An application of the Cole-Vishkin algorithm: approximating vertex covers in anonymous networks

Jukka Suomela
Principles of Distributed Computing
10 March 2010
Vertex cover problem

• **Vertex cover** for a graph $G$:
  • Subset $C$ of nodes that “covers” all edges: each edge incident to at least one node in $C$

• **Minimum vertex cover**:
  • Vertex cover with the smallest number of nodes

• **Minimum-weight vertex cover**:
  • Vertex cover with the smallest total weight
Vertex cover problem

- Classical NP-hard optimisation problem: given a graph $G$, find a minimum vertex cover

- Simple 2-approximation algorithm:
  - Find a maximal matching, output all endpoints
  - At most 2 times as large as minimum VC

- No polynomial-time algorithm with approximation factor 1.9999 known
Research question

• Can we find a 2-approximation of a minimum vertex cover in a **distributed setting**?

• Focus:
  
  • Fast, synchronous, **deterministic** distributed algorithms
  
  • Port-numbering model
Distributed algorithms

- Communication graph $G$
- Node = computer
- Edge = communication link
Distributed algorithms

- All nodes are identical, run the same algorithm
- We can choose the algorithm
- An adversary chooses the structure of $G$
- Our algorithm must produce a valid vertex cover in any graph $G$
Synchronous distributed algorithms

1. Each node reads its own **local input**:  
   - node identifier  
   - if we assume unique node IDs  
   - node weight  
   - if we study weighted graphs
Synchronous distributed algorithms

1. Each node reads its own local input
2. Repeat synchronous communication rounds
   ...

[Diagram showing a network of nodes communicating]
Synchronous distributed algorithms

1. Each node reads its own local input
2. Repeat synchronous communication rounds until all nodes have announced their local outputs
   - $1 = \text{in vertex cover}$
Synchronous distributed algorithms

- Running time = number of rounds
- Worst-case analysis
Distributed algorithms: two models

1. Unique identifiers
   - The standard model commonly used in the field

2. Port-numbering model
   - Much weaker model of computation
   - Our focus today
Model 1: Unique identifiers

- Node identifiers are a permutation of $1, 2, \ldots, n$
  - Or a subset of $1, 2, \ldots, \text{poly}(n)$
- Permutation chosen by adversary
Model 2: Port-numbering model

- No unique identifiers
- A node of degree $d$ can refer to its neighbours by integers 1, 2, ..., $d$
- Port-numbering chosen by adversary
Cole-Vishkin algorithm

- **Colour reduction technique**
  - For cycles and trees — similar ideas can be used in more general graphs as well

- Replaces a $k$-colouring with an $O(\log k)$-colouring in one round
  - Repeated application: replaces a $k$-colouring with a 6-colouring in $O(\log^* k)$ rounds
  - Simple additional tricks can be used to find a 3-colouring
Cole-Vishkin algorithm

- Colour reduction technique
- If we have unique identifiers:
  - Interpret unique IDs as an $n$-colouring
  - Cole-Vishkin finds a 3-colouring in $O(\log^* n)$ rounds
- However, we can’t use this trick in the port-numbering model
  - And we are trying to find a vertex cover, not a colouring!
Vertex cover in the port-numbering model

- Convenient to study a more general problem: minimum-weight vertex cover
  - More general problems are sometimes easier to solve?

Notation: \( w(v) = \text{weight of } v \)
Edge packings and vertex covers

- **Edge packing**: weight \( y(e) \geq 0 \) for each edge \( e \)
  - Packing constraint: \( y[v] \leq w(v) \) for each node \( v \), where \( y[v] = \) total weight of edges incident to \( v \)
Edge packings and vertex covers

- **Edge packing**: weight $y(e) \geq 0$ for each edge $e$
  - Packing constraint: $y[v] \leq w(v)$ for each node $v$, where $y[v] = \text{total weight of edges incident to } v$

![Graph diagram]

$y[u] = 2$
$w(u) = 6$
Edge packings and vertex covers

• **Edge packing**: weight \( y(e) \geq 0 \) for each edge \( e \)
  
  • Packing constraint: \( y[v] \leq w(v) \) for each node \( v \), where \( y[v] \) = total weight of edges incident to \( v \)

\[ y[v] = 3 + 0 + 4 + 0 + 0 + 2 \]
\[ w(v) = 9 \]
Edge packings and vertex covers

- Node \( v \) is \textbf{saturated} if \( y[v] = w(v) \)
  - Total weight of edges incident to \( v \) is \textit{equal} to \( w(v) \), i.e., the packing constraint holds with equality
Edge packings and vertex covers

• Edge $e$ is **saturated** if at least one endpoint of $e$ is saturated
  
  • Equivalently: edge weight $y(e)$ can’t be increased

$2 + \varepsilon$ would violate a packing constraint
Edge packings and vertex covers

- **Maximal edge packing**: all edges saturated
  \[ \iff \text{none of the edge weights } y(e) \text{ can be increased} \]
  \[ \iff \text{saturated nodes form a vertex cover!} \]
Edge packings and vertex covers

- Minimum-weight vertex cover $C^*$ difficult to find:
  - Centralised setting: NP-hard
  - Distributed setting: integer problem (choose 0 or 1), symmetry-breaking issues

- Maximal edge packing $y$ easy to find:
  - Centralised setting: trivial greedy algorithm
  - Distributed setting: linear problem, no symmetry-breaking issues (?)
Edge packings and vertex covers

- Minimum-weight vertex cover $C^*$ difficult to find
- Maximal edge packing $y$ easy to find?
- Saturated nodes $C(y)$ in $y$: 2-approximation of $C^*$
  - Textbook proof: LP-duality, relaxed complementary slackness
  - Minimum-weight fractional vertex cover and maximum-weight edge packing are dual problems
  - But we there’s a simple and more elementary proof...
Edge packings and vertex covers

\[ \sum_{v \in C(y)} w(v) \]  
Total weight of saturated nodes

= \[ \sum_{v \in C(y)} y[v] \]  
Saturated nodes have \( y[v] = w(v) \)

= \[ \sum_{e \in E} y(e) | e \cap C(y) | \]  
Interchange the order of summation

\leq 2 \sum_{e \in E} y(e) | e \cap C^* | \]  
Each edge is covered at least \textit{once} by \( C^* \) and at most \textit{twice} by \( C(y) \)

= 2 \sum_{v \in C^*} y[v] \]  
Interchange the order of summation

\leq 2 \sum_{v \in C^*} w(v) \]  
All nodes have \( y[v] \leq w(v) \)
Edge packings and vertex covers

\[ \sum_{v \in C(y)} w(v) \]
\[ = \sum_{v \in C(y)} y[v] \]
\[ = \sum_{e \in E} y(e) \mid e \cap C(y) \mid \]
\[ \leq 2 \sum_{e \in E} y(e) \mid e \cap C^* \mid \]
\[ = 2 \sum_{v \in C^*} y[v] \]
\[ \leq 2 \sum_{v \in C^*} w(v) \]

Saturated nodes

\[ \sum_{v \in C(y)} \sum_{e \in E: v \in e} y(e) \]

Interchange the order of summation

\[ \sum_{e \in E} \sum_{v \in C(y): v \in e} y(e) \quad y[v] = w(v) \]

Interchange the order of summation

Each edge is covered at least once by \( C^* \) and at most twice by \( C(y) \)

All nodes have \( y[v] \leq w(v) \)
Part I: Summary

• Goal:
  • Find a 2-approximation of minimum-weight vertex cover
  • Deterministic algorithm in the port-numbering model

• Idea:
  • Find a maximal edge packing, take saturated nodes

• Part II:
  • Begin with a “greedy but safe” algorithm
  • We will see later how the Cole-Vishkin technique helps
Part II: Finding a maximal edge packing
Finding a maximal edge packing: phase I

- \(y[v]\) = total weight of edges incident to node \(v\)
- **Residual capacity** of node \(v\): \(r(v) = w(v) - y[v]\)
- Saturated node:
  \(r(v) = 0\)
Finding a maximal edge packing: phase I

Start with a trivial edge packing \( y(e) = 0 \)
Finding a maximal edge packing: phase I

Each node $v$ offers $r(v)/\text{deg}(v)$ units to each incident edge
Finding a maximal edge packing: basic idea

Each edge **accepts** the smallest of the 2 offers it received

Increase \( y(e) \) by this amount

- Safe, can’t violate packing constraints
Finding a maximal edge packing: phase I

Update residuals...
Finding a maximal edge packing: phase I

Update residuals, discard saturated nodes and edges...
Finding a maximal edge packing: phase I

Update residuals, discard saturated nodes and edges, repeat...

Offers...
Finding a maximal edge packing: phase I

Update residuals, discard saturated nodes and edges, repeat...

Offers...

Increase weights...
Finding a maximal edge packing: phase I

Update residuals, discard saturated nodes and edges, repeat...

Offers...

Increase weights...

Update residuals...
Finding a maximal edge packing: phase I

Update residuals, discard saturated nodes and edges, repeat...

Offers...

Increase weights...

Update residuals and graph, etc.
Finding a maximal edge packing: phase I

This is a simple deterministic distributed algorithm.

We are making some progress towards finding a maximal edge packing — but...
Finding a maximal edge packing: phase 1

This is a simple deterministic distributed algorithm.

We are making some progress towards finding a maximal edge packing — but this is too slow!
Finding a maximal edge packing: colouring trick

- Offer is a local minimum:
  - Node will be saturated
  - And all edges incident to it will be saturated as well
Finding a maximal edge packing: colouring trick

- Offer is a local minimum:
  - Node will be saturated
- Otherwise there is a neighbour with a different offer:
  - Interpret the offer sequences as “colours”
  - Nodes $u$ and $v$ have different colours: \{u, v\} is multicoloured
Finding a maximal edge packing: colouring trick

• Some progress guaranteed:
  • On each iteration, for each node, at least one incident edge becomes saturated or multicoloured
  • Such edges are be discarded in phase I: node degrees decrease by at least one on each iteration
  • Hence in $\Delta$ iterations all edges are saturated or multicoloured

$\Delta = \text{maximum degree}$
Finding a maximal edge packing: colouring trick

- Phase I: in $\Delta$ rounds all edges are **saturated** or **multicoloured**
  - Saturated edges are good — we’re trying to construct a maximal edge packing
  - Why are the multicoloured edges useful?
Finding a maximal edge packing: colouring trick

- Phase I: in $\Delta$ rounds all edges are saturated or multicoloured
  - Saturated edges are good – we’re trying to construct a maximal edge packing
  - Why are the multicoloured edges useful?
  - Let’s focus on unsaturated nodes and edges
Finding a maximal edge packing: colouring trick

- Colours are sequences of $\Delta$ offers, which are rational numbers.
- Assume that node weights are integers $1, 2, \ldots, W$.
- Let’s analyse the offers more carefully in that case…
Finding a maximal edge packing: colouring trick

- Offers are rationals of the form $q/(\Delta!)^\Delta$
  - Proof idea: multiply weights by $(\Delta!)^\Delta$
  - Then $r(v)$ is a multiple of $(\Delta!)^\Delta$ before iteration 1
  - Offer $r(v)/\deg(v)$ is a multiple of $(\Delta!)^{\Delta-1}$ on iteration 1
  - $r(v)$ is a multiple of $(\Delta!)^{\Delta-1}$ after iteration 1
    - ... (more formally: proof by induction)
- $r(v)$ is a multiple of $\Delta!$ before iteration $\Delta$
- Offers are integers on iteration $\Delta$
Finding a maximal edge packing: colouring trick

- Offers are rationals of the form $q/(\Delta!)^\Delta$
  - Proof idea: if we multiplied weights by $(\Delta!)^\Delta$, then the offers would integers throughout the algorithm
  - Without scaling, we get in the worst case $q/(\Delta!)^\Delta$
- If node weights are integers 1, 2, ..., $W$, then offers are rationals between 0 and $W$
  - Offer of $v$ is at most $r(v) \leq w(v) \leq W$
- There are at most $W(\Delta!)^\Delta$ possible offers!
Finding a maximal edge packing: colouring trick

- Colours are sequences of $\Delta$ offers, which are rational numbers.
- Assume that node weights are integers $1, 2, \ldots, W$.
- Then there are at most $W(\Delta!)^\Delta$ possible offers.
- And hence only $k = (W(\Delta!)^\Delta)^\Delta$ possible colours.
Finding a maximal edge packing: colouring trick

- Only $k = (W(\Delta!))^\Delta$ possible colours
- Replace “inconvenient” colours (sequences of rationals) with “convenient” colours (integers 1, 2, ..., $k$)
Finding a maximal edge packing: phase II

• We have a proper $k$-colouring of the unsaturated subgraph

• Orient from lower to higher colour (acyclic directed graph)
Finding a maximal edge packing: phase II

- We have a proper $k$-colouring of the unsaturated subgraph
- Orient from lower to higher colour (acyclic directed graph)
- Partition in $\Delta$ forests
  - Each node assigns its outgoing edges to different forests
Finding a maximal edge packing: phase II

- For each forest in parallel...
Finding a maximal edge packing: phase II

• For each forest in parallel:
  • Use Cole-Vishkin style colour reduction algorithm
  • Given a $k$-colouring, finds a 3-colouring in time $O(\log^* k)$
Finding a maximal edge packing: phase II

- For each forest and each colour $j = 1, 2, 3$ in sequence:
  - Consider all outgoing edges of colour-$j$ nodes
Finding a maximal edge packing: phase II

- For each forest and each colour \( j = 1, 2, 3 \) in sequence:
  - Consider all outgoing edges of colour-\( j \) nodes
  - Node-disjoint stars: easy to saturate all such edges in parallel
  - Two simple cases:
    - saturate centre
    - saturate all leaves
Finding a maximal edge packing: phase II

- This way we can saturate all multicoloured edges:
  - Each edge belongs to one forest, and its tail has colour 1, 2, or 3
  - We simply go through all forests and all colours and therefore saturate everything
Finding a maximal edge packing: algorithm overview

- **Phase I:**
  - All edges become saturated or multicoloured

- **Phase II:**
  - Multicoloured edges are partitioned in $\Delta$ forests
  - Forests are 3-coloured
  - 3-coloured forests are saturated
Finding a maximal edge packing: running time analysis

• Total running time:
  • All edges become saturated or multicoloured: $O(\Delta)$
  • Multicoloured forests are 3-coloured: $O(\log^* k)$
  • 3-coloured forests are saturated: $O(\Delta)$

• $O(\Delta + \log^* k) = O(\Delta + \log^* W)$
  • $k$ is huge, but $\log^*$ grows slowly
Finding a maximal edge packing: summary

- Maximal edge packing and 2-approximation of vertex cover in time $O(\Delta + \log^* W)$
  - $W =$ maximum node weight
- Unweighted graphs: running time simply $O(\Delta)$, independent of $n$
- Everything can be implemented in the port-numbering model
Finding a maximal edge packing: recap

Phase I:

- **Residuals**
  \[ r(v) = w(v) - y[v] \]
- **Offer** \( r(v)/\deg(v) \)
- **Accept minimum**, increase weights
- **Progress**: edges become *saturated* or *multicoloured* (different offers)
Finding a maximal edge packing: recap

Phase II:

- Saturated edges are already ok, we focus on multicoloured edges
- Colours are sequences of offers, re-colour with integers 1, 2, \ldots, k
- Partition in \( \Delta \) forests
- Cole-Vishkin: 3-colouring
- Use colours to saturate all edges
Finding a maximal edge packing: some intuition

- Regular graph with uniform weights:
  - Symmetry-breaking (e.g., graph colouring) is not possible in the port-numbering model
  - But it is trivial to find a maximal edge packing directly

- “Irregular” graph:
  - We have symmetry-breaking information, which can be used to find a graph colouring, which can be used to find a maximal edge packing

- Handling these two cases turns out to be enough!
Take-home messages

• Non-trivial problems can be solved in very restrictive models of distributed computing

• Generalise!
  • More difficult problems may be easier to solve: vertex cover $\rightarrow$ weighted vertex cover $\rightarrow$ weighted set cover...

• Cole-Vishkin technique is a powerful tool
  • Wide range of applications far beyond the textbook examples of colouring cycles with numerical IDs
  • $\log^*$ of almost everything is something reasonable