exists \mathrm{T}$ such that $\mathrm{cost}(\mathrm{T}) \leq \mathrm{cost}(\mathrm{T}')$. So, $\mathrm{OPT} \leq \mathrm{OPT}'$.
- Finally OPT=OPT'. And this is an approximation factor preserving reduction.

MST-based algorithm

Theorem 3.3

The cost of an MST on R is within 2*OPT.

MST-based algorithm

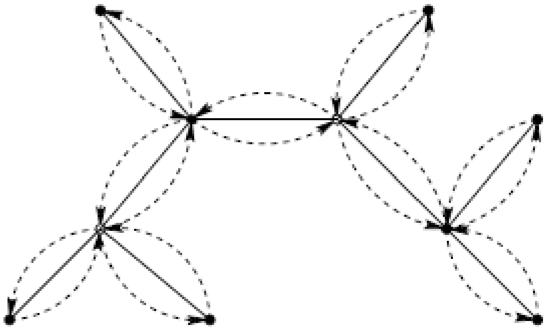
Theorem 3.3

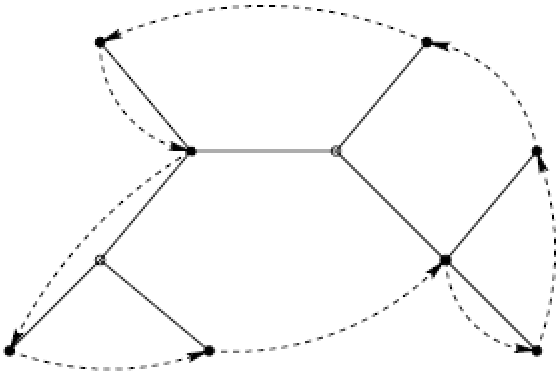
The cost of an MST on R is within 2*OPT.

Proof

- Consider a Steiner tree of cost OPT. By doubling its edges we obtain an Eulerian graph connecting all vertices of R and, possibly, some Steiner vertices.
- Find an Euler tour of this graph.
- Next obtain a Hamiltonian cycle on the vertices of R by traversing the Euler tour and short-cutting Steiner vertices and previously visited vertices of R.

- Because of triangle inequality, the shortcuts do not increase the cost of the tour. If we delete one edge of this Hamiltonian cycle, we obtain a path that spans R and has cost at most 2 OPT. This path is also a spanning tree on R. Hence, the MST on R has cost at most 2 OPT.



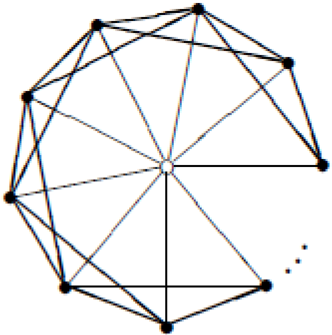


Tight Example

For a tight example, consider a graph with n required vertices and one Steiner vertex. An edge between the Steiner vertex and a required vertex has cost 1, and an edge between two required vertices has cost 2 (not all edges of cost 2 are shown below). In this graph, any MST on R has cost $2(n-1)$, while $OPT = n$

Tight Example

For a tight example, consider a graph with n required vertices and one Steiner vertex. An edge between the Steiner vertex and a required vertex has cost 1, and an edge between two required vertices has cost 2 (not all edges of cost 2 are shown below). In this graph, any MST on R has cost $2(n-1)$, while $OPT = n$



TSP

Travelling salesman problem (TSP)

Given a complete graph with non-negative edge costs, find a minimum cost cycle visiting every vertex exactly once.

TSP

Travelling salesman problem (TSP)

Given a complete graph with non-negative edge costs, find a minimum cost cycle visiting every vertex exactly once.

Theorem 3.6

For any polynomial time computable function $\alpha(n)$, TSP cannot be approximated within a factor of $\alpha(n)$, unless $P = NP$.

Assume, for a contradiction, that there is a factor $\alpha(\mathrm{n})$ polynomial time approximation algorithm, A, for the general TSP problem. We will show that A can be used for deciding the Hamiltonian cycle problem (which is NP hard) in polynomial time, thus implying P = NP.

Assume, for a contradiction, that there is a factor $\alpha(n)$ polynomial time approximation algorithm, A, for the general TSP problem. We will show that A can be used for deciding the Hamiltonian cycle problem (which is NP hard) in polynomial time, thus implying P = NP.

Proof

- The central idea is a reduction from the Hamiltonian cycle problem to TSP, that transforms a graph G on n vertices to an edge-weighted complete graph G' on n vertices such that:
  - if G has a Hamiltonian cycle, then the cost of an optimal TSP tour in G' is n
  - if G does not have a Hamiltonian cycle, then an optimal TSP tour in G' is of cost $\geq \alpha(n) * n$.
- The reduction is simple. Assign a weight of 1 to edges of G, and a weight of $\alpha(n) * n$ to non-edges, to obtain G'.

A simple factor 2 algorithm

The lower bound we will use for obtaining this factor is the cost of an MST in G.

A simple factor 2 algorithm

The lower bound we will use for obtaining this factor is the cost of an MST in G.

Algorithm: Metric TSP factor 2

1. Find an MST, T, of G.
2. Double every edge of the MST to obtain an Eulerian graph.
3. Find an Eulerian tour, T , on this graph.
4. Output the tour that visits vertices of G in the order of their first appearance in T . Let C be this tour.

Theorem 3.8

Algorithm 3.7 is a factor 2 approximation algorithm for metric TSP.

Theorem 3.8

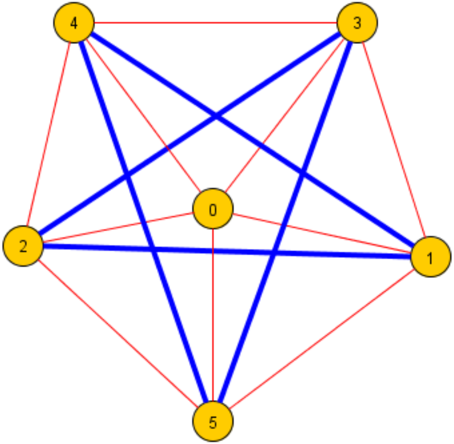
Algorithm 3.7 is a factor 2 approximation algorithm for metric TSP.

Proof

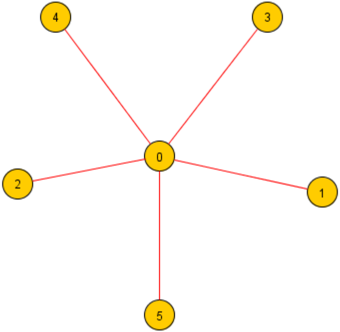
As noted above, cost(T) $\leq$ OPT. Since T' contains each edge of T twice, cost(T') = 2*cost(T). Because of triangle inequality, after the short-cutting step, cost(C) $\leq$ cost(T$'$). Combining these inequalities we get that cost(C) $\leq$ 2OPT.

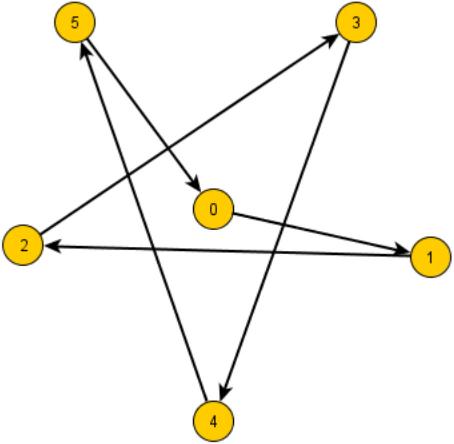
Tight Example

A tight example for this algorithm is given by a complete graph on n vertices with edges of cost 1 and 2. We present the graph for n = 6 below, where thick edges have cost 2 and remaining edges have cost 1. For arbitrary n the graph has 2n-2 edges of cost 1, with these edges forming the union of a star and an n 1 cycle; all remaining edges have cost 2.


4
3
0
2
1
5

Suppose that the MST found by the algorithm is the spanning star created by edges of cost 1. Moreover, suppose that the Euler tour constructed in Step 3 visits vertices in order shown next for n = 6:


0
1
2
3
4
5


5
3
0
2
1
4

Improving the factor to 3/2

Algorithm - factor 3/2

- Find an MST of G, say T.
- Compute a minimum cost perfect matching, M, on the set of odd-degree vertices of T. Add M to T and obtain an Eulerian graph.
- Find an Euler tour, T , of this graph.
- Output the tour that visits vertices of G in order of their first appearance in T . Let C be this tour.

Lemma 3.11

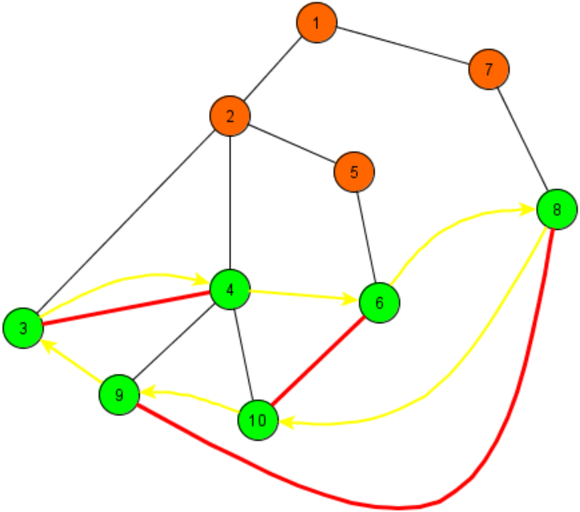
Let $V' \subseteq V$ be the set of odd-degree vertices of G ($|V'|$ is even) and let M be a minimum cost perfect matching on V'. Then, $\mathrm{cost}(M) \leq OPT/2$.

Lemma 3.11

Let $V' \subseteq V$ be the set of odd-degree vertices of G ($|V'|$ is even) and let M be a minimum cost perfect matching on V'. Then, $\text{cost}(M) \leq \text{OPT}/2$.

Proof

Let τ :optimal TSP tour. Let τ' be the tour on V' obtained by short-cutting τ. By the triangle inequality, $\text{cost}(\tau') \leq \text{cost}(\tau)$. Now, τ' is the union of two perfect matchings on V', each consisting of alternate edges of τ. Thus, the cheaper of these matchings has cost $\leq \text{cost}(\tau')/2 \leq \text{OPT}/2$


1
7
2
5
8
4
6
3
9
10

Theorem 3.12

Algorithm 3.10 achieves an approximation guarantee of 3/2 for metric TSP.

Theorem 3.12

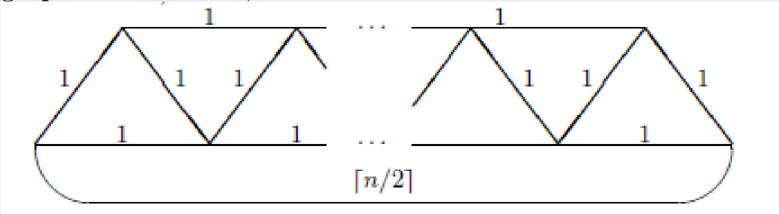
Algorithm 3.10 achieves an approximation guarantee of 3/2 for metric TSP.

Proof

The cost of the Euler tour,
$\mathrm{cost}(\mathrm{T}') \leq \mathrm{cost}(\mathrm{T}) + \mathrm{cost}(\mathrm{M}) \leq \mathrm{OPT} + \frac{1}{2}\mathrm{OPT} = \frac{3}{2}\mathrm{OPT}$, where the first inequality follows by using the two lower bounds on OPT. Using the triangle inequality, $\mathrm{cost}(\mathrm{C}) \leq \mathrm{cost}(\mathrm{T})$, and the theorem follows.

Tight Example

A tight example for this algorithm is given by the following graph on n vertices, with n odd:



Introduction

- Approximation algorithms for clustering problems in metric spaces typically have $\Omega(n^2)$ running time.

Introduction

- Approximation algorithms for clustering problems in metric spaces typically have $\Omega(n^2)$ running time.
- Surprisingly, these lower bounds do not necessarily hold when one wants to estimate the cost of an optimal solution.

Introduction

- Approximation algorithms for clustering problems in metric spaces typically have $\Omega(n^2)$ running time.
- Surprisingly, these lower bounds do not necessarily hold when one wants to estimate the cost of an optimal solution.
- There is a constant factor approximation algorithm for the metric uncapacitated facility location problem with uniform costs and in which every point can open a facility, that runs in $O(n\log^2 n)$time, that is, in time sublinear in the input size.

Problem Definition

(Metric) Minimum Facility Location Problem

We are given a metric (P,D), and a subset $F \subseteq P$ of facilities. For each facility $v \in F$, we are given a non-negative cost f(v), and for each point $u \in P$, a nonnegative demand d(u). The problem consists of finding a set $F' \subseteq F$, so as to minimize I= $\sum_{v \in F'} f(v) + \sum_{u \in P} d(u) * D(u, F')$ where D(u, F') $= \min_{v \in F} D(u, v)$.

Problem Definition

(Metric) Minimum Facility Location Problem

We are given a metric (P,D), and a subset $F \subseteq P$ of facilities. For each facility $v \in F$, we are given a non-negative cost f(v), and for each point $u \in P$, a nonnegative demand d(u). The problem consists of finding a set $F' \subseteq F$, so as to minimize I= $\sum_{v \in F'} f(v) + \sum_{u \in P} d(u) * D(u, F')$ where D(u, F') = $\min_{v \in F} D(u, v)$.

We will focus on the variant of the facility location problem with $F = P$ and with uniform costs and demands.

Problem Definition

(Metric) Minimum Facility Location Problem

We are given a metric (P,D), and a subset $F \subseteq P$ of facilities. For each facility $v \in F$, we are given a non-negative cost f(v), and for each point $u \in P$, a nonnegative demand d(u). The problem consists of finding a set $F' \subseteq F$, so as to minimize $I = \sum_{v \in F'} f(v) + \sum_{u \in P} d(u) * D(u, F')$ where D(u, F') $= \min_{v \in F} D(u, v)$.

We will focus on the variant of the facility location problem with $F = P$ and with uniform costs and demands.

- That is, $\forall v \in F$, $f(v) = c$ for some $c \geq 0$, and $\forall u \in P$, $d(u) = 1$.

Problem Definition

(Metric) Minimum Facility Location Problem

We are given a metric (P,D), and a subset $F \subseteq P$ of facilities. For each facility $v \in F$, we are given a non-negative cost f(v), and for each point $u \in P$, a nonnegative demand d(u). The problem consists of finding a set $F' \subseteq F$, so as to minimize I= $\sum_{v \in F'} f(v) + \sum_{u \in P} d(u) * D(u, F')$ where D(u, F') = $\min_{v \in F} D(u, v)$.

We will focus on the variant of the facility location problem with $F = P$ and with uniform costs and demands.

- That is,$\forall v \in F$, $f(v) = c$ for some $c \geq 0$, and $\forall u \in P$, $d(u) = 1$.
- We can assume that $c = 1$, if we re-scale the given metric.

Problem Definition

(Metric) Minimum Facility Location Problem

We are given a metric (P,D), and a subset $F \subseteq P$ of facilities. For each facility $v \in F$, we are given a non-negative cost $f(v)$, and for each point $u \in P$, a nonnegative demand $d(u)$. The problem consists of finding a set $F' \subseteq F$, so as to minimize I= $\sum_{v \in F'} f(v) + \sum_{u \in P} d(u) * D(u, F')$ where D(u, F') $= \min_{v \in F} D(u, v)$.

We will focus on the variant of the facility location problem with $F = P$ and with uniform costs and demands.

- That is, $\forall v \in F$, $f(v) = c$ for some $c \geq 0$, and $\forall u \in P$, $d(u) = 1$.
- We can assume that $c = 1$, if we re-scale the given metric.
- Thus, I= $\min_{F' \subseteq P} \{|F'| + \sum_{u \in P} D(u, F')\}$

Preliminaries

- Let (P,D) be a metric with a point set $P = p_1, ..., p_n$. For any point $p_i \in P$, and $\forall r \geq 0$, we denote by $B(p_i, r)$ the set of points in P which are at distance at most r from p_i. For each i, $1 \leq i \leq n$, let $r_i > 0$ be the number satisfying $\sum_{p \in B(p_i, r_i)} (r_i - D(p_i, p)) = 1$.

Preliminaries

- Let (P,D) be a metric with a point set $P = p_1, ..., p_n$. For any point $p_i \in P$, and $\forall r \geq 0$, we denote by $B(p_i, r)$ the set of points in P which are at distance at most r from p_i. For each i, $1 \leq i \leq n$, let $r_i > 0$ be the number satisfying $$\sum_{p \in B(p_i, r_i)} (r_i - D(p_i, p)) = 1.$$
- We can easily see that $\forall i : 1 \leq i \leq n$, we have $\frac{1}{n} \leq r_i \leq 1$

Lemma 1

For every i, with $1 \leq i \leq n$, we have $\frac{1}{|B(p_i,r_i)|} \leq r_i \leq \frac{2}{|B(p_i,r_i/2)|}$.

Lemma 1

For every i, with $1 \leq i \leq n$, we have $\frac{1}{|B(p_i,r_i)|} \leq r_i \leq \frac{2}{|B(p_i,r_i/2)|}$.

Proof

By the definition of r_i, we have $\sum_{p \in B(p_i,r_i)} (r_i - D(p_i, p)) = 1$, which implies $\sum_{p \in B(p_i,r_i)} r_i \geq 1$, and thus $r_i \geq 1/|B(p_i, r_i)|$. The other inequality follows directly from the following: $1 = \sum_{p \in B(p_i,r_i)} (r_i - D(p_i, p)) \geq \sum_{p \in B(p_i,r_i/2)} (r_i - D(p_i, p)) \geq |B(p_i, r_i/2)| * r_i/2$.

MP Algorithm

1. Compute the value of r_i for every $p_i \in P$.
2. Sort the input such that $r_1 \leq r_2 \leq \ldots \leq r_n$.
3. For $i = 1$ to n: if there is no open facility in $B(p_i, 2r_i)$ then open the facility at p_i.

MP Algorithm

1. Compute the value of r_i for every $p_i \in P$.
2. Sort the input such that $r_1 \leq r_2 \leq ... \leq r_n$.
3. For $i = 1$ to n: if there is no open facility in $B(p_i, 2r_i)$ then open the facility at p_i.

This simple algorithm will return a set of open facilities for which the total cost is at most 3 times the minimum.

Cost Estimation

Lemma 2

$$\frac{1}{4}C_{OPT} \leq 4\sum_{p_i \in P} r_i \leq 6C_{OPT}.$$

Cost Estimation

Lemma 2

$$\frac{1}{4}C_{OPT} \leq 4\sum_{p_i \in P} r_i \leq 6C_{OPT}.$$

Proof:

Cost Estimation

Lemma 2

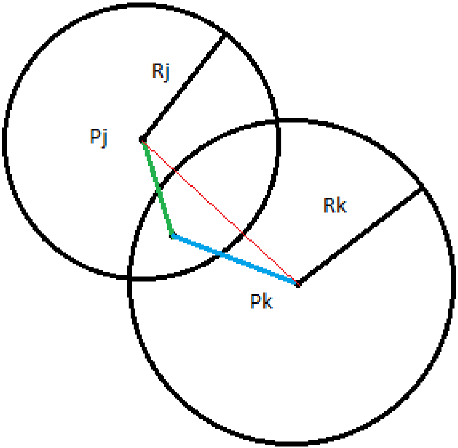
$$\frac{1}{4}C_{\mathrm{OPT}} \leq 4\sum_{p_i \in P} r_i \leq 6C_{\mathrm{OPT}}.$$

Proof:

Lower bound

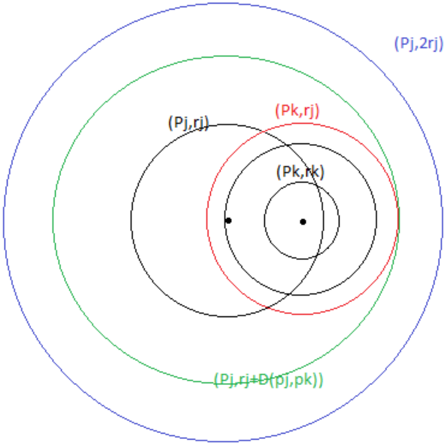
- Since in the MP algorithm for every $p_i \in P$ there is an open facility within distance at most $2\ r_i$ (for if not, then the algorithm would open the facility at p_i), we get that $2\sum_{p_i \in P} r_i \geq C^c_{MP}$.
- It remains to show that $\sum_{p_i \in P} r_i$ is an upper bound for C^f_{MP}.

- We first observe that every $p_i \in P$ is contained in at most one ball $B(p_j, r_j)$, for some $p_j \in F_{MP}$.



So, $\displaystyle\sum_{p_i \in P} r_i \geq \sum_{p_j \in F_{MP}} \sum_{p_k \in B(p_j, r_j)} r_k.$

So, $\sum_{p_i \in P} r_i \geq \sum_{p_j \in F_{MP}} \sum_{p_k \in B(p_j, r_j)} r_k$.

- Next, we observe that if $p_j \in F_{MP}$ and $p_k \in B(p_j, r_j)$, then we must have $r_j \leq 2r_k$.
- 

Lower bound

- $\sum_{p_i \in P} r_i \geq \sum_{p_j \in F_{MP}} \sum_{p_k \in B(p_j, r_j)} r_k \geq \sum_{p_j \in F_{MP}} \sum_{p_k \in B(p_j, r_j)} \frac{r_j}{2} = \frac{1}{2} \sum_{p_j \in F_{MP}} r_j |B(p_j, r_j)| \geq \frac{1}{2} \sum_{p_j \in F_{MP}} 1 = \frac{1}{2} C^f_{MP}$ (using Lemma 1)

Lower bound

- $$\sum_{p_i \in P} r_i \geq \sum_{p_j \in F_{MP}} \sum_{p_k \in B(p_j, r_j)} r_k \geq \sum_{p_j \in F_{MP}} \sum_{p_k \in B(p_j, r_j)} \frac{r_j}{2} = \frac{1}{2} \sum_{p_j \in F_{MP}} r_j |B(p_j, r_j)| \geq \frac{1}{2} \sum_{p_j \in F_{MP}} 1 = \frac{1}{2} C^f_{MP} \text{ (using Lemma 1)}$$
- Thus, we have: $2 * \sum_{p_i \in P} r_i \geq \frac{C^c_{MP}}{2} + \frac{C^f_{MP}}{2} \geq \frac{C_{MP}}{2} \geq \frac{C_{OPT}}{2}$

Estimating r_i in time $O(r_i n \log n)$

There is a constant factor approximation algorithm (randomized with high probability) of the above complexity.

Estimating r_i in time $O(r_i n \log n)$

There is a constant factor approximation algorithm (randomized with high probability) of the above complexity.

Lemma 3

Let j_0 be the maximum integer j, with $1 \leq j \leq \log n$, such that $|B(p_i, 2^{-j})| \geq 2^j$. Then, we have $2^{-(j0+1)} \leq r_i \leq 2^{-j0+1}$.

Estimating r_i in time $O(r_i n \log n)$

There is a constant factor approximation algorithm (randomized with high probability) of the above complexity.

Lemma 3

Let j_0 be the maximum integer j, with $1 \leq j \leq \log n$, such that $|B(p_i, 2^{-j})| \geq 2^j$. Then, we have $2^{-(j0+1)} \leq r_i \leq 2^{-j0+1}$.

Proof

Use Lemma 1...

Estimating r_i in time $O(r_i n\log n)$

There is a constant factor approximation algorithm (randomized with high probability) of the above complexity.

Lemma 3

Let j_0 be the maximum integer j, with $1 \le j \le \log n$, such that $|B(p_i, 2^{-j})| \ge 2^j$. Then, we have $2^{-(j0+1)} \le r_i \le 2^{-j0+1}$.

Proof

Use Lemma 1...

Algorithm

Our algorithm to estimate j_0 runs as follows:

- Set $j=\log n$.
- Decrease j by one until for the first time $:|B(p_i, 2^{-j})| \ge 2^j$

Approximation of $|B(p_i, 2^{-j})|$

- At each step, we pick uniformly at random, and with replacement, $K_j = c2^{-j} n \log n$ sample points
- Let N_j be the number of sample points that are inside the ball $B(p_i, 2^{-j})$.
- Return $\beta_j = nN_j/K_j$ as the estimator of $|B(p_i, 2^{-j})|$.

Quality of the estimator

Lemma 4

If $j \geq j_0 + 2$, then $\Pr[\beta_j \geq 2^j] < 1/\mathrm{poly}(n)$.

Quality of the estimator

Lemma 4

If $j \geq j_0 + 2$, then $\Pr[\beta_j \geq 2^j] < 1/\mathrm{poly}(n)$.

Proof

Since $j \geq j_0 + 2$, it follows that $B(p_i, 2^{-j}) \subseteq B(p_i, 2^{-(j0+1)})$. Let q be the probability that a randomly chosen sample point is in $B(p_i, 2^{-j})$. We have $q \leq |B(p_i, 2^{-(j0+1)})|/n$. By the choice of j_0, we have $|B(p_i, 2^{-(j0+1)})| < 2^{j_0+1}$, and thus $q < 2^{j0+1}/n \leq 2^{j-1}/n$. The expected number of sample points that fall inside $B(p_i, 2^{-j})$ is $E[N_j] = qK_j < \frac{c\log n}{2}$. Applying the Chernoff bound, we obtain $\Pr[\beta_j \geq 2^j] = \Pr[N_j \geq c\log n] < 1/\mathrm{poly}(n)$.

Quality of the estimator

Lemma 5

If $j \leq j_0 - 1$, then $\Pr[\beta_j \geq 2^j] > 1 - 1/\mathrm{poly}(n)$.

Quality of the estimator

Lemma 5

If $j \leq j_0 - 1$, then $\Pr[\beta_j \geq 2^j] > 1 - 1/\text{poly}(n)$.

Proof

Since $j \leq j_0 - 1$, it follows that $|B(p_i, 2^{-j})| \geq |B(p_i, 2^{-j_0})| \geq 2^{j_0} \geq 2^{j+1}$. We have that $q \geq 2^{j+1}/n$. The expected number of sample points that fall inside $B(p_i, 2^{-j})$ is $E[N_j] = qK_j \geq 2c\log n$. Applying the Chernoff bound, we obtain $\Pr[\beta_j \geq 2^j] = \Pr[N_j \geq c\log n] > 1 - 1/\text{poly}(n)$.

Lemma 6

The described procedure estimates the value of r_i to within a constant factor in time $O(r_i\, n \log n)$, with high probability.

Lemma 6

The described procedure estimates the value of r_i to within a constant factor in time $O(r_i\ n \log n)$, with high probability.

Proof

Let j_0' be the estimated value of j_0. By Lemmas 4 and 5, it follows that with high probability, $j_0 \leq j_0' \leq j_0 + 1$. If we use the value $r_i' = 2^{-j_0'}$ as an estimation of r_i, then by Lemma 3 we obtain that $r_i/4 \leq r_i \leq 2r_i$. Moreover, with high probability, the running time of the procedure is at most $\sum_{j=0}^{\log n} O(K_j) =$ $O(r_i n \log n)$.

Estimating the cost of the facility location problem

- To approximate the cost of the facility location problem it suffices to estimate the sum: $\sum_{i} r_i$ of the radii $r_1, ..., r_n$ of the points $p_1, ..., p_n$.

Estimating the cost of the facility location problem

- To approximate the cost of the facility location problem it suffices to estimate the sum: $\sum_i r_i$ of the radii $r_1, ..., r_n$ of the points $p_1, ..., p_n$.
- A standard approach to this problem would be to sample a set of s points (for a suitable s), determine (possibly approximately) their radii, and then output n times their average radius as an approximation for $\sum_i r_i$.

Estimating the cost of the facility location problem

- To approximate the cost of the facility location problem it suffices to estimate the sum: $\sum_i r_i$ of the radii $r_1, ..., r_n$ of the points $p_1, ..., p_n$.
- A standard approach to this problem would be to sample a set of s points (for a suitable s), determine (possibly approximately) their radii, and then output n times their average radius as an approximation for $\sum_i r_i$.
- But, in order to guarantee that we get a constant factor approximation we need to sample $s = \Omega(n)$ points.

Estimating the cost of the facility location problem

- To approximate the cost of the facility location problem it suffices to estimate the sum: $\sum_{i} r_i$ of the radii $r_1, ..., r_n$ of the points $p_1, ..., p_n$.
- A standard approach to this problem would be to sample a set of s points (for a suitable s), determine (possibly approximately) their radii, and then output n times their average radius as an approximation for $\sum_i r_i$.
- But, in order to guarantee that we get a constant factor approximation we need to sample $s = \Omega(n)$ points.
- This is due to the fact that the average radius can be as small as $1/n$.

So, we will use : Adaptive sampling:

So, we will use : Adaptive sampling:

- We start with a constant size sample of points and determine their average radius.

So, we will use : Adaptive sampling:

- We start with a constant size sample of points and determine their average radius.
- If our sample is too small we double it and continue until we have found a sample of sufficient size.

So, we will use : Adaptive sampling:

- We start with a constant size sample of points and determine their average radius.
- If our sample is too small we double it and continue until we have found a sample of sufficient size.
- For the analysis we will parameterize the sample size s by the average value of the r_i. Combining this with the running time of the adaptive algorithm leads to a sublinear algorithm.

Estimating the Sum of the Radii

- Let us first assume that we know the cost of the solution c, and we sample a set of s points independently and uniformly at random.

Estimating the Sum of the Radii

- Let us first assume that we know the cost of the solution c, and we sample a set of s points independently and uniformly at random.
- $\mathrm{s} = \Theta(\frac{\mathrm{n}}{\mathrm{c}}\mathrm{logn})$

Estimating the Sum of the Radii

- Let us first assume that we know the cost of the solution c, and we sample a set of s points independently and uniformly at random.
- $\mathrm{s} = \Theta(\frac{\mathrm{n}}{\mathrm{c}}\mathrm{logn})$
- Estimation of each $\mathrm{r_i}$: $\mathrm{O}(\mathrm{r_i n} * \mathrm{logn})$

Estimating the Sum of the Radii

- Let us first assume that we know the cost of the solution c, and we sample a set of s points independently and uniformly at random.
- $\mathrm{s} = \Theta(\frac{\mathrm{n}}{\mathrm{c}}\mathrm{logn})$
- Estimation of each $\mathrm{r_i}$: $\mathrm{O}(\mathrm{r_i n} * \mathrm{logn})$
- E[time] = s*E[one step] =s$*$ E[one step] = $\mathrm{sO}(\frac{1}{\mathrm{n}}\sum_{\mathrm{i}} \mathrm{r_i nlogn}) = \mathrm{O}(\mathrm{nlog^2 n})$.

Estimating the Sum of the Radii

- Let us first assume that we know the cost of the solution c, and we sample a set of s points independently and uniformly at random.
- $s = \Theta(\frac{n}{c} \log n)$
- Estimation of each r_i: $O(r_i n * \log n)$
- E[time] = s*E[one step] =s$*$ E[one step] = $sO(\frac{1}{n}\sum_i r_i n \log n) = O(n \log^2 n)$.
- Let x_i, for $i \in 1, 2, ..., s$, be the radii of the sample points taken by the algorithm.

Estimating the Sum of the Radii

- Let us first assume that we know the cost of the solution c, and we sample a set of s points independently and uniformly at random.
- $s = \Theta(\frac{n}{c} \log n)$
- Estimation of each r_i: $O(r_i n * \log n)$
- E[time] = s*E[one step] =s$*$ E[one step] = $sO(\frac{1}{n}\sum_i r_i n \log n) = O(n \log^2 n)$.
- Let x_i, for $i \in 1, 2, ..., s$, be the radii of the sample points taken by the algorithm.
- $E[X_i] = \frac{\sum_j r_j}{n}$

- Let $S = \sum_{i=1}^{s} x_i$ and hence,
  $E[S] = \frac{s * \sum_i r_i}{n} = \frac{\Theta(\frac{n}{c} \log n) \sum_i r_i}{n} = \Theta(\frac{\sum_i r_i}{c} * \log n) = \Theta(\log n)$.

- Let $S = \sum_{i=1}^{s} x_i$ and hence, $E[S] = \frac{s * \sum_i r_i}{n} = \frac{\Theta(\frac{n}{c} \log n) \sum_i r_i}{n} = \Theta(\frac{\sum_i r_i}{c} * \log n) = \Theta(\log n)$.
- S will be used as an estimator of $\frac{s}{n} \sum_i r_i$

- Let $S = \sum_{i=1}^{s} x_i$ and hence, $E[S] = \frac{s * \sum_i r_i}{n} = \frac{\Theta(\frac{n}{c} \log n) \sum_i r_i}{n} = \Theta(\frac{\sum_i r_i}{c} * \log n) = \Theta(\log n)$.
- S will be used as an estimator of $\frac{s}{n} \sum_i r_i$
- From Hoeffding inequality and $0 \le x_i \le 1$:
  - $\Pr[S \ge (1+\varepsilon)E[S]] \le e^{-\frac{\varepsilon^2 E[S]}{2(1+\varepsilon/3)}}$
  - $\Pr[S \ge (1-\varepsilon)E[S]] \le e^{-\frac{\varepsilon^2 E[S]}{2}}$
  - $\Pr[|S - E[S]| \ge \varepsilon E[S]] \le 2e^{-\Theta(\varepsilon^2 E[S])} - 2e^{-\Theta(\varepsilon^2 \log n)}$

Removing the assumption

In fact, we don't know the cost c before. So, we do adaptive sampling:

Removing the assumption

In fact, we don't know the cost c before. So, we do adaptive sampling:

Algorithm

- We start in the first phase by guessing $c = n$ (underestimation of the cost).
- If $S < \frac{s}{n}c$, then we start a new phase with estimated cost $c/2$, and so on.
- If $S \geq \frac{s}{n}c$, we return S*n/s as the approximation of the cost.

- The probability that the algorithm ends in a bad phase (when S far away from $\frac{s}{n} * c$ is low, because $\Pr[S \geq (1+\varepsilon) * E[S]] <$1/poly(n), as shown above.

- The probability that the algorithm ends in a bad phase (when S far away from $\frac{\mathrm{s}}{\mathrm{n}} * \mathrm{c}$ is low, because $\Pr[\mathrm{S} \geq (1+\varepsilon) * \mathrm{E}[\mathrm{S}]]$ <1/poly(n), as shown above.
- Since we need to have at least one facility in a solution, we have $\mathrm{c} \geq 1$, therefore we have at most a logarithmic number of phases.

- The probability that the algorithm ends in a bad phase (when S far away from $\frac{\mathrm{s}}{\mathrm{n}} * \mathrm{c}$ is low, because $\Pr[\mathrm{S} \geq (1+\varepsilon) * \mathrm{E}[\mathrm{S}]]$ <1/poly(n), as shown above.
- Since we need to have at least one facility in a solution, we have $\mathrm{c} \geq 1$, therefore we have at most a logarithmic number of phases.
- Note that we only get a constant slowdown by running these phases to guess c (at most 2 times the last phase).

- The probability that the algorithm ends in a bad phase (when S far away from $\frac{s}{n} * c$ is low, because $\Pr[S \geq (1+\varepsilon) * E[S]] <$1/poly(n), as shown above.
- Since we need to have at least one facility in a solution, we have $c \geq 1$, therefore we have at most a logarithmic number of phases.
- Note that we only get a constant slowdown by running these phases to guess c (at most 2 times the last phase).

Theorem

There exists a constant factor approximation algorithm for the uniform case of the Minimum Facility Location problem which runs in time $O(n\log^2 n)$ with high probability.

Lower bounds

Estimating the Cost in the General Case of the Uniform Minimum Facility Location Problem Requires $\Omega(n_2)$ Time (Even for Randomized Algorithms).

Lower bounds

Estimating the Cost in the General Case of the Uniform Minimum Facility Location Problem Requires $\Omega(n_2)$ Time (Even for Randomized Algorithms).

- Suppose that we relax the restriction: F=P.

Lower bounds

Estimating the Cost in the General Case of the Uniform Minimum Facility Location Problem Requires $\Omega(n_2)$ Time (Even for Randomized Algorithms).

- Suppose that we relax the restriction: F=P.

Theorem 2

For any $\varrho \geq 1$, every approximation algorithm (even a randomized one) with approximation ratio ϱ for the cost of the Minimum Facility Location problem as defined above requires time $\Omega(n^2)$.

Proof

We show the existence of two instances of the metric spaces which are undistinguishable by any o(n^2)-time algorithms and such that the cost of the Minimum Facility Location in one instance is greater ϱ times than the one in the other instance (for every ϱ).

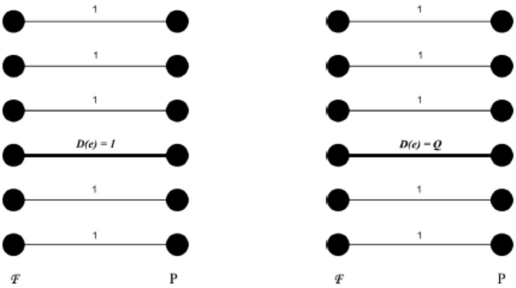


Fig. 1. Two metric spaces undistinguishable by any $o(n^2)$-time algorithms whose costs of the Minimum Facility Location differ by factor ϱ. The perfect matching connecting $\mathcal{F}$ with P is selected at random and the edge e is selected as a random edge from the matching. We set $Q = 2\,n\,(\varrho - 1) + 2$. The distances not shown are all equal to $n^3\,\varrho$