Local approximation algorithms for vertex cover

Jukka Suomela
Joint work with Matti Åstrand, Patrik Floréen, Valentin Polishchuk, Joel Rybicki, and Jara Uitto

HIIT seminar, 16 October 2009
Part I: Introduction

Vertex cover problem in a distributed setting
Given a graph $G = (V, E)$, find a smallest $C \subseteq V$ that covers every edge of $G$

- i.e., each edge $e \in E$ incident to at least one node in $C$

Classical NP-hard optimisation problem
Vertex cover in a distributed setting

Node = computer
Edge = communication link

Each node must decide whether it is in the cover $C$
Vertex cover in a distributed setting

Graph is unknown, all nodes run the same algorithm

*Initially*: Each node knows its own degree and the maximum degree $\Delta$
Vertex cover in a distributed setting

*Port numbering*: each node has chosen an ordering on its incident edges
Vertex cover in a distributed setting

Communication primitives:

- “send message $m$ to port $i$”
- “let $m$ be the message received from port $i$”
Vertex cover in a distributed setting

*Synchronous communication round*: Each node

1. performs local computation
Vertex cover in a distributed setting

**Synchronous communication round**: Each node

1. performs local computation
2. sends a message to each neighbour
Vertex cover in a distributed setting

*Synchronous communication round*: Each node

1. performs local computation
2. sends a message to each neighbour (message propagation...)
Vertex cover in a distributed setting

*Synchronous communication round*: Each node

1. performs local computation
2. sends a message to each neighbour
3. receives a message from each neighbour
Finally: Each node performs local computation and announces its output: whether it is in the cover $C$

Running time = number of communication rounds
Focus:

- deterministic algorithm
- strictly *local algorithm*, running time independent of $n = |V|$ (but may depend on maximum degree $\Delta$)
- *the best possible approximation ratio*
Prior work

Kuhn et al. (2006):

- (2 + \(\varepsilon\))-approximation in \(O(\log \Delta / \varepsilon^4)\) rounds

Czygrinow et al. (2008), Lenzen & Wattenhofer (2008):

- (2 − \(\varepsilon\))-approximation requires \(\Omega(\log^* n)\) rounds, even if \(\Delta = 2\)

What about 2-approximation?

Is it possible in \(f(\Delta)\) rounds, for some \(f\)?
Deterministic 2-approximation algorithm for vertex cover

- Running time $O(\Delta)$ synchronous rounds

Surprise: node identifiers not needed

- Negative result for $(2 - \epsilon)$-approximation holds even if there are unique node identifiers
- Our algorithm can be used in anonymous networks
Maximal matchings and edge packings
In a centralised setting, 2-approximation is easy: find a *maximal matching*, take all matched nodes.

But matching requires $\Omega(\log^* n)$ rounds and unique identifiers.

- symmetry breaking!
**Background: maximal edge packing**

*Edge packing* = edge weights from $[0, 1]$, for each node $v \in V$, total weight on incident edges $\leq 1$

*Maximal*, if no weight can be increased
Maximal matching $\implies$ maximal edge packing

(matched: weight 1, unmatched: weight 0)
Maximal matching requires symmetry breaking

Maximal edge packing does not
Node *saturated* if total weight on incident edges = 1

Saturated nodes in a maximal edge packing = 2-approximation of vertex cover (proof: LP duality)
Node *saturated* if total weight on incident edges = 1

Saturated nodes in a maximal edge packing = 2-approximation of vertex cover

***

So we only need to design a distributed algorithm that finds a maximal edge packing

Warm-up: how to find a (non-trivial) edge packing?
A simple approach: a node of degree $d$ offers $1/d$ of its “capacity” to each incident edge

(Capacity $= 1 - \text{total weight of incident edges}$)
Each edge *accepts* the minimum of the two offers

(cf. Khuller et al. 1994, Papadimitriou and Yannakakis 1993)
Finding an edge packing

Looks good, some progress is guaranteed, and we might even saturate some nodes.

But this is not a maximal edge packing yet.
Finding an edge packing

Remaining capacities are now unwieldy fractions, even though our starting point was unweighted!

Unweighted instance $\Rightarrow$ weighted subproblems
Finding an edge packing

Pessimist’s take:

• Solving this will be as hard as finding maximal edge packings in weighted graphs

• Let’s try something else

Optimist’s take:

• If we solve this, we can also find maximal edge packings in weighted graphs

• Let’s do it!
Finding maximal edge packings in unweighted graphs
Finding an edge packing

Construct a 2-coloured *bipartite double cover*

Each original node simulates two nodes of the cover
Finding an edge packing

Find a maximal matching in the 2-coloured graph

Easy in $O(\Delta)$ rounds
Finding an edge packing

Give \( \frac{1}{2} \) units of weight to each edge in matching.
Many possibilities...
Finding an edge packing

Many possibilities...
Finding an edge packing

Many possibilities...
Finding an edge packing

Always: weight $\frac{1}{2}$ paths and cycles and weight 1 edges

Valid edge packing
Finding a maximal edge packing

Not necessarily maximal – but all unsaturated edges adjacent to two weight \( \frac{1}{2} \) edges
In any graph:

Unsaturated edges adjacent to two weight $\frac{1}{2}$ edges

$\Delta = 3$
Finding a maximal edge packing

In any graph:

Unsaturated edges adjacent to two weight $\frac{1}{2}$ edges

Delete saturated edges

$\Delta = 3 \rightarrow \Delta = 2$
Each node has lost at least one neighbour

Remaining capacity of each node is exactly $\frac{1}{2}$

$\Delta = 3 \rightarrow \Delta = 2$
Finding a maximal edge packing

Repeat

\[ \Delta = 2 \]
Finding a maximal edge packing

Delete saturated edges

$\Delta = 2 \rightarrow \Delta = 1$
Finding a maximal edge packing

Each node has lost at least one neighbour

Remaining capacity of each node is exactly $\frac{1}{4}$

$\Delta = 2 \rightarrow \Delta = 1$
Finding a maximal edge packing

Repeat...
Finding a maximal edge packing

Repeat...

Maximum degree decreases on each iteration

Everything saturated in $\Delta$ iterations
Maximal edge packing in $(\Delta + 1)^2$ rounds

$\implies$ 2-approximation of vertex cover
Maximal edge packing in $(\Delta + 1)^2$ rounds

$\implies$ 2-approximation of vertex cover

But it seems that this cannot be generalised to approximate minimum-weight vertex cover

A different approach needed
Finding maximal edge packings in weighted graphs
Recall the simple algorithm: a node of degree \( d \) offers \( \frac{1}{d} \) of its “capacity” to each incident edge. Each edge accepts the minimum of the two offers.
Starting point has non-uniform capacities, ok if subproblems have non-uniform capacities!
Let’s study this approach more carefully...
Key observation: For each node

1. at least one incident edge becomes saturated
   (= cannot increase edge weight), or ...
Finding an edge packing

Key observation: for each node

1. at least one incident edge becomes saturated, or
2. at least one incident edge got two different offers
Finding an edge packing

Key observation: for each node

1. at least one incident edge becomes saturated, or
2. at least one incident edge got two different offers

We can interpret the offers as “colours”

Progress is guaranteed: edges become saturated or multi-coloured
After $\Delta$ iterations: each edge saturated or multi-coloured

At this point, colours are huge integers

$$1, 2, \ldots, (W(\Delta!)^\Delta)^\Delta$$

but Cole–Vishkin (1986) techniques can be used to reduce the number of colours to $\Delta + 1$ very fast

Then we can use the colours to saturate all edges

($W = \text{maximum weight}$)
Finding an edge packing

In summary, maximal edge packing in $O(\Delta + \log^* W)$ rounds, where $W = \text{maximum weight}$

That is, $O(\Delta)$ rounds in unweighted graphs!

- Better running time and easier to design than pessimist’s algorithm
- A similar approach can be used for the set cover problem
Summary

- Two distributed 2-approximation algorithms for the vertex cover problem
- Running times: $O(\Delta^2)$ and $O(\Delta)$ rounds, deterministic, can be self-stabilised
- Strictly local algorithms – running time independent of number of nodes
- Be optimistic: more general problems are sometimes easier to tackle