Approximation Algorithms

- for finding solutions to "hard" problems in polynomial time (NP-complete problems)
- provide "near optimal" solutions
- convert problem to an optimization problem
- place bound on approximate solution

Let $C^* = \text{optimal soln.}$

$C = \text{approx. soln.}$

- if minimize $\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$
  then approximation algorithm achieves approximation ratio $\rho(n)$ and is called
  $\rho(n)$-approximation algorithm

- PTAS (Polynomial time approximation scheme)
  Provide $\epsilon$ to algorithm so that algorithm will be a
  $(1+\epsilon)$-approximation algorithm
Complexity Classes

$P$: class of problems solvable in polynomial time $O(n^k)$ for constant $k$.

$NP$: class of problems verifiable in polynomial time.

Any problem in $P$ also in $NP$.

$P \subseteq NP$, but is $P$ proper subset of $NP$? Unknown.

$NP$-Complete: class of problems in $NP$ but as $(NPC)$ “hard” as any problem in $NP$. 
Reductions

If any NPC problem can be solved in polynomial time, then every problem in NP has polynomial time algorithm.

Showing problem B is NP-Complete:

1. Show B in NP

2. Show B is NP-Hard

1. Showing B in NP:
   - show that a solution to B can be verified in polynomial time

2. Showing B is NP-Hard
   - have problem A which is NP-Hard

   instance of A \[\xrightarrow{\text{reduction algorithm}}\] \text{instance of B}

First problem shown to be NP-Hard:

Circuit Satisfiability Problem
Problem: Vertex Cover Problem

Un directed graph \( G = (V, E) \)

Vertex cover: Subset \( V' \) of \( V \) such that
if \((u,v) = e \in E\), then \( u \in V' \) or \( v \in V' \).

Find vertex cover of minimum size.

Algorithm:

Set \( C = \emptyset \)

while \( E \neq \emptyset \)

- take any edge \( e \in E \), where \( e = (u,v) \)
- add \( u \) and \( v \) to \( C \)
- remove all edges from \( E \) incident on \( u \) or \( v \)

return \( C \)

Analysis

Let \( A \) = set of edges picked in loop of algorithm
\( C^* \) = optimal solution
\( C \) = approximated solution

Since every edge picked must odd at least vertex,
\(|C^*| \geq |A|\).

For each edge picked we add two vertices,
\(|C| = 2 \cdot |A|\)

\(\therefore 2 \cdot |C^*| \geq |C|\)

and we have 2-approximation algorithm.
Partition

Problem:

Set $S$ of $n$ items with weights $S_1, \ldots, S_n$. Partition $S$ into sets $A$ and $B$ and minimize

$$\max \left( \sum_{i \in A} S_i, \sum_{i \in B} S_i \right)$$

Sum of all defined as:

$$\sum_{i=1}^{n} S_i = w(S) = 2L$$

We know optimal solution $C \geq L$

Using PTAS, we choose $\varepsilon$, and choose $m$ such that

$$\varepsilon \leq \frac{1}{m+1}$$

Assume W.L.O.G. that

$$S_1 \geq S_2 \ldots \geq S_n$$
Algorithm:

1st phase: Find optimal partition $A', B'$ of $S$

2nd phase: $A \leftarrow A'$

$B \leftarrow B'$

for $i = m + 1$ to $n$:

if $w(A) \leq w(B)$

$A = A \cup \{i\}$

else

$B = B \cup \{i\}$
Analysis:

We know that approximation ratio is \( \frac{W(A)}{L} \).

Let us consider last item added to \( A \) to \( S_k \), which is \( k^{th} \) element in \( S \).

We have two cases:

Case 1: \( k \) was added to \( A \) in Phase 1.

- We know that \( A^1 \) is optimal for \( \{S_1, \ldots, S_m\} \).
- We know that \( S_{m+1}, \ldots, S_n \) were all added to \( B \) in Phase 2.

- \( A^1 = A = \) optimal

Case 2: \( k \) was added to \( A \) in Phase 2:

- We know that
  \[ W(A) - S_k \leq W(B) \]
  \[ W(A) - S_k \leq 2L - W(A) \]
  \[ W(A) \leq L + \frac{S_k}{2} \]

Since we assumed \( S_1 > S_2 \ldots S_n \), we know

\[ 2L \geq (m+1)S_k \Rightarrow \frac{S_k}{2L} \leq \frac{1}{m+1} \]

We have:

\[ \text{approx. ratio} = \frac{W(A)}{L} \leq \frac{L + \frac{S_k}{2L}}{L} \leq 1 + \frac{\frac{S_k}{2L}}{m+1} \approx 1 + \varepsilon \]

We thereby show that the approx. algorithm is an \((1 + \varepsilon)\) - approximation algorithm.
Problem: Let $S = \{S_1, \ldots, S_n\}$ and $t$

where $t$ is an integer.

Find subset $S' \subseteq S$ such that

\[ \sum_{i=1}^{n} S'_i = t \]

Algorithm:

\[ n = |S| \]
\[ L_0 = \langle 0 \rangle \]

for $i = 1$ to $n$:

\[ L_i = \text{merge-list} \left( L_{i-1}, L_{i-1} + x_i \right) \]
\[ L_i = \text{trim} \left( L_i, \frac{t}{2n} \right) \]

remove for $L_i$ every element greater than $t$

let $z^*$ be largest value in $L_n$

return $z^*$

Analysis:

We have a $(1+\varepsilon)$-approx. algorithm.
Traveling Salesman Problem

Problem:
Undirected Graph $G(V, E)$
with cost $c(u, v)$ for each edge $(u, v) \in E$

$\Rightarrow$ find Hamiltonian tree of $G$ with minimum cost

Algorithm:
Select a vertex $v \in V$
compute MST $T$
$L = \text{list of vertices in preorder walk of } T$
return Ham. cycle that visits $V$ in order $L$

Analysis
We assume triangle inequality.
We have a 2-approx. algorithm.