Sublinear Algorithms
Lecture 2

Sofya Raskhodnikova
Penn State University

Thanks to Madhav Jha (Penn State) for help with creating these slides.

Tentative Plan

Lecture 1. Background. Testing properties of images and lists.

Lecture 2. Testing properties of lists. Sublinear-time approximation for graph problems.

Lecture 3. Properties of functions. Monotonicity and linearity testing.

Lecture 4. Techniques for proving hardness. Other models for sublinear computation.

Property Testing

Simple Examples

Testing if a List is Sorted

Input: a list of n numbers $x_1, x_2, \ldots, x_n$

- Question: Is the list sorted?

  Requires reading entire list: $\Omega(n)$ time

- Approximate version: Is the list sorted or ϵ-far from sorted?

  (An ϵ fraction of x_i's have to be changed to make it sorted.)

  [Ergün Kannan Kumar Rubinfeld Viswanathan 98, Fischer 01]: $O((\log n)/\epsilon)$ time

  $\Omega(\log n)$ queries

- Attempts:

  1. Test: Pick a random i and reject if $x_i > x_{i+1}$.

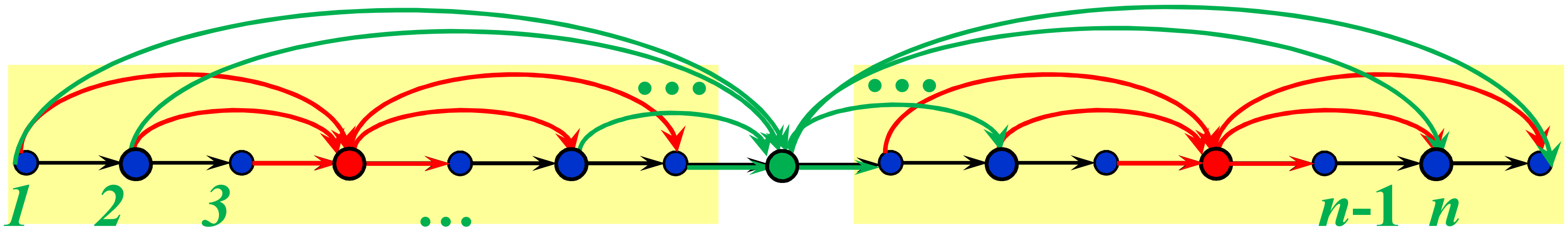
     Fails on: 1 1 1 1 1 1 1 0 0 0 0 0 0 0 $\leftarrow$ 1/2-far from sorted

  2. Test: Pick random $i < j$ and reject if $x_i > x_j$.

     Fails on: 1 0 2 1 3 2 4 3 5 4 6 5 7 6 $\leftarrow$ 1/2-far from sorted

Is a list sorted or ϵ-far from sorted?

Idea: Associate positions in the list with vertices of the directed line.



Construct a graph (2-spanner)

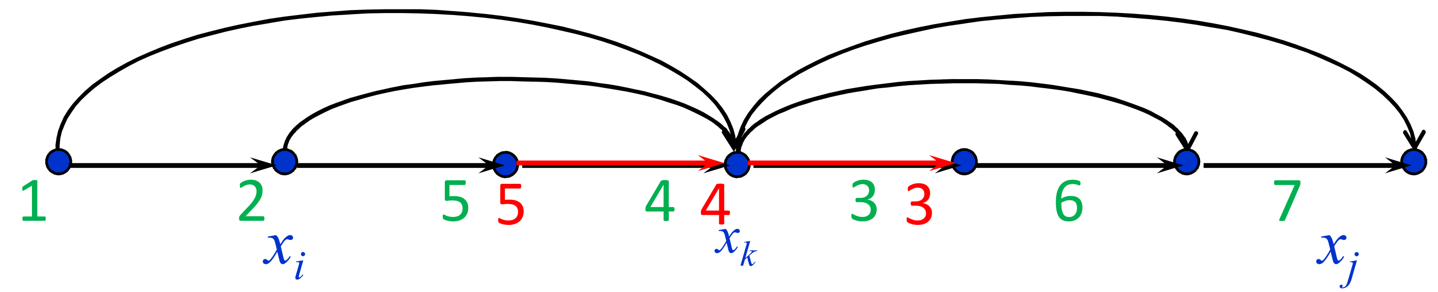
- by adding a few "shortcut" edges (i, j) for $i < j$
- where each pair of vertices is connected by a path of length at most 2

$\leq n \log n$ edges

Is a list sorted or ϵ-far from sorted?

Test [Dodis Goldreich Lehman Raskhodnikova Ron Samorodnitsky 99]

Pick a random edge (x_i, x_j) from the 2-spanner and reject if $x_i > x_j$.



Analysis:

- Call an edge (x_i, x_j) **violated** if $x_i > x_j$, and **good** otherwise.
- If x_i is an endpoint of a **bad** edge, call it **bad**. Otherwise, call it **good**.

Claim 1. All **good** numbers x_i are sorted.

Proof: Consider any two good numbers, x_i and x_j.

They are connected by a path of (at most) two **good** edges (x_i, x_k), (x_k, x_j).

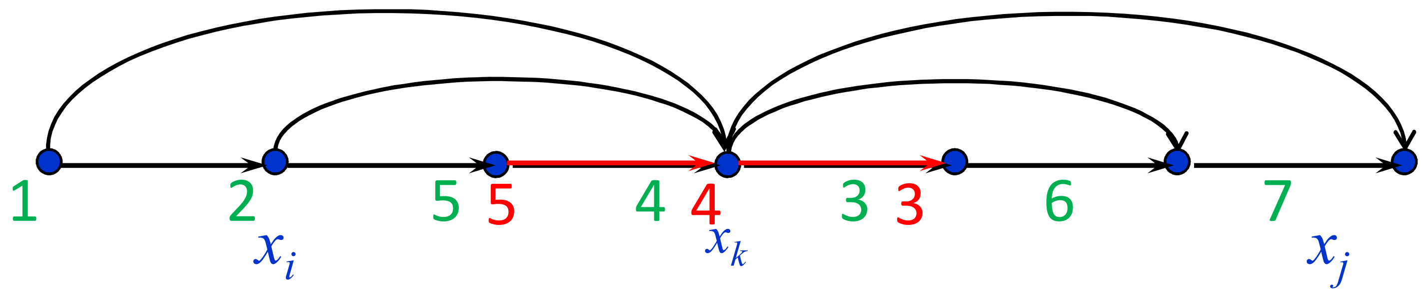
$\Rightarrow$ $x_i \leq x_k$ and $x_k \leq x_j$

$\Rightarrow$ $x_i \leq x_j$

Is a list sorted or ϵ-far from sorted?

Test [Dodis Goldreich Lehman Raskhodnikova Ron Samorodnitsky 99]

Pick a random edge (x_i, x_j) from the 2-spanner and reject if $x_i > x_j$.



Analysis:

- Call an edge (x_i, x_j) **violated** if $x_i > x_j$, and **good** otherwise.
- If x_i is an endpoint of a **bad** edge, call it **bad**. Otherwise, call it **good**.

Claim 1. All **good** numbers x_i are sorted.

Claim 2. An ϵ-far list **violates** $\geq \epsilon/(2 \log n)$ fraction of edges in 2-spanner.

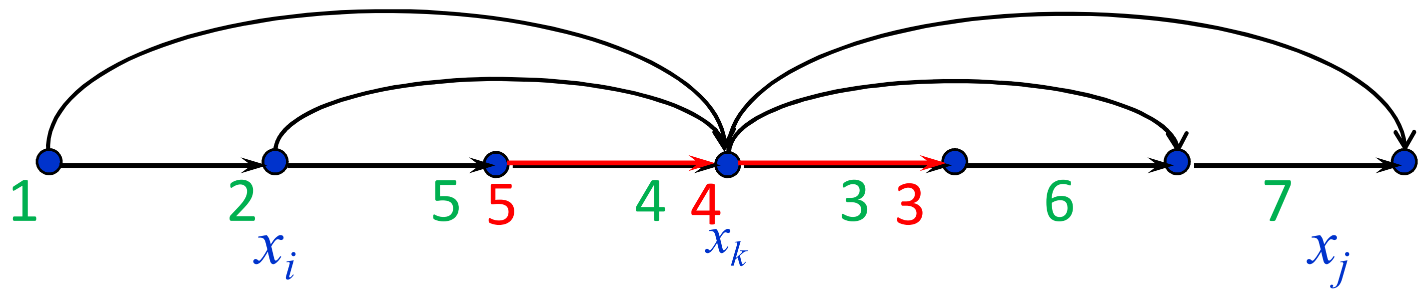
Proof: If a list is ϵ-far from sorted, it has $\geq \epsilon n$ **bad** numbers. (Claim 1)

$\Rightarrow$ 2-TC-spanner has $\geq \epsilon n/2$ **violated** edges out of $\leq n \log n$

Is a list sorted or ϵ-far from sorted?

Test [Dodis Goldreich Lehman Raskhodnikova Ron Samorodnitsky 99]

Pick a random edge (x_i, x_j) from the 2-spanner and **reject** if $x_i > x_j$.



Analysis:

- Call an edge (x_i, x_j) **violated** if $x_i > x_j$, and **good** otherwise.

Claim 2. An ϵ-far list **violates** $\geq \epsilon/(2 \log n)$ fraction of edges in 2-spanner.

By Witness Lemma, it suffices to sample $(4 \log n)/\epsilon$ edges from 2-spanner.

Algorithm

Sample $(4 \log n)/\epsilon$ edges (x_i, x_j) from the 2-spanner and **reject** if $x_i > x_j$.

Guarantee: All sorted lists are accepted.

All lists that are ϵ-far from sorted are rejected with probability $\geq 2/3$.

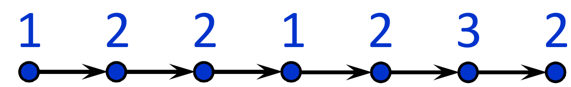
Time: $O((\log n)/\epsilon)$

Comparison to Binary-Search-Based Test

- Binary-Search-Based Test worked only for testing if a sequence is strictly increasing.
  - There is a simple reduction from testing strict sortedness to testing non-strict sortedness.
- Spanner-based test is nonadaptive: queries can be determined in advance, before seeing answers to previous queries.
  - Binary-Search-Based Test can be made nonadaptive.

Lipschitz Property

- A list of n numbers $x_1, x_2, \ldots, x_n$ is Lipschitz if the numbers do not change too quickly: $|x_i - x_{i-1}| \le 1$ for all i.



The spanner-based test for sortedness can test the Lipschitz property in $O(\log n / \varepsilon)$ time.

It applies to a more general class of properties.

Randomized Approximation in sublinear time

Simple Examples

Reminder: a Toy Example

0	0	0	1	…	0	1	0	0

Input: a string $w \in \{0,1\}^n$

Goal: Estimate the fraction of 1's in w (like in polls)

It suffices to sample $s = 1 / \varepsilon^2$ positions and output the average to get the fraction of 1's $\pm\varepsilon$ (i.e., additive error ε) with probability $\geq$ 2/3

Hoeffding Bound

Let $\mathrm{Y}_1, \ldots, \mathrm{Y}_s$ be independently distributed random variables in [0,1] and let $\mathrm{Y} = \sum_{i=1}^{s} \mathrm{Y}_\mathrm{i}$ (sample sum). Then $\Pr[|\mathrm{Y} - \mathrm{E}[\mathrm{Y}]| \geq \delta] \leq 2\mathrm{e}^{-2\delta^2/s}$.

Y_i = value of sample i. Then E[Y] $= \sum_{i=1}^{s} \mathrm{E}[\mathrm{Y}_\mathrm{i}] = s \cdot$ (fraction of 1's in w)

$$\Pr[|(\text{sample average}) - (\text{fraction of } 1'\text{s in } w)| \geq \varepsilon] = \Pr[|\mathrm{Y} - \mathrm{E}[\mathrm{Y}]| \geq \varepsilon s]$$
$$\leq 2\mathrm{e}^{-2\delta^2/s} = 2e^{-2} < 1/3$$

Apply Hoeffding Bound with $\delta = \varepsilon s$

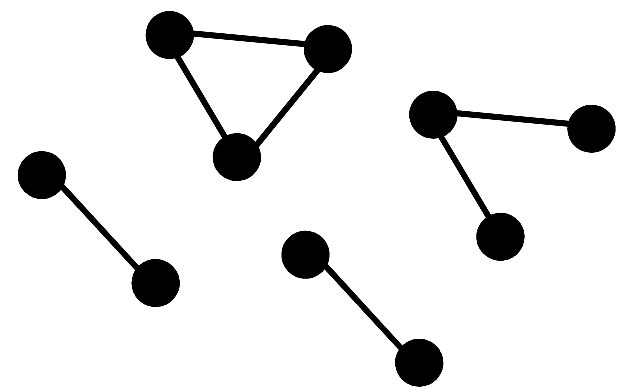
substitute $s = 1 / \varepsilon^2$

Approximating # of Connected Components

[Chazelle Rubinfeld Trevisan]

Input: a graph $G = (V, E)$ on n vertices

- in adjacency lists representation
  (a list of neighbors for each vertex)
- maximum degree *d*



Exact Answer: Ω(dn) time

Additive approximation: # of CC ±εn

with probability $\geq$ 2/3

Time:

- Known: $O\left(\frac{d}{\varepsilon^2}\log\frac{1}{\varepsilon}\right), \Omega\left(\frac{d}{\varepsilon^2}\right)$
- Today: $O\left(\frac{d}{\varepsilon^3}\cdot\log\frac{1}{\varepsilon}\right)$

No dependence on n!

Approximating # of CCs: Main Idea

- Let C = number of components
- For every vertex u, define
  n_u = number of nodes in *u's component*
  – for each component A: $\sum_{u \in A} \frac{1}{n_u} = 1$

$$\sum_{u \in V} \frac{1}{n_u} = C$$

Breaks C up into contributions of different nodes

- Estimate this sum by estimating n_u's for a few random nodes
  – If u's component is small, its size can be computed by BFS.
  – If u's component is big, then $1/n_u$ is small, so it does not contribute much to the sum
  – Can stop BFS after a few steps

Similar to property tester for connectedness [Goldreich Ron]

Approximating # of CCs: Algorithm

Estimating n_u = the number of nodes in u's component:

- Let estimate $\hat{n}_u = \min\left\{n_u, \frac{2}{\varepsilon}\right\}$
  - When u's component has $\leq 2/\varepsilon$ nodes, $\hat{n}_u = n_u$
  - Else $\hat{n}_u = 2/\varepsilon$, and so $0 < \frac{1}{\hat{n}_u} - \frac{1}{n_u} < \frac{1}{\hat{n}_u} = \frac{\varepsilon}{2}$

$$\left|\frac{1}{\hat{n}_u} - \frac{1}{n_u}\right| \leq \frac{\varepsilon}{2}$$

- Corresponding estimate for C is $\hat{C} = \sum_{u \in V} \frac{1}{\hat{n}_u}$. It is a good estimate: $\left|\hat{C} - C\right| = \left|\sum_{u \in V} \frac{1}{\hat{n}_u} - \sum_{u \in V} \frac{1}{n_u}\right| \leq \sum_{u \in V} \left|\frac{1}{\hat{n}_u} - \frac{1}{n_u}\right| \leq \frac{\varepsilon n}{2}$

APPROX_#_CCs (G, d, ε)

1. **Repeat** s=Θ(1/ε²) times:
2. pick a random vertex u
3. compute $\hat{n}_u$ via BFS from u, storing all discovered nodes in a sorted list and stopping after at most 2/ε new nodes
4. **Return** $\tilde{C}$ = (average of the values $1/\hat{n}_u$) $\cdot\, n$

Run time: $O\left(\frac{d}{\varepsilon^3} \cdot \log \frac{1}{\varepsilon}\right)$

Approximating # of CCs: Analysis

Want to show: $\Pr\left[\left|\tilde{C} - \hat{C}\right| > \frac{\varepsilon n}{2}\right] \leq \frac{1}{3}$ ✓

Hoeffding Bound

Let $Y_1, \ldots, Y_s$ be independently distributed random variables in [0,1] and let $Y = \sum_{i=1}^{s} Y_i$ (sample sum). Then $\Pr[|Y - E[Y]| \geq \delta] \leq 2e^{-2\delta^2/s}$.

Let $Y_i = 1/\hat{n}_u$ for the i^{th} vertex u in the sample

- $Y = \sum_{i=1}^{s} Y_i = \frac{s\tilde{C}}{n}$ and $E[Y] = \sum_{i=1}^{s} E[Y_i] = s \cdot E[Y_1] = s \cdot \frac{1}{n} \sum_{u \in V} \frac{1}{\hat{n}_v} = \frac{s\hat{C}}{n}$

$$\Pr\left[\left|\tilde{C} - \hat{C}\right| > \frac{\varepsilon n}{2}\right] = \Pr\left[\left|\frac{n}{s} Y - \frac{n}{s} E[Y]\right| > \frac{\varepsilon n}{2}\right] = \Pr\left[|Y - E[Y]| > \frac{\varepsilon s}{2}\right] \leq 2e^{-\frac{\varepsilon^2 s}{2}}$$

- Need $s = \Theta\left(\frac{1}{\varepsilon^2}\right)$ samples to get probability $\leq \frac{1}{3}$

Approximating # of CCs: Analysis

So far: $\left|\hat{C} - C\right| \leq \frac{\varepsilon n}{2}$

$$\Pr\left[\left|\tilde{C} - \hat{C}\right| > \frac{\varepsilon n}{2}\right] \leq \frac{1}{3}$$

- With probability $\geq \frac{2}{3}$,

$$\left|\tilde{C} - C\right| \leq \left|\tilde{C} - \hat{C}\right| + \left|\hat{C} - C\right| \leq \frac{\varepsilon n}{2} + \frac{\varepsilon n}{2} \leq \varepsilon n$$

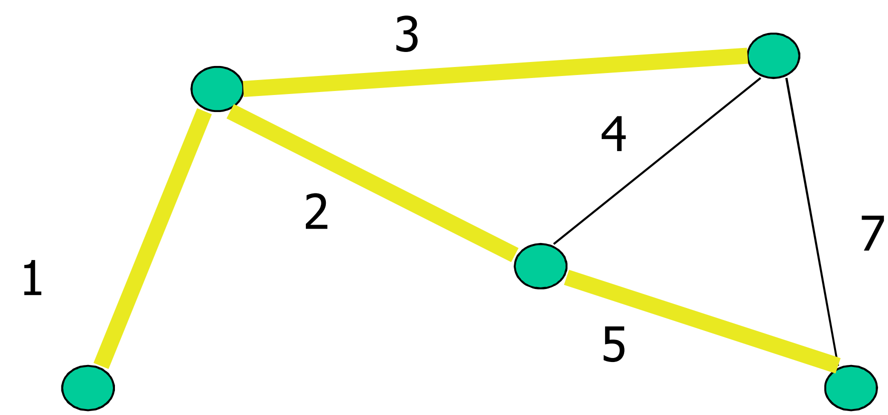
Summary:

The number of connected components in n-vertex graphs of degree at most d can be estimated within $\pm\varepsilon n$ in time $O\left(\frac{d}{\varepsilon^3} \cdot \log\frac{1}{\varepsilon}\right)$.

Minimum spanning tree (MST)

- What is the cheapest way to connect all the dots?

Input: a weighted graph
with n vertices and m edges



- Exact computation:
  - Deterministic $O(m \cdot \text{inverse-Ackermann}(m))$ time [Chazelle]
  - Randomized $O(m)$ time [Karger Klein Tarjan]

Partially based on slides by Ronitt Rubinfeld: http://stellar.mit.edu/S/course/6/fa10/6.896/courseMaterial/topics/topic3/lectureNotes/lecst11/lecst11.pdf

Approximating MST Weight in Sublinear Time

[Chazelle Rubinfeld Trevisan]

Input: a graph $G = (V, E)$ on n vertices

- in adjacency lists representation
- maximum degree *d* and maximum allowed weight *w*
- weights in {1,2,…,*w*}

Output: (1+ ε)-approximation to MST weight, w_{MST}

Number of queries:

- Known: $O\left(\frac{dw}{\varepsilon^3}\log\frac{dw}{\varepsilon}\right)$, $\Omega\left(\frac{dw}{\varepsilon^2}\right)$
- Today: $small\ polynomial\ in\ d, w, 1/\varepsilon$

No dependence on n!

Idea Behind Algorithm

- Characterize MST weight in terms of number of connected components in certain subgraphs of *G*

- Already know that number of connected components can be estimated quickly

MST and Connected Components: Warm-up

- Recall Kruskal's algorithm for computing MST exactly.

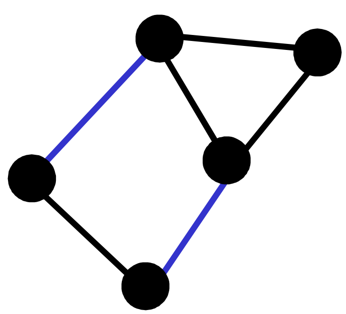
Suppose all weights are 1 or 2. Then MST weight = (# weight-1 edges in MST) + 2 · (# weight-2 edges in MST)

$= n - 1 + $ (# of weight-2 edges in MST)

MST has $n - 1$ edges

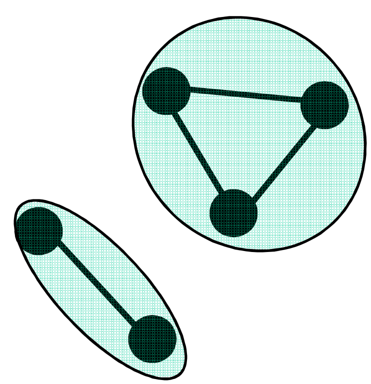
$= n - 1 + $ (# of CCs induced by weight-1 edges) $- 1$

By Kruskal

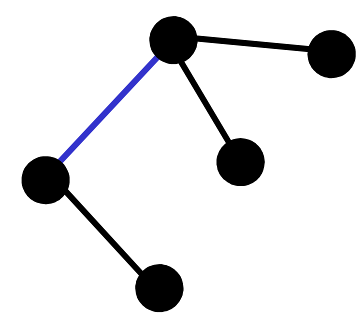


weight 1

weight 2



connected components
induced by weight-1 edges



MST

MST and Connected Components

In general: Let G_i = subgraph of G containing all edges of weight $\leq i$

C_i = number of connected components in G_i

Then MST has $C_i - 1$ edges of weight $> i$.

Claim

$$w_{MST}(G) = n - w + \sum_{i=1}^{w-1} C_i$$

✔

- Let β_i be the number of edges of weight $> i$ in MST
- Each MST edge contributes 1 to w_{MST}, each MST edge of weight >1 contributes 1 more, each MST edge of weight >2 contributes one more, …

$$w_{MST}(G) = \sum_{i=0}^{w-1} \beta_i = \sum_{i=0}^{w-1} (C_i - 1) = -w + \sum_{i=0}^{w-1} C_i = n - w + \sum_{i=1}^{w-1} C_i$$

Algorithm for Approximating w_{MST}

APPROX_MSTweight (G, w, d, ε)

1. **For** $i = 1$ to $w - 1$ **do**:
2. $\tilde{C}_i \leftarrow$APPROX_#CCs($G_i\,, d, \varepsilon/\mathrm{w}$).
3. **Return** $\widetilde{w}_{MST} = n - w + \sum_{i=1}^{w-1} \tilde{C}_i$.

Claim. $w_{MST}(G) = n - w + \sum_{i=1}^{w-1} C_i$

Analysis:

- Suppose all estimates of C_i's are good: $\left|\tilde{C}_i - C_i\right| \le \frac{\varepsilon}{w}\, n$.

  Then $|\widetilde{w}_{MST} - w_{MST}| = |\sum_{i=1}^{w-1}(\tilde{C}_i - C_i)| \le \sum_{i=1}^{w-1} |\tilde{C}_i - C_i| \le w \cdot \frac{\varepsilon}{w}\, n = \varepsilon n$
- Pr[all $w - 1$ estimates are good]$\ge (2/3)^{w-1}$
- Not good enough! Need error probability $\le \frac{1}{3w}$ for each iteration
- Then, by Union Bound, Pr[error]$\le w \cdot \frac{1}{3w} = \frac{1}{3}$

Can amplify success probability of any algorithm by repeating it and taking the median answer.

Can take more samples in APPROX_#CCs. What's the resulting run time?

Multiplicative Approximation for w_{MST}

For MST cost, additive approximation $\Longrightarrow$ multiplicative approximation

$$w_{MST} \geq n - 1 \quad \Longrightarrow \quad w_{MST} \geq n/2 \text{ for } n \geq 2$$

- εn-additive approximation:

$$w_{MST} - \varepsilon n \leq \hat{w}_{MST} \leq w_{MST} + \varepsilon n$$

- $(1 \pm 2\varepsilon)$-multiplicative approximation:

$$w_{MST}(1 - 2\varepsilon) \leq w_{MST} - \varepsilon n \leq \hat{w}_{MST} \leq w_{MST} + \varepsilon n \leq w_{MST}(1 + 2\varepsilon)$$