Approximation algorithms III

September 29, 2003

1 Previous Lecture

1. Art Gallery Problem.

2. Set Covering Problem - a $O(\log n)$ approximation.

3. TSP with the triangle inequality - 1.5 approximation.

4. Mumbled about clustering.

2 Clustering

Problem: A set $P$ of $n$ points in the plane, find $k$ smallest discs centered at input points, such that they cover all the points of $P$.

For $k = 3$ a possible solution:
$r$ - radius of clustering.

**Q:** How to compute this optimal cover by $k$ discs of minimum radius?

Known as $k$-center problem.

**Bad news:** $k$-center clustering is $NP$-Complete.

**Good news:** There is a simple 2-approximation algorithm.

Namely, one can compute in polynomial time, $k$ discs of radius at most twice the optimal radius.

Formal definition:

**$k$-center clustering**

Input: $P$ a set of $n$ points

- $d(p, q)$ - distance function with triangle inequality.
- $k$ - a parameter

Output: A subset $S$ that realizes:

$$r_{opt}(P, k) = \min_{S \subseteq P, |S| = k} d(S, P)$$

where $d(S, P) = \max_{x \in X} d(S, x)$ and $d(S, x) = \min_{s \in S} d(s, x)$.

2.1 The approximation algorithm

Idea: Let assume we have a solution for $m = 3$ centers, pick the $m + 1$ center to be...?

A:???

Always pick the bottleneck point, which is furthest away for the current set of centers.
Algorithm AprxKCenter($P$, $k$)

$P = \{p_1, \ldots, p_n\}$

$S = \{p_1\}$, $u_0 \leftarrow p_1$, $i \leftarrow 1$

For $j = 1 \ldots n$ do $d_j \leftarrow d(p_j, u_0)$

While $|S| \leq k$ do

For $j = 1 \ldots n$ do $d_j \leftarrow \min(d_j, d(p_j, u_{i-1}))$

$r_i \leftarrow \max(d_1, \ldots, d_n)$

$u_i \leftarrow \text{point of } P \text{ realizing } r_i$

$S \leftarrow S \cup \{u_i\}$

$i \leftarrow i + 1$

return $S$

$r_{i+1}$ - radius of $i$ discs centered at $u_1, \ldots, u_i$ that covers $P$. (The running time of this algorithm is $O(kn)$. Why?)

\textbf{Lemma 2.1} $r_1 \geq r_2 \geq \ldots \geq r_k$.

\textit{Proof:} We add new centers, and as such the distance to the closest center can not increase.

\hfill \blacksquare

\textbf{Observation 2.2} $r_{k+1}$ is the radius of the clustering generated by AprxKCenter.

\textbf{Lemma 2.3} $r_{k+1} \leq 2r_{\text{opt}}(P, k)$

\textit{Proof:} Consider the $k$ discs forming the optimal solution: $D_1, \ldots, D_k$ and consider the $k$ center points of $S$.

If every disk $D_i$ contain at least one point of $S$ we are done, since every point of $P$ is in distance at most $2r_{\text{opt}}(P, k)$ from one of the points of $S$.

$q$ center of disk in optimal solution. We have $d(p, u_7) \leq d(p, q) + d(q, u_7) \leq 2r_{\text{opt}}$.

Otherwise, there must be two points of $S$ in the same disk $D_i$ in the optimal solution.
But then, the distance between them is at least $r_{k+1}$ as the distance between them is at least the radius of the clustering when they were added, and arguing as above, we have that $r_{k+1} \leq 2r_{opt}$.

2.1.1 $k$-center for points in the plane.

The $k$-center clustering problem is $NP$-hard even for points in the plane, and the regular euclidean distance.

It is hard to approximate within a factor of 1.8 even in this easier settings.

One can do 2-approximation in $\Theta(n \log k)$ time ([Feder, Greene, 88]). In fact, it can be solve in linear time ([Clustering motion, Har-Peled, 01])

3 Subset Sum

Subset Sum

$X = \{x_1, \ldots, x_n\}$ - a set of $n$ integer positive numbers
$t$ - target number
Q: Is there a subset $S \subseteq X$ s.t. $\sum_{s \in S} s = t$?

Subset Sum is (of course) NP-Complete.

It can be solved in polynomial time if the $x_i$ are small.
Let $x_i \leq M$ for all $i$.
Then $t \leq Mn$.

SolveSubsetSum($X, t, M$)

$b[0 \ldots Mn]$ - boolean array initialized to FALSE.

// Where $b[i]$ is true if it can be achieved by a subset of $X$.

$b[0] \leftarrow True.$

For $i = 1 \ldots n$ do

For $j = Mn$ down to $x_i$ do

$b[j] \leftarrow B[j - x_i] \lor B[j]$ 
return $B[t]$.

Running time: $O(Mn^2)$

Q: What to do if want to solve this problem faster?
A: First, we turn the problem into an optimization problem.
Definition 3.1 (Subset Sum Optimization Ver.) For an instance of Subset Sum \((X, t)\), the optimal solution, denoted by \(opt\) is the largest number one can get as a subset sum which is smaller or equal to \(t\).

Find a subset of \(X\) such that it sum is smaller than \(t\) but very close to \(t\).

Formally,

\[
\text{Approx Subset Sum} \\
X = \{x_1, \ldots, x_n\} - \text{a set of } n \text{ integer positive numbers} \\
t - \text{target number} \\
\epsilon - \text{approximation parameter} \\
Q: \text{Find a subset } S \subseteq X \text{ s.t. } (1 - \epsilon)t \leq \sum_{s \in S} s \leq t, \text{ assuming that there is a subset that sums to } t.
\]

Q: How to solve this efficiently?

Lemma 3.2 If there is a subset sum that adds up to \(t\) one can find a subset sum that adds up to at least \(t/2\) in \(O(n \log n)\) time.

Proof: Add the numbers from largest to smallest. Clearly, we must reach a sum which is at least \(t/2\).

\[\blacksquare\]

4 An \(\epsilon\)-approximation algorithms

Definition 4.1 For a maximization problem PROB, an algorithm \(A(I, \epsilon)\) (i.e., \(A\) receives as input an instance of PROB, and an approximation parameter \(\epsilon > 0\)) is a polynomial time approximation scheme (PTAS) if for any instance \(I\) we have \((1 - \epsilon)|Opt(I)| \leq |A(I, \epsilon)| \leq |Opt(I)|\), where \(|Opt(I)|\) denote the price of the optimal solution for \(I\), and \(|A(I, \epsilon)|\) denotes the price of the solution outputted by \(A\). Furthermore, the running time of the algorithm is polynomial in \(n\) (the input size).

For a minimization problem, the condition is that \(|Opt(I)| \leq |A(I, \epsilon)| \leq (1 + \epsilon)|Opt(I)|\).

Example 4.2 An approximation algorithm with running time \(O(n^{1/\epsilon})\) is a PTAS, while an algorithm with running time \(O(1/\epsilon^n)\) is not.

Definition 4.3 An approximation algorithm is fully polynomial time approximation scheme (FPTAS) if it is a PTAS, and its running time is polynomial both in \(n\) and \(1/\epsilon\).

Example 4.4 A PTAS with running time \(O(n^{1/\epsilon})\) is not a FPTAS, while a PTAS with running time \(O(n^2/\epsilon^3)\) is a FPTAS.
5 Approximating subset-sum

\[ S = \{a_1, \ldots, a_n\} \] and a number \( x \) let
\[ x + S = \{a_1 + x, a_2 + x, \ldots a_n + x\} \]

**Definition 5.1** For two positive real numbers \( z \leq y \), the number \( y \) is a \( \delta \)-approximation to \( z \) if
\[ \frac{y}{1 + \delta} \leq z \leq y. \]

If we want to approximate the exact subset sum, we can remove from the lists we create close numbers. Namely,

\[
\text{ApproxSubsetSum}(S, t) \\
\begin{align*}
n &\leftarrow |S| \\
L_0 &\leftarrow \{0\} \\
\text{For } i = 1 \ldots n &\text{ do} \\
L_i &\leftarrow L_{i-1} \cup (L_{i-1} + x_i) \\
\text{Remove from } L_i &\text{ all elements larger than } t \\
\text{Return largest element in } L_n
\end{align*}
\]

**Observation 5.2** If \( x \in E_i \) then there exists a number \( y \in L_i \) such that \( y \leq x \leq y(1 + \delta) \), where \( \delta = \epsilon/2n \).

**Theorem 5.3** \( \text{ApproxSubsetSum} \) returns a number \( u \leq t \), such that \( \text{opt}/(1+\epsilon) \leq u \leq \text{opt} \leq t \), where \( \text{opt} \) is the optimal solution. The running time is \( O((n^3/\epsilon) \ln n) \).

**Proof:** Let \( E_i \) be the list generated by the algorithm.

Let \( P_i \) be the list of numbers without any trimming (i.e., the set generated by the exact algorithm.).

One can show, using induction, that for any \( x \in P_i \) there exists \( y \in L_i \) such that
\[ y \leq x \leq (1 + \delta)^i y. \]
**Claim 5.4** For any $x \in P_i$ there exists $y \in L_i$ such that

$$y \leq x \leq (1 + \delta)^i y.$$ 

**Proof:** If $x \in P_1$ the claim follows by the observation above. Otherwise, if $x \in P_{i-1}$ the claim follows by induction as there is $y' \in L_{i-1}$ such that $y' \leq x \leq (1 + \delta)^{i-1} y'$, by the observation above it follows, that there exists $y \in L_i$ such that $y \leq y' \leq (1 + \delta)y$, namely

$$y \leq y' \leq x \leq (1 + \delta)^i y$$

as required.

Thus, $x \in P_i \setminus P_{i-1}$, then $x = \alpha + x_i$ where $\alpha \in P_{i-1}$ and thus, by induction, there exists $\alpha' \in L_{i-1}$ such that

$$\alpha' \leq \alpha \leq (1 + \delta)^{i-1} \alpha'$$

Thus, $\alpha' + x_i \in E_i$ and by the observation, there is a $x' \in L_i$ such that

$$x' \leq \alpha' + x_i \leq (1 + \delta) x'$$

Thus,

$$x' \leq \alpha' + x_i \leq \alpha + x \leq (1 + \delta)^{i-1} \alpha' + x_i \leq (1 + \delta)^{i-1} (\alpha' + x_i) \leq (1 + \delta)^i x'$$

which establish the claim.

Observe that $L_i \subseteq P_i$.

Consider the optimal solution $\text{Opt} \in P_n$. Clearly, there exists $z \in L_n$ such that $z \leq \text{opt} \leq (1 + \delta)^n z$. However,

$$(1 + \delta)^n = (1 + \epsilon/2n)^n \leq e^{\epsilon/2} \leq 1 + \epsilon,$$

since $1 + x \leq e^x$ for $x \geq 0$. (Similar useful inequality is $1 + x \geq e^{x/2}$ for $x \in [0, 1/2]$)

Thus, $\text{opt}/(1 + \epsilon) \leq z \leq \text{opt} \leq t$.

**Analyzing the Running time:**

We need to bound the length of the lists $L_i$.

Observation: in a trimmed list, for any $x$ there are no two numbers in the trimmed list between $x$ and $(1 + \delta)x$.

So after trimming, we have one number in each interval of the type $[(1 + \delta)^i, (1 + \delta)^{i+1}]$.

Thus, the number of elements in $L_i$ is at most

$$\ln_{1+\delta}(t + 2) = \frac{\ln(t + 2)}{\ln(1 + \delta)} = O\left(\frac{\ln t}{\delta}\right) = O\left(\frac{n}{\epsilon} \cdot \ln t \right).$$

because of Taylor expansion:

$$\ln(1 + x) = x - x^2/2 + x^3/3 + \ldots$$

However, the length of the input (if we count bits) is at least $\ln(t)$. Thus, the length of the lists is at most polynomial in $n$, $1/\epsilon$ and $\ln t$. 


Here is a more involved argument:

$L_{i-1} + x_i$ is a set of numbers between $x_i$ and $ix_i$ because $x_i$ larger than $x_1 \ldots x_{i-1}$. As such, the number of different values in this range, after trimming is at most

$$\log_{1+\delta} \frac{ix_i}{x_i} = O\left(\frac{\ln i}{\delta}\right) = O\left(\frac{\ln n}{\delta}\right)$$

Thus,

$$|L_i| \leq |L_{i-1}| + O((\ln n)/\delta).$$

Finally,

$$|L_i| \leq |L_{i-1}| + O\left(\frac{n \ln n}{\epsilon}\right).$$

Thus, the total running time of the algorithm is $O(|L_n|n) = O\left(\frac{n^3}{\epsilon} \ln n\right)$.

6 Bin Packing

Min Bin Packing.

$a_1 \ldots a_n$ - $n$ numbers in $[0, 1]$.

Q: What is the minimum number of unit bins do you need to use to store all the numbers in $S$?

Bin packing is NPC. Why?

A: ???

How to do approximation?

A: Do next fit. Namely, put number in current bin if that bin can contain it. If the current bin can not contain it, create a new bin and put the number in this bin. Clearly, we need at least

$$[A] \text{ bins where } A = \sum_{i=1}^{n} a_i$$

Every two consecutive bins contain numbers that add up to more than 1 (Why?). As such, the number of bins used is $2 \cdot [A]$. Thus, this is a 2 approximation.

A better strategy, is to sort the number from largest to smallest and insert them in this order, where in each stage, we scan all current bins, and see if can insert the current number into one of those bins. If we can not, we can create a new bin for this number.

**Theorem 6.1** Decreasing first fit is a 1.5-approximation to Min Bin Packing.