A Near-Optimal Sublinear-Time Algorithm for Approximating the Minimum Vertex Cover Size

Krzysztof Onak
CMU

Joint work with Dana Ron, Michal Rosen, and Ronitt Rubinfeld
The Problem

Vertex Cover: set $S$ of vertices such that each edge has endpoint in $S$. 

![Diagram of a graph with vertices covered by a vertex cover set](image)
The Problem

- **Vertex Cover:** set $S$ of vertices such that each edge has endpoint in $S$

- **Our Goal:** $(2, \varepsilon n)$-estimate for the minimum vertex cover size

- $X$ is an $(\alpha, \beta)$-estimate for $Y$ if

\[ Y \leq X \leq \alpha Y + \beta \]
The Model

Graph $G$ of degree $d$:

Query access to adjacency list of each node
Query Complexity

Positive results for \((2, \epsilon n)\)-estimation:

- Parnas, Ron (2007): \(d^{O(\log(d)/\epsilon^3)}\)
- Marko, Ron (2007): \(d^{O(\log(d/\epsilon))}\)
- Nguyen, O. (2008): \(2^{O(d)}/\epsilon^2\)
- Yoshida, Yamamoto, Ito (2009): \(O(d^4/\epsilon^2)\)
Query Complexity

Positive results for $(2, \epsilon n)$-estimation:

- Parnas, Ron (2007): $d^{O\left(\log(d)/\epsilon^3\right)}$
- Marko, Ron (2007): $d^{O\left(\log(d/\epsilon)\right)}$
- Nguyen, O. (2008): $2^{O(d)}/\epsilon^2$
- Yoshida, Yamamoto, Ito (2009): $O(d^4/\epsilon^2)$
- This work: $\tilde{O}(d/\epsilon^3)$
Query Complexity

Positive results for $(2, \epsilon n)$-estimation:

- Parnas, Ron (2007): $d^{O(\log(d)/\epsilon^3)}$
- Marko, Ron (2007): $d^{O(\log(d/\epsilon))}$
- Nguyen, O. (2008): $2^{O(d)}/\epsilon^2$
- Yoshida, Yamamoto, Ito (2009): $O(d^4/\epsilon^2)$
- This work: $\tilde{O}(d/\epsilon^3)$

A negative result due to Parnas and Ron (2007):

$(C, \epsilon n)$-estimation requires $\Omega(d)$ queries for any constant $C$
Quick Review
General Approach

Idea of Parnas and Ron (2007):

- If we had query access to a small vertex cover, we could approximate its size up to $\pm \epsilon n$ by sampling $O(1/\epsilon^2)$ vertices.
General Approach

Idea of Parnas and Ron (2007):

- If we had query access to a small vertex cover, we could approximate its size up to $\pm \epsilon n$ by sampling $O(1/\epsilon^2)$ vertices.
- Construct oracle that provides query access to a small vertex cover.
General Approach

Idea of Parnas and Ron (2007):

- If we had query access to a small vertex cover, we could approximate its size up to $\pm \epsilon n$ by sampling $O(1/\epsilon^2)$ vertices.
- Construct oracle that provides query access to a small vertex cover.
- Parnas and Ron’s construction: simulation of local distributed algorithms of Kuhn, Moscibroda, and Wattenhofer (2006).
Simulation of the Greedy Algorithm

Classical 2-approximation algorithm [Gavril, Yannakakis]:

- Greedily find a maximal matching $M$
- Output the set of nodes matched in $M$
Simulation of the Greedy Algorithm

Classical 2-approximation algorithm [Gavril, Yannakakis]:

- Greedily find a maximal matching $M$
- Output the set of nodes matched in $M$
Simulation of the Greedy Algorithm

Classical 2-approximation algorithm [Gavril, Yannakakis]:

- Greedily find a maximal matching $M$
- Output the set of nodes matched in $M$
Simulation of the Greedy Algorithm

Classical $2$-approximation algorithm [Gavril, Yannakakis]:
- Greedily find a maximal matching $M$
- Output the set of nodes matched in $M$

The idea from Nguyen and O. (2008):
- Construction of $M$: consider edges in random order
Simulation of the Greedy Algorithm

Classical 2-approximation algorithm [Gavril, Yannakakis]:

- Greedily find a maximal matching $M$
- Output the set of nodes matched in $M$

The idea from Nguyen and O. (2008):

- Construction of $M$: consider edges in random order
- (Try to) locally check if an edge belongs to $M$
Simulation of the Greedy Algorithm

Random order \equiv \text{random numbers } r(e) \text{ assigned to each edge}

Algorithm: To check if \( e \in M \)
- recursively check if adjacent edges \( g \) s.t. \( r(g) < r(e) \) are in \( M \)
- \( e \in M \iff \text{none in } M \)
Simulation of the Greedy Algorithm

Random order ≡ random numbers $r(e)$ assigned to each edge

Algorithm: To check if $e \in M$

- recursively check if adjacent edges $g$ s.t. $r(g) < r(e)$ are in $M$

- $e \in M \iff$ none in $M$
Simulation of the Greedy Algorithm

Random order $\equiv$ random numbers $r(e)$ assigned to each edge

Algorithm: To check if $e \in M$

- recursively check if adjacent edges $g$ s.t. $r(g) < r(e)$ are in $M$
- $e \in M \iff$ none in $M$
Simulation of the Greedy Algorithm

Random order $\equiv$ random numbers $r(e)$ assigned to each edge

Algorithm: To check if $e \in M$
- recursively check if adjacent edges $g$ s.t. $r(g) < r(e)$ are in $M$
- $e \in M \iff$ none in $M$
Simulation of the Greedy Algorithm

Random order ≡ random numbers $r(e)$ assigned to each edge

Algorithm: To check if $e \in M$
- recursively check if adjacent edges $g$
  s.t. $r(g) < r(e)$ are in $M$
- $e \in M \iff$ none in $M$
Simulation of the Greedy Algorithm

Random order \equiv random numbers \( r(e) \) assigned to each edge

**Algorithm:** To check if \( e \in M \)
- recursively check if adjacent edges \( g \)
  s.t. \( r(g) < r(e) \) are in \( M \)
- \( e \in M \) \( \iff \) none in \( M \)
Simulation of the Greedy Algorithm

Random order \equiv \text{random numbers } r(e) \text{ assigned to each edge}

Algorithm: To check if \( e \in M \)
- recursively check if adjacent edges \( g \)
  such that \( r(g) < r(e) \) are in \( M \)
- \( e \in M \iff \text{none in } M \)
Simulation of the Greedy Algorithm

Random order $\equiv$ random numbers $r(e)$ assigned to each edge

Algorithm: To check if $e \in M$
  - recursively check if adjacent edges $g$
    s.t. $r(g) < r(e)$ are in $M$
  - $e \in M \iff$ none in $M$
Simulation of the Greedy Algorithm

Random order $\equiv$ random numbers $r(e)$ assigned to each edge

Algorithm: To check if $e \in M$

- recursively check if adjacent edges $g$ s.t. $r(g) < r(e)$ are in $M$
- $e \in M \iff$ none in $M$

Krzysztof Onak – *Approximating the Vertex Cover Size Near-Optimally* – p. 8/24
Simulation of the Greedy Algorithm

Random order \equiv \text{random numbers } r(e) \text{ assigned to each edge}

Algorithm: To check if \( e \in M \)
- recursively check if adjacent edges \( g \) s.t. \( r(g) < r(e) \) are in \( M \)
- \( e \in M \) \( \iff \) none in \( M \)
Simulation of the Greedy Algorithm

Random order \equiv \text{random numbers } r(e) \text{ assigned to each edge}

Algorithm: To check if \( e \in M \)
- recursively check if adjacent edges \( g \) s.t. \( r(g) < r(e) \) are in \( M \)
- \( e \in M \iff \text{none in } M \)
Simulation of the Greedy Algorithm

Random order $\equiv$ random numbers $r(e)$ assigned to each edge

Algorithm: To check if $e \in M$

- recursively check if adjacent edges $g$ s.t. $r(g) < r(e)$ are in $M$
- $e \in M \iff$ none in $M$

Krzysztof Onak – *Approximating the Vertex Cover Size Near-Optimally* – p. 8/24
Simulation of the Greedy Algorithm

Random order $\equiv$ random numbers $r(e)$ assigned to each edge

Algorithm: To check if $e \in M$
- recursively check if adjacent edges $g$ s.t. $r(g) < r(e)$ are in $M$
- $e \in M \iff$ none in $M$
Complexity of the Simulation


For every edge, the expected number of recursive calls is $2^{O(d)}$. 
Complexity of the Simulation


For every edge, the expected number of recursive calls is $2^O(d)$

We also proposed the following heuristic:

- For every edge $e$ consider adjacent edges $g$ in increasing order of $r(g)$
- Once an adjacent edge in $M$ detected, no need for further recursive calls: $e \notin M$. 
Complexity of the Simulation

- **Nguyen, O. (2008):**
  
  For every edge, the expected number of recursive calls is $2^{O(d)}$

- **We also proposed the following heuristic:**
  
  For every edge $e$ consider adjacent edges $g$ in increasing order of $r(g)$
  
  Once an adjacent edge in $M$ detected, no need for further recursive calls: $e \notin M$.

- **Yoshida, Yamamoto, Ito (2009):**
  
  The expected number of recursive calls is $O(d)$ for a random edge
Our New Algorithm (Part 1)
Overview

What happens to three factors of $d$?
Overview

What happens to three factors of $d$?

1. Slight improvement in the analysis of Yoshida et al.
Overview

What happens to three factors of $d$?

1. Slight improvement in the analysis of Yoshida et al.

2. Better bound on the number of recursive calls in a specific version of the exploration method
Overview

What happens to three factors of $d$?

1. Slight improvement in the analysis of Yoshida et al.

2. Better bound on the number of recursive calls in a specific version of the exploration method

3. Technique for limiting the exploration of neighbor sets
Overview

What happens to three factors of $d$?

1. Slight improvement in the analysis of Yoshida et al.
2. Better bound on the number of recursive calls in a specific version of the exploration method
3. Technique for limiting the exploration of neighbor sets

In this talk:
- Item 2 in Part 1
- Item 3 in Part 2
Our Exploration Method

How it works
(determining whether a vertex $v$ is in the vertex cover):
Our Exploration Method

How it works (determining whether a vertex $v$ is in the vertex cover):

- Consider edges incident to $v$ in ascending order of their random numbers
Our Exploration Method

How it works (determining whether a vertex \( v \) is in the vertex cover):

- Consider edges incident to \( v \) in ascending order of their random numbers
- To determine whether an edge is in the maximal matching, use the previously described heuristic
Our Exploration Method

How it works
(determining whether a vertex $v$ is in the vertex cover):

- Consider edges incident to $v$ in ascending order of their random numbers
- To determine whether an edge is in the maximal matching, use the previously described heuristic

Our bound:

The expected number of visited edges for a random vertex is

$$O \left( \frac{\text{average}_\text{degree} \cdot \text{maximum}_\text{degree}}{\text{minimum}_\text{degree}} \right)$$
Analysis

We reuse ideas from the bound of Yoshida et al.
Analysis

- We reuse ideas from the bound of Yoshida et al.
- No clear reduction of our bound to their bound
Analysis

- We reuse ideas from the bound of Yoshida et al.
- No clear reduction of our bound to their bound
- Let $X_k(e)$ = number of oracle calls on $e$ over all rankings of edges when starting from an endpoint of the $k$-th edge in the ranking.
Analysis

- We reuse ideas from the bound of Yoshida et al.
- No clear reduction of our bound to their bound
- Let $X_k(e) = \#\text{oracle calls on } e \text{ over all rankings of edges when starting from an endpoint of the } k\text{-th edge in the ranking}$
- Using the idea of slight mutations of rankings, we show

$$X_{k+1}(e) - X_k(e) \leq (m - 2)! \cdot d$$
Analysis

- We reuse ideas from the bound of Yoshida et al.
- No clear reduction of our bound to their bound
- Let $X_k(e) = \#\text{oracle calls on } e \text{ over all rankings of edges when starting from an endpoint of the } k\text{-th edge in the ranking}$
- Using the idea of slight mutations of rankings, we show

$$X_{k+1}(e) - X_k(e) \leq (m - 2)! \cdot d$$

- This suffices to inductively obtain a sufficiently good upper-bound on $X_k(e)$
 Quadratic Algorithm

- Pick $O(1/\epsilon^2)$ random vertices and estimate the fraction in the matching
Quadratic Algorithm

- Pick $O(1/\epsilon^2)$ random vertices and estimate the fraction in the matching.

- If the graph is near-regular, 
  \[
  \frac{\text{maximum\_degree}}{\text{minimum\_degree}} = \text{poly}(1/\epsilon),
  \]
  the number of recursive calls is $O(d/\text{poly}(\epsilon))$. 

Krzysztof Onak – *Approximating the Vertex Cover Size Near-Optimally* – p. 14/24
Quadratic Algorithm

- Pick $O(1/\epsilon^2)$ random vertices and estimate the fraction in the matching.
- If the graph is near-regular, \[ \frac{\text{maximum\_degree}}{\text{minimum\_degree}} = \text{poly}(1/\epsilon), \]
  the number of recursive calls is $O(d/\text{poly}(\epsilon))$
- Non-regular graphs: can “regularize” on the fly
Quadratic Algorithm

- Pick $O(1/\epsilon^2)$ random vertices and estimate the fraction in the matching

- If the graph is near-regular,

  $$\frac{\text{maximum\_degree}}{\text{minimum\_degree}} = \text{poly}(1/\epsilon),$$

  the number of recursive calls is $O(d/\text{poly}(\epsilon))$

- Non-regular graphs: can “regularize” on the fly

- For each recursive call, the query complexity is bounded by $O(d)$
Quadratic Algorithm

- Pick $O(1/\epsilon^2)$ random vertices and estimate the fraction in the matching
- If the graph is near-regular,
  \[ \frac{\text{maximum\_degree}}{\text{minimum\_degree}} = \text{poly}(1/\epsilon), \]
  the number of recursive calls is $O(d/\text{poly}(\epsilon))$
- Non-regular graphs: can “regularize” on the fly
- For each recursive call, the query complexity is bounded by $O(d)$
- Total: $O(d^2/\text{poly}(\epsilon))$ queries
Our New Algorithm (Part 2)
Limiting the Exploration of Neighbor Sets

We always look at all adjacent $O(d^2 / \text{poly}(\epsilon))$ edges
Limiting the Exploration of Neighbor Sets

- We always look at all adjacent $O(d^2 / \text{poly}(\epsilon))$ edges

- **Hope:** To make recursive calls, only $O(d / \text{poly}(\epsilon))$ vertex labels are necessary
Limiting the Exploration of Neighbor Sets

- We always look at all adjacent $O(d^2 / \text{poly}(\epsilon))$ edges

- **Hope:** To make recursive calls, only $O(d / \text{poly}(\epsilon))$ vertex labels are necessary

- **Simplest attempt:**
  - For every vertex, assign random numbers to incident edges without looking at them
  - Query only the relevant edges with the lowest numbers
Limiting the Exploration of Neighbor Sets

- We always look at all adjacent $O(d^2 / \text{poly}(\epsilon))$ edges
- **Hope:** To make recursive calls, only $O(d / \text{poly}(\epsilon))$ vertex labels are necessary
- **Simplest attempt:**
  - For every vertex, assign random numbers to incident edges without looking at them
  - Query only the relevant edges with the lowest numbers
- **Problem:**
  - An edge can have different numbers assigned at the endpoints
  - This could result in an inconsistent execution of the algorithm
  - Hard to predict results
Our Approach

We introduce data structures $D[v]$ for each vertex $v$:

- $D[v]$ provides access to the list of edges adjacent to $v$, sorted according to their random numbers.
Our Approach

We introduce data structures $D[v]$ for each vertex $v$:

- $D[v]$ provides access to the list of edges adjacent to $v$, sorted according to their random numbers.

- For each edge $(u, w)$, $D[u]$ and $D[w]$ may communicate to fix the random number assigned to $(u, w)$. 
Our Approach

We introduce data structures $D[v]$ for each vertex $v$:

- $D[v]$ provides access to the list of edges adjacent to $v$, sorted according to their random numbers.
- For each edge $(u, w)$, $D[u]$ and $D[w]$ may communicate to fix the random number assigned to $(u, w)$.

How we implement this:

- Each $D[v]$ tries to discover only the necessary head of the list.
- We partition the range $[0, 1]$ into a logarithmic number of “layers”.
- The algorithm discovers edges in the next layer, only if need be.
Selecting a Random Number

- Partition \((0, 1]\) into \(\Theta(\log n)\) ranges:
  - \((0, 2^{-\log n}]\)
  - \((2^{-i}, 2^{-i+1}]\) for \(1 \leq i \leq \log n\)

\[
\begin{array}{cccccc}
\mathcal{I}_0 & \mathcal{I}_1 & \mathcal{I}_2 & \mathcal{I}_3 & \mathcal{I}_4 \\
0 & 1/16 & 1/8 & 1/4 & 1/2 & 1
\end{array}
\]
Selecting a Random Number

Partition \((0, 1]\) into \(\Theta(\log n)\) ranges:

- \((0, 2^{-\log n}]\)
- \((2^{-i}, 2^{-i+1}]\) for \(1 \leq i \leq \log n\)

\[
\begin{array}{cccccc}
\mathcal{I}_0 & \mathcal{I}_1 & \mathcal{I}_2 & \mathcal{I}_3 & \mathcal{I}_4 \\
0 & 1/16 & 1/8 & 1/4 & 1/2 & 1
\end{array}
\]

To assign a random number, consider ranges from left to right:

\[
\text{for } i = 0 \text{ to } k: \\
\text{with probability } \frac{|\mathcal{I}_i|}{\sum_{j=i}^{k} |\mathcal{I}_j|} \\
\text{return random number in } \mathcal{I}_i
\]
Reducing the Query Complexity

One vertex’s point of view:

- We use this process to assign random numbers
Reducing the Query Complexity

One vertex’s point of view:

- We use this process to assign random numbers
- Consecutive iterations of the loop need not be simulated all at once
Reducing the Query Complexity

One vertex’s point of view:

- We use this process to assign random numbers
- Consecutive iterations of the loop need not be simulated all at once
- Each $D[v]$ simulates this process for all edges incident to $v$
Reducing the Query Complexity

One vertex’s point of view:

- We use this process to assign random numbers

- Consecutive iterations of the loop need not be simulated all at once

- Each $D[v]$ simulates this process for all edges incident to $v$

- Each iteration of the loop simulated simultaneously for all incident edges
Reducing the Query Complexity

Extending to the entire graph:

The same iteration of the loop may be executed by both $u$ and $v$ for an edge $(u, v)$
Reducing the Query Complexity

Extending to the entire graph:

- The same iteration of the loop may be executed by both \( u \) and \( v \) for an edge \( (u, v) \)

- We make sure that the decision made in the first execution is in effect by making \( D[u] \) and \( D[v] \) talk to each other
Reducing the Query Complexity

How do we reduce the number of queries?

- For an edge \((u, v)\) as long as \(D[u]\) and \(D[v]\) don’t assign a specific number:
  - Their decisions are consistent
  - No need to communicate
  - No need to know each other
  - No need to make a query
Reducing the Query Complexity

How do we reduce the number of queries?

- For an edge \((u, v)\) as long as \(D[u]\) and \(D[v]\) don’t assign a specific number:
  - Their decisions are consistent
  - No need to communicate
  - No need to know each other
  - No need to make a query

- The number of queries approximately proportional to the number of recursive calls from an edge.
Reducing the Query Complexity

How do we reduce the number of queries?

- For an edge \((u, v)\) as long as \(D[u]\) and \(D[v]\) don’t assign a specific number:
  - Their decisions are consistent
  - No need to communicate
  - No need to know each other
  - No need to make a query

- The number of queries approximately proportional to the number of recursive calls from an edge.

Note: To reduce the running time, quickly select the edges chosen for the currently selected range.
Open Questions
Open Questions

- **Vertex Cover**: almost done...
Open Questions

- **Vertex Cover**: almost done...

- Next problem: approximating the size of the maximum matchings up to $\pm \epsilon n$
Open Questions

- **Vertex Cover:** almost done...

- Next problem: approximating the size of the maximum matchings up to $\pm \epsilon n$

- Best algorithm runs in $d^{O(1/\epsilon^2)}$ time. Is there a $\text{poly}(d/\epsilon)$-time algorithm?

  (see [Nguyen, O. 2008] and [Yoshida, Yamamoto, Ito 2009])
Open Questions

- **Vertex Cover:** almost done...

- Next problem: approximating the size of the **maximum matchings** up to $\pm \epsilon n$

  - Best algorithm runs in $d^{O(1/\epsilon^2)}$ time. Is there a $\text{poly}(d/\epsilon)$-time algorithm? (see [Nguyen, O. 2008] and [Yoshida, Yamamoto, Ito 2009])

  - Perhaps not. Is there a $\text{poly}(1/\epsilon)$-time algorithm for planar graphs? (see [Hassidim, Kelner, Nguyen, O. 2009])
Thank You