Fast distributed approximation algorithms for vertex cover and set cover in anonymous networks

Åstrand, Matti

ACM
2010

Åstrand , M & Suomela , J 2010 , Fast distributed approximation algorithms for vertex cover and set cover in anonymous networks . in F Meyer auf der Heide & C A Phillips (eds) , þÿSPAA 10 : Proceedings of the Twenty-Second Annual Symposium on Pa þÿAlgorithms and Architectures, June 13 15, 2010. Thira, Santorini, Gree þÿ294 302 , ACM Symposium on Parallelism in Algorithms and Architectu Santorini , Greece , 13/06/2010 . https://doi.org/10.1145/1810479.1810533

http://hdl.handle.net/10138/27922
https://doi.org/10.1145/1810479.1810533

acceptedVersion

Fast Distributed Approximation Algorithms for Vertex Cover and Set Cover in Anonymous Networks

Matti Åstrand
matti.astrand@helsinki.fi

Jukka Suomela
jukka.suomela@cs.helsinki.fi

Helsinki Institute for Information Technology HIIT, University of Helsinki
P.O. Box 68, FI-00014 University of Helsinki

ABSTRACT

We present a distributed algorithm that finds a maximal edge packing in $O(\Delta + \log^* W)$ synchronous communication rounds in a weighted graph, independent of the number of nodes in the network; here Δ is the maximum degree of the graph and W is the maximum weight. As a direct application, we have a distributed 2-approximation algorithm for minimum-weight vertex cover, with the same running time. We also show how to find an f-approximation of minimum-weight set cover in $O(f^2k^2 + fk \log^* W)$ rounds; here k is the maximum size of a subset in the set cover instance, f is the maximum frequency of an element, and W is the maximum weight of a subset. The algorithms are deterministic, and they can be applied in anonymous networks.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*

General Terms

Algorithms, Theory

1. INTRODUCTION

In this work, we present deterministic distributed approximation algorithms for two classical problems: minimum-weight vertex cover and minimum-weight set cover.

1.1 Edge Packings and Vertex Covers

Let $\mathcal{G} = (V, E)$ be a simple, undirected, node-weighted graph; each node $v \in V$ is associated with a positive weight w_v. A set $C \subseteq V$ is a *vertex cover* if each edge has at least one endpoint in C, and it is a *minimum-weight vertex cover* if it also minimises its total weight

$$w(C) = \sum_{v \in C} w_v.$$

http://doi.acm.org/10.1145/1810479.1810533

While vertex cover is a classical NP-hard optimisation problem, there is a simple technique for obtaining efficient approximation algorithms: find a maximal edge packing (a maximal dual solution) and output all saturated nodes. For a nonnegative function $y\colon E \to [0, +\infty)$, let us define the shorthand notation

$$y[v] = \sum_{e \in E\colon v \in e} y(e)$$

for each $v \in V$. We say that y is an *edge packing* if $y[v] \leq w_v$ for all $v \in V$. A node $v \in V$ is *saturated* in the edge packing y if $y[v] = w_v$. An edge $e = \{u, v\} \in E$ is saturated if u or v or both are saturated, i.e., $y(e)$ cannot be increased without violating the constraint $y[u] \leq w_u$ or $y[v] \leq w_v$. An edge packing y is *maximal* if all edges are saturated.

Let $C(y)$ be the set of nodes saturated in y. The classical result by Bar-Yehuda and Even [6] shows that if y is a maximal edge packing then $C(y)$ is a 2-approximation of a minimum-weight vertex cover; for the sake of completeness, we give a short proof here. First, observe that $C(y)$ is a vertex cover by definition: if an edge is not covered by $C(y)$, then y is not maximal. To show the approximation ratio, let C^* be a minimum-weight vertex cover. As $C(y)$ contains at most two endpoints of each edge and C^* contains at least one endpoint of each edge, we have

$$\begin{aligned} w(C(y)) &= \sum_{v \in C(y)} y[v] = \sum_{e \in E} y(e)\,|e \cap C(y)| \\ &\leq 2 \sum_{e \in E} y(e)\,|e \cap C^*| = 2 \sum_{v \in C^*} y[v] \leq 2w(C^*). \end{aligned}$$

In a centralised setting, a maximal edge packing y is easy to find: for each $e \in E$, in an arbitrary order, increase the value $y(e)$ until one of the endpoints of e becomes saturated. In this work, we give an efficient *distributed* algorithm that finds a maximal edge packing, and hence also a 2-approximation of a minimum-weight vertex cover.

1.2 Fractional Packings and Set Covers

To deal with the set cover problem in a distributed setting, it is convenient to restate the problem by using a bipartite graph $\mathcal{H} = (S \cup U, A)$. Each node $s \in S$ represents a *subset*, each node $u \in U$ represents an *element* of the universe, and an edge $\{s, u\} \in A$ denotes that the element $u \in U$ is a member of the subset $s \in S$. Each subset node $s \in S$ is associated with a positive weight w_s. A collection $C \subseteq S$ is a *set cover* if each element $u \in U$ has at least one neighbour in C, and it is a *minimum-weight set cover* if it also minimises its total weight $w(C) = \sum_{s \in C} w_s$.

Let $y\colon U \to [0, +\infty)$ be a nonnegative function. Define the shorthand notation

$$y[s] = \sum_{u \in N(s)} y(u)$$

for each $s \in S$; here $N(s) \subseteq U$ is the set of elements adjacent to the subset node s. We say that y is a *fractional packing* if $y[s] \leq w_s$ for all subset nodes $s \in S$. A subset node $s \in S$ is *saturated* in the fractional packing y if $y[s] = w_s$. An element $u \in U$ is saturated if at least one adjacent subset node s with $\{s, u\} \in A$ is saturated, i.e., $y(u)$ cannot be increased. A fractional packing y is *maximal* if all elements $u \in U$ are saturated.

The classical result mentioned in Section 1.1 has a straightforward generalisation to the set cover problem [6]. Let $C(y)$ be the set of subset nodes $s \in S$ saturated in y. Let f be the maximum degree of the elements $u \in U$, that is, an element occurs in at most f subsets. Now if y is a maximal fractional packing, then $C(y)$ is an f-approximation of a minimum-weight set cover. The proof in the previous section holds almost verbatim, replacing the constant 2 with f.

1.3 Model of Distributed Computing

In our vertex cover algorithm, the graph $\mathcal{G} = (V, E)$ represents a distributed system: each node $v \in V$ is a computational entity and each edge $\{u, v\} \in E$ denotes a communication link between the nodes u and v. Similarly, in our set cover algorithm, the graph $\mathcal{H} = (S \cup U, A)$ represents a distributed system, and each node $x \in S \cup U$ is a computational entity. (Naturally, we do not assume that the physical structure of a real-world distributed system is exactly equal to $\mathcal{H}$; it is sufficient that we can efficiently simulate computation in $\mathcal{H}$ by using the physical computers and communication links between them. Section 5 gives an example of such a simulation.)

We use the model of synchronous distributed algorithms. All nodes execute the same algorithm. During each *synchronous communication round*, each node in parallel (i) performs local computation, (ii) sends one message to each of its neighbours, (iii) waits while the messages are propagated along the edges of the communication graph, and (iv) receives one message from each of its neighbours. As usual, the running time of a synchronous distributed algorithm is the total number of synchronous communication rounds.

In the case of the vertex cover problem, each node $v \in V$ gets its own weight w_v as input. When the algorithm terminates, each node must produce one bit of output: whether it is part of the vertex cover or not. In the set cover problem, each subset node $s \in S$ gets its own weight w_s as input; the elements $u \in U$ have no input. When the algorithm terminates, each $s \in S$ must output one bit indicating whether it is in the set cover or not.

Our algorithms are designed for *anonymous networks*; we do not assume that the nodes have any unique identifiers. Hence we have to specify carefully how the nodes can address their neighbours when they send and receive messages. We consider two models:

Port-Numbering Model. A node v with degree $\deg(v)$ can refer to its neighbours by integers $1, 2, \ldots, \deg(v)$; these integers are called *port numbers*. A node can send a different message to each neighbour, and it knows which message was received from which neighbour.

Broadcast Model. There are no port numbers. A node has to send the same message to each neighbour, and it does not know which message was received from which neighbour.

Put otherwise, in the port-numbering model, a node v produces a *vector* with $\deg(v)$ outgoing messages and it receives a *vector* with $\deg(v)$ incoming messages; the ith outgoing message corresponds to the same neighbour as the ith incoming message. In the broadcast model, a node v produces only *one* outgoing message and it receives a *multiset* with $\deg(v)$ incoming messages. We discuss the properties of the broadcast model in more detail in Section 7.

1.4 Notation

We focus on the case of bounded degrees and bounded weights. In the vertex cover algorithm, we assume that there are global parameters Δ and W such that $\deg(v) \leq \Delta$ and $w_v \in \{1, 2, \ldots, W\}$ for all $v \in V$. We assume that all nodes know these two parameters, which may represent, e.g., intrinsic hardware constraints such as the number of physical communication ports in a device and the precision of the registers used to store the weights; note that the algorithms are fast even if one chooses a very large value of W such as $W = 2^{64}$.

Similarly, in the case of the set cover algorithm, we assume that there are global parameters f, k, and W such that $\deg(u) \leq f$, $\deg(s) \leq k$, and $w_s \in \{1, 2, \ldots, W\}$ for all $u \in U$ and $s \in S$.

Throughout this work, logarithms are to base 2. The function $\log^* n$ denotes the iterated logarithm of n, that is, $\log^* n = 0$ if $n \leq 1$, and otherwise $\log^* n = 1 + \log^*(\log n)$.

1.5 Contributions

In Section 3, we present a distributed algorithm that finds a maximal edge packing in $O(\Delta + \log^* W)$ synchronous communication rounds. As a direct application, we have a distributed 2-approximation algorithm for minimum-weight vertex cover with the same running time. For unweighted graphs ($W = 1$), the running time is simply $O(\Delta)$. This algorithm assumes the port-numbering model.

In Section 4, we present a distributed algorithm that finds a maximal fractional packing in $O(f^2 k^2 + fk \log^* W)$ rounds. Again, as a direct application, we have a distributed f-approximation algorithm for minimum-weight set cover with the same running time. This algorithm does not require port numbering; we show how to implement the algorithm in the broadcast model.

In Section 5, we show how to apply the algorithm of Section 4 to find a maximal edge packing and 2-approximation of vertex cover in $O(\Delta^2 + \Delta \log^* W)$ rounds in the broadcast model.

Our algorithms are strictly *local* in the sense that that running time of the algorithm does not depend on the number of nodes in the network [27, 31]. Moreover, our algorithms are deterministic. Among others, this means that standard techniques [4, 5, 23] can be used to convert our algorithms into efficient *self-stabilising algorithms* [10].

2. RELATED WORK

In an unweighted graph, a maximal matching provides a maximal edge packing and therefore a 2-approximation of a minimum vertex cover. Hańćkowiak et al.'s [13] distributed

Table 1: A Comparison of Fast Distributed Algorithms for Vertex Cover

deterministic	weighted	approximation	time ($W = 1$)	algorithm
no	yes	2	$O(\log n)$	[12]
no	yes	2	$O(\log n)$	[17]
yes	no	3	$O(\Delta)$	[30]
yes	no	2	$O(\log^4 n)$	[13] (matching)
yes	no	2	$O(\Delta + \log^* n)$	[28] (matching)
yes	no	2	$O(\Delta^2)$	[2]
yes	yes	$2+\varepsilon$	$O(\log(\varepsilon^{-1}) \log n)$	[16]
yes	yes	$2+\varepsilon$	$O(\varepsilon^{-4} \log \Delta)$	[21] + [14]
yes	yes	2	$O(\Delta + \log^* n)$	[28] (edge colouring)
yes	yes	2	$O(1)$ if $\Delta \leq 3$	[2]
yes	yes	2	$O(\Delta)$	this work

In the table, $n = |V|$ and $\varepsilon > 0$. The running times are stated for the case of unweighted graphs. For randomised algorithms the running times hold in expectation or with high probability.

algorithm finds a maximal matching in $O(\log^4 n)$ rounds, and Panconesi and Rizzi's [28] algorithm finds a maximal matching in $O(\Delta + \log^* n)$ rounds.

In a weighted graph, we can use an edge colouring to find a maximal edge packing. Given an edge colouring with k colours, we can find a maximal edge packing in $O(k)$ rounds: first saturate all edges of colour 1 in parallel, then saturate all edges of colour 2 in parallel, etc. For example, Panconesi and Rizzi's [28] algorithm finds an $O(\Delta)$-edge colouring in $O(\Delta + \log^* n)$ rounds, and hence provides an $O(\Delta + \log^* n)$-time algorithm for finding a maximal edge packing.

However, any deterministic algorithm that uses maximal matchings or edge colourings has two drawbacks. First, such algorithms assume that each node has a unique identifier – indeed, finding a maximal matching or an edge colouring is impossible in anonymous networks. Second, the running time of any such algorithm must depend on the number of nodes in the network, even in the case $\Delta = 2$ – this is the seminal result by Linial [25].

Other vertex cover algorithms are summarised in Table 1. To our knowledge, none of the algorithms from prior work has the same combination of features as our algorithm: (i) deterministic, (ii) 2-approximation, (iii) for weighted vertex cover, and (iv) running time independent of n.

Many vertex cover algorithms have also a generalisation to set covering. For example, LP approximation schemes [18, 21] and deterministic rounding [14] provide a $(2+\varepsilon)$-approximation for vertex cover, and the same technique gives an $(f+\varepsilon)$-approximation for set cover. However, there are also algorithms that are relevant specifically in the case of the set cover problem. A trivial constant-time algorithm provides a k-approximation: each element $u \in U$ chooses an adjacent subset $s \in S$ of minimum weight; all such subsets are added to the cover. Randomised LP rounding [18, 20, 21, 22] gives the expected approximation factor of $O(\log k)$.

Several lower bounds are known for local algorithms (distributed algorithms with running time independent of the number of nodes in the network). Czygrinow et al. [9] and Lenzen and Wattenhofer [24] have shown that finding a constant-factor approximation of maximum independent set in a directed cycle is not possible in $O(1)$ rounds using a deterministic algorithm. As a direct consequence, even if $W = 1$ and $\Delta = 2$, there is no deterministic local $(2-\varepsilon)$-approximation algorithm for vertex cover. A straightforward local reduction shows that there is no deterministic local $(\min\{f, k\} - \varepsilon)$-approximation algorithm for set cover, either (see Section 6 for details). This lower bound is tight, as our local algorithm achieves the approximation factor f and the trivial algorithm achieves the approximation factor k.

The running time of our edge packing algorithm depends on both Δ and W, and both of these are unavoidable. First, even if $W = 1$, there is no $O(1)$-approximation algorithm for vertex cover with running time $o(\log \Delta / \log \log \Delta)$ [19]. Second, even if $\Delta = 2$, there is no deterministic algorithm for maximal edge packing with running time $O(1)$ [2].

3. VERTEX COVER IN THE PORT-NUMBERING MODEL

In this section we present an algorithm that finds a maximal edge packing (and hence a 2-approximation of vertex cover) in $O(\Delta + \log^* W)$ rounds. The algorithm uses the port-numbering model.

3.1 Overview

Our algorithm works in two phases. Phase I constructs a (possibly non-maximal) edge packing y, and a (possibly improper) node colouring c in the graph $\mathcal{G}$. We say that an edge is *multicoloured* if its endpoints have different colours in c. The key observation is that if an edge is not saturated in y, then it is multicoloured in c. Phase II uses the colouring c to saturate all multicoloured edges.

Phase I uses the degrees and the weights of the nodes in $\mathcal{G}$ to derive both the edge packing y and the colouring c. It turns out that in those cases in which finding multicoloured edges is impossible – regular graphs with equal node weights – we can saturate the edges already during Phase I.

Phase I uses steps that are similar to, e.g., Khuller et al.'s [16] algorithm or Papadimitriou and Yannakakis's [29] "safe algorithm": each node $v \in V$ offers $w_v / \deg(v)$ units to each incident edge, and each edge accepts the minimum of the offers that it receives. Phase II is similar to the graph colouring algorithm by Goldberg et al. [11] and the edge colouring algorithm by Panconesi et al. [28]: we partition the multicoloured edges into Δ forests of rooted trees, and

we use Cole and Vishkin's [8] colour reduction techniques to 3-colour each tree.

3.2 Phase I

For an edge packing y and an improper colouring c, let $r_y(v) = w_v - y[v]$ be the *residual weight* of $v \in V$, and let

$$E_{yc} = \big\{\{u,v\} \in E : r_y(v) > 0,\, r_y(u) > 0,\, c(u) = c(v)\big\}$$

be the set of edges that are not saturated in y and not multicoloured in c. Let $\mathcal{G}_{yc} = (V_{yc}, E_{yc})$ be the subgraph of $\mathcal{G}$ induced by E_{yc}, and let $\deg_{yc}(v)$ be the degree of $v \in V_{yc}$ in the subgraph $\mathcal{G}_{yc}$.

In the distributed algorithm, the colour $c(v)$ is stored in the local memory of the node $v \in V$, and identical copies of the value $y(e)$ are stored in both endpoints of the edge $e \in E$. During Phase I, the colours will be sequences of rational numbers. Initially, we set $y(e) \leftarrow 0$ for each $e \in E$, and $c(v)$ is an empty sequence for each node $v \in V$. At this point, no edge is saturated or multicoloured, and hence $\mathcal{G}_{yc} = \mathcal{G}$.

During the algorithm, we will increment the values $y(e)$ and add more elements to the sequences $c(v)$ until $E_{yc} = \emptyset$. It turns out that it is sufficient to repeat the following steps for Δ times:

(i) Set $x(v) \leftarrow r_y(v)/\deg_{yc}(v)$ for each node $v \in V_{yc}$.

(ii) Set $y(e) \leftarrow y(e) + \min\{x(u), x(v)\}$ for each edge $e = \{u,v\} \in E_{yc}$.

(iii) Add the element $x(v)$ to the sequence $c(v)$ for each node $v \in V_{yc}$, and add the element 1 for each node $v \in V \setminus V_{yc}$.

The following lemma shows the correctness of the algorithm.

LEMMA 1. *In each iteration of steps (i)–(iii), the maximum degree of $\mathcal{G}_{yc}$ decreases by at least one.*

PROOF. Let $v \in V_{yc}$ before step (i). If we have $x(u) \geq x(v)$ for each neighbour u of v in $\mathcal{G}_{yc}$, then we will saturate v during the step (ii). Otherwise there is a neighbour u of v with $x(u) < x(v)$. If the edge $\{u,v\}$ is not saturated after step (ii), it is multicoloured after step (iii). In summary, each node is removed from $\mathcal{G}_{yc}$ or loses at least one edge. Moreover, edges which were saturated remain saturated and edges which were multicoloured remain multicoloured. □

It follows that after Δ iterations, $\mathcal{G}_{yc}$ is empty and all edges are saturated or multicoloured. At this point, the colours $c(v)$ are – somewhat inconveniently – sequences of Δ rational numbers. However, the rational numbers in $c(v)$ are not arbitrary, as shown by the following lemma.

LEMMA 2. *For each $v \in V$ and for each element q of $c(v)$, we have $0 < q \leq W$ and $q(\Delta!)^\Delta \in \mathbb{N}$.*

PROOF. A simple induction shows that if we multiply each weight w_v by $(\Delta!)^k$ before running the algorithm, then $x(v)$, $y(e)$, and $r_y(v)$ will be integral during the first k iterations of steps (i)–(iii). □

Hence an injection can be defined from the possible values of $c(v)$ to the set $\{1, 2, \ldots, \chi\}$ for $\chi = (W(\Delta!)^\Delta)^\Delta$. In what follows, we re-interpret the colouring c as a mapping $c\colon V \to \{1, 2, \ldots, \chi\}$.

3.3 Phase II

Let

$$A = \big\{(u,v) : \{u,v\} \in E,\; r_y(v) > 0,\, r_y(u) > 0,\, c(u) < c(v)\big\}$$

be the set of unsaturated edges, oriented from a lower to higher colour. Note that the directed graph $\mathcal{G}' = (V, A)$ is acyclic, and c is a proper χ-colouring in $\mathcal{G}'$.

We partition A in Δ forests, $F_1, F_2, \ldots, F_\Delta$: using the port numbering, each node u adds the first outgoing edge to F_1, the second outgoing edge to F_2, etc. The outdegree of each node in $\mathcal{F}_i = (V, F_i)$ is at most one, and hence it is a forest of rooted trees, with edges oriented towards the root nodes.

For each forest $\mathcal{F}_i$ in parallel, we can find a 3-colouring in $O(\log^* \chi)$ rounds by using a Cole–Vishkin style colour reduction algorithm [8, 11]. For each $j \in \{1,2,3\}$, let F_{ij} consist of the edges $(u,v) \in F_i$ such that u has colour j in the forest $\mathcal{F}_i$.

We consider each $i \in \{1, 2, \ldots, \Delta\}$ and $j \in \{1,2,3\}$ in sequence, and saturate all edges in F_{ij}. Since the graph induced by F_{ij} consists of rooted stars (i.e., rooted trees of height 1), it is easy to saturate all edges of F_{ij} in parallel. Consider a star, with the root node $v \in V$ and with the leaf nodes $L \subset V$. In one communication round, the root node can gather the residual weights $r_y(u)$ for all leaves $u \in L$ and compute the ratio

$$\alpha = \sum_{u \in L} \frac{r_y(u)}{r_y(v)}.$$

If $\alpha < 1$, we increase $y(\{u,v\})$ for each $u \in L$ by $r_y(u)$, and we saturate all leaf nodes. Otherwise we increase $y(\{u,v\})$ for each $u \in L$ by $r_y(u)/\alpha$, and we saturate the root node.

The sets F_{ij} form a partition of A. In summary, we have saturated all edges of A in $O(\Delta + \log^* \chi)$ communication rounds.

THEOREM 1. *There is a deterministic algorithm that finds a maximal edge packing in $O(\Delta + \log^* W)$ synchronous communication rounds.*

PROOF. Phase II saturates all edges that were not saturated in Phase I. Since Phase I takes $O(\Delta)$ rounds and Phase II takes $O(\Delta + \log^* \chi)$ rounds, it is sufficient to show that $\log^* \chi = O(\log^* \Delta + \log^* W)$. To this end, let $M = \max\{W, \Delta, 4\}$ and observe that $\log\log \chi \leq 4 \log M$. □

4. SET COVER IN THE BROADCAST MODEL

In this section we present an algorithm that finds a maximal fractional packing (and hence an f-approximation of set cover) in $O(f^2k^2 + fk \log^* W)$ rounds in the broadcast model.

Our fractional packing algorithm builds on the same basic idea as the edge packing algorithm in Section 3: we construct a (non-maximal) packing and an (improper) colouring hand in hand, and if a step does not improve the packing, we can show that it will improve the colouring. However, there are also many differences between the two algorithms. For example, while the edge packing algorithm runs the Cole–Vishkin colour reduction step only once, in the case of fractional packing we will have to run the colour reduction step repeatedly.

4.1 Preliminaries

Recall that we represent a set cover instance as a bipartite graph $\mathcal{H} = (S \cup U, A)$ with subset nodes $s \in S$ and elements $u \in U$. We use $N(s) \subseteq U$ to denote the set of elements adjacent to $s \in S$, and conversely $N(u) \subseteq S$ to denote the set of subset nodes adjacent to $u \in U$. Let

$$K = \big\{(u, s, v) : s \in S,\, u \in N(s),\, v \in N(s),\, u \neq v\big\}$$

consist of all length-2 simple paths in $\mathcal{H}$ that start and end in U. If we interpret $(u, s, v) \in K$ as a directed edge from u to v, we can construct the directed multigraph $\mathcal{K} = (U, K)$. In $\mathcal{K}$ there are $|N(u) \cap N(v)|$ directed edges from u to v. The outdegree, indegree, and number of distinct neighbours of any node $u \in U$ in $\mathcal{K}$ is at most $D = (k-1)f$. The graph $\mathcal{K}$ is used only in the analysis of the algorithm; we do not maintain it explicitly.

During the execution of the algorithm, each element $u \in U$ is associated with two values: $y(u) \in [0, +\infty)$ and $c(u) \in \{1, 2, \ldots, D+1\}$. Here y is a fractional packing, and c is an improper colouring of $\mathcal{K}$. We say that an edge $(u, s, v) \in K$ is *multicoloured* if $c(u) \neq c(v)$. Recall that an element $u \in U$ is *saturated* if it is adjacent to a saturated subset node $s \in S$.

Let $r_y(s) = w_s - y[s]$ be the residual weight of the subset node $s \in S$. Let $U_y \subseteq U$ be the set of elements $u \in U$ that are *not* saturated in y, let

$$U_{yi} = \big\{u \in U_y : c(y) = i\big\}$$

consist of unsaturated elements of colour i, and let $U_{yi}(s) = U_{yi} \cap N(s)$ be the set of unsaturated elements of colour i adjacent to s. Let

$$K_{yc} = \big\{(u, s, v) \in K : u \in U_y,\, v \in U_y,\, c(u) = c(v)\big\}$$

be the set of edges of $\mathcal{K}$ that are not multicoloured in c and join unsaturated elements. We define the subgraph $\mathcal{K}_{yc}$ of $\mathcal{K}$ by setting $\mathcal{K}_{yc} = (U_y, K_{yc})$.

Remark 1. Even thought the definitions resemble those of the edge packing algorithm, they are not completely analogous. In Section 3, we associate a colour $c(v)$ with each *node* of $\mathcal{G}$ and a value $y(e)$ with each *edge* of $\mathcal{G}$. In this section, we associate a colour $c(u)$ with each *node* of $\mathcal{K}$ and also a value $y(u)$ with each *node* of $\mathcal{K}$.

4.2 Algorithm

Initially, we set $y(u) \leftarrow 0$ and $c(u) \leftarrow 1$ for each element $u \in U$. None of the nodes of $\mathcal{K}$ is saturated and none of the edges is multicoloured, and hence $\mathcal{K}_{yc} = \mathcal{K}$; see Figure 1 for an example.

The algorithm consists of $D+1$ *iterations*. Each iteration $j \in \{1, 2, \ldots, D+1\}$ proceeds as follows:

(a) For each colour $i \in \{1, 2, \ldots, D+1\}$, perform the *saturation phase* for colour i; see Section 4.3.

(b) Perform the *colouring phase*; see Section 4.4.

Eventually, after the last iteration, U_y will be empty and hence y is a maximal edge packing.

4.3 Saturation Phases

The saturation phase for colour i consists of the following steps; see Figure 1a for an example. In the following, we use the shorthand notation $S' = \{s \in S : U_{yi}(s) \neq \emptyset\}$.

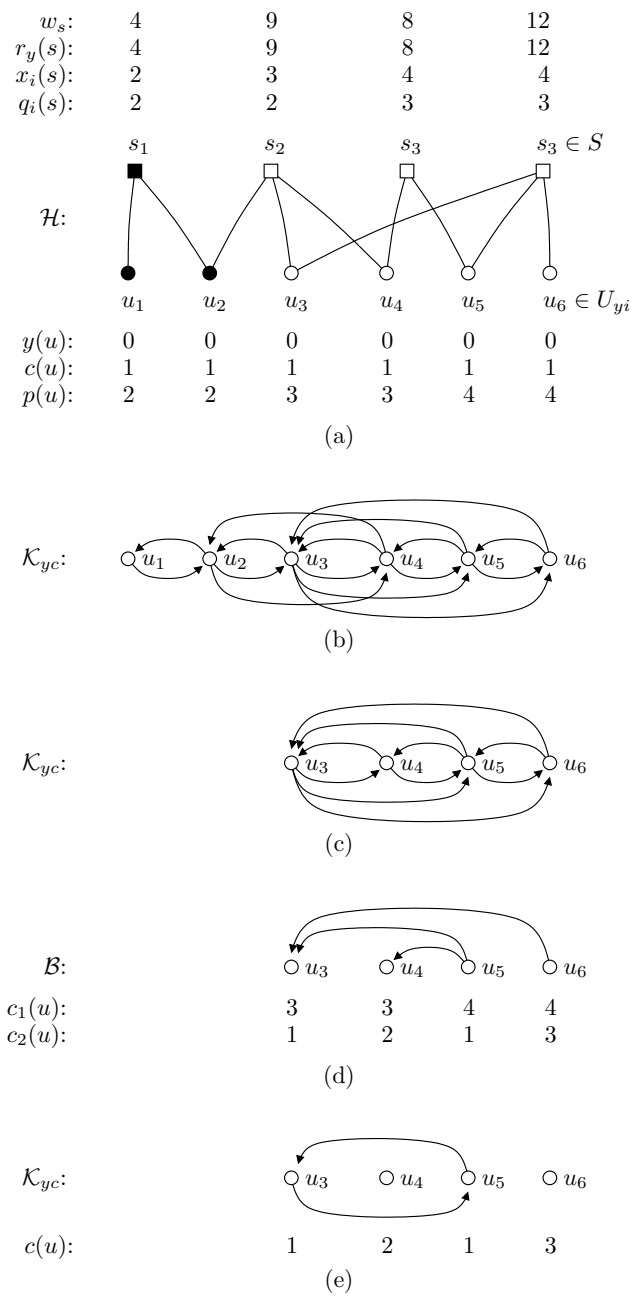

Figure 1: Fractional packing algorithm, the first iteration. Initially, all elements are in U_{y1}. (a) The saturation phase for colour $i = 1$; black nodes are newly saturated. (b) Directed multigraph K_{yc} before the saturation phase. (c) Graph K_{yc} after the saturation phase. Nodes u_1 and u_2 are saturated and hence removed from the graph. Nodes u_3 and u_4 were adjacent to the saturated node u_2, and hence their outdegree has decreased. (d) Subgraph $\mathcal{B} = \mathcal{B}_1$. Nodes u_5 and u_6 were not adjacent to any saturated node; hence they have a positive outdegree in $\mathcal{B}$. A χ-colouring c_1 of $\mathcal{B}$ and a weak 3-colouring c_2 of $\mathcal{B}$; for each node, at least one of the successors has a different colour in c_2. (e) Graph K_{yc} after the colouring phase. In comparison with figure (b), the outdegree of each node has decreased by at least one.

(i) Each $u \in U$ broadcasts $y(u)$. Each $s \in S$ computes $y[s]$ and $r_y(s)$.

(ii) Each $s \in S$ broadcasts $r_y(s)$. Each $u \in U$ determines whether it is saturated and whether it is in U_{yi}.

(iii) Each $u \in U$ broadcasts a bit indicating whether $u \in U_{yi}$. Each $s \in S$ computes the size of $U_{yi}(s)$. Each $s \in S'$ sets $x_i(s) \leftarrow r_y(s)/|U_{yi}(s)|$.

(iv) Each $s \in S'$ broadcasts $x_i(s)$. Each $u \in U_{yi}$ sets $p(u) \leftarrow \min\{x_i(s) : s \in N(u)\}$.

(v) Each $u \in U_{yi}$ broadcasts $p(u)$. Each $s \in S'$ sets $q_i(s) \leftarrow \min\{p(v) : v \in U_{yi}(s)\}$.

(vi) Each $u \in U_{yi}$ sets $y(u) \leftarrow y(u) + p(u)$.

These steps require $O(1)$ rounds per colour, in total $O(D)$ rounds. Clearly the edge packing y remains feasible after the saturation phase.

We will use the following observation in the colouring phase.

LEMMA 3. *Consider a node $u \in U_{yi}$ that was not saturated in step (vi). Let s be a neighbour of u with $p(u) = x_i(s)$, and let v be a neighbour of s with $q_i(s) = p(v)$. Then s is not saturated, $v \neq u$, and $p(u) > p(v)$.*

PROOF. If u is not saturated, the subset node s is not saturated either. Hence there was a neighbour $t \in U_{yi}(s)$ that increased $y(t)$ by less than $x_i(s)$, that is, $p(t) < x_i(s)$. This means that $p(v) = q_i(s) \leq p(t) < x_i(s) = p(u)$, and the claim follows. □

4.4 Colouring Phase

We begin by introducing some notation that facilitates the analysis of the colouring phase. For each colour i, let us define the set

$$B_i = \big\{(u,s,v) \in K : p(u) = x_i(s), \; q_i(s) = p(v), \; u, v \in U_{yi}\big\}$$

and the subgraph $\mathcal{B}_i = (U_{yi}, B_i)$ of the graph $\mathcal{K}_{yc}$. Each node $u \in U_{yi}$ is associated with a value $p(u)$, and Lemma 3 shows that these values are strictly decreasing in the direction of the edges. It follows that the subgraphs $\mathcal{B}_i$ are directed acyclic graphs. The subgraphs are by construction node-disjoint; let $\mathcal{B}$ be the union of these graphs.

Now consider a colour i and an element $u \in U_{yi}$ that was not saturated in any of the saturation phases. Then we can choose two nodes s and v as in Lemma 3; in particular, $v \neq u$. Note that before this iteration, both u and v were unsaturated and they had the same colour i; therefore (u,s,v) was an edge in $\mathcal{K}_{yc}$. If v became saturated during the saturation phases, the edge (u,s,v) is no longer part of $\mathcal{K}_{yc}$. Otherwise the edge (u,s,v) is in the set B_i. In summary, each element that was not saturated either loses at least one outgoing edge during the saturation phases, or has at least one outgoing edge in the subgraph $\mathcal{B}$.

Now we are ready to describe the distributed algorithm that implements the colouring phase. First, we use the rational numbers $p(u)$ and a reasoning similar to Lemma 2 to construct a χ-colouring c_1 of the subgraph $\mathcal{B}$, with

$$\chi = W(k!)^{(D+1)^2}.$$

Second, we use the algorithm that we describe in Section 4.5 to construct a *weak* 3-colouring c_2 of $\mathcal{B}$ in $O(\log^* \chi)$ rounds: each element $u \in U_{yi}$ with a positive outdegree in $\mathcal{B}$ has at least one successor v with a different colour. Put otherwise, there is a subgraph $\mathcal{B}'$ of $\mathcal{B}$ such that c_2 is a proper 3-colouring of $\mathcal{B}'$, and each node with a positive outdegree in $\mathcal{B}$ has a positive outdegree in $\mathcal{B}'$ as well.

Then we set $c_3(u) \leftarrow 3c(u) + c_2(u)$ for each node $u \in U_{yi}$ to construct an improper $3(D+1)$-colouring c_3 of $\mathcal{K}_{yc}$, with the following properties:

(a) If (u,s,v) is an edge of $\mathcal{B}'$ then $c_3(u) \neq c_3(v)$.

(b) If $(u,s,v) \in K$ and $c(u) \neq c(v)$ then $c_3(u) \neq c_3(v)$.

In other words, edges in $\mathcal{B}'$ become multicoloured, and multicoloured edges in $\mathcal{K}$ remain multicoloured. Finally, we use a trivial $O(D)$-time colour reduction algorithm to construct an improper $(D+1)$-colouring c_4 of $\mathcal{K}_{yc}$ with the same properties.

We set $c \leftarrow c_4$; after that, we have $e \notin K_{yc}$ for each edge e of $\mathcal{B}'$. Hence the outdegree of each node $u \in U_y$ in K_{yc} has decreased by at least one during the iteration: either we saturated a neighbour of u in one of the saturation phases, or we have at least one outgoing edge in $\mathcal{B}$. In the latter case, we also have at least one outgoing edge in $\mathcal{B}'$, which we multicoloured in the colouring phase. In the worst case, a node $u \in U$ loses only one outgoing edge during each iteration $j = 1, 2, \ldots, D$ and finally becomes saturated and removed from U_y during the last iteration $j = D+1$.

THEOREM 2. *There is a deterministic algorithm that finds a maximal fractional packing in $O(f^2k^2 + fk \log^* W)$ synchronous communication rounds.*

PROOF. Each iteration takes $O(D + \log^* \chi)$ communication rounds and there are $O(D)$ iterations. The claim follows from $\log^* \chi = O(\log^* D + \log^* W)$ and $D = O(fk)$. □

4.5 Weak Colour Reduction

In the colour reduction algorithm, each node $u \in U_y$ maintains a value $c'(u)$. Let $\mathcal{B}(u) \subseteq U_y$ be the set of successors of u in $\mathcal{B}$ and let $L(u) = \{c'(v) : v \in \mathcal{B}(u), c'(v) \neq c'(u)\}$. If $L(u) \neq \emptyset$, we define $\ell(u) = \min L(u)$. The graph $\mathcal{B}'$ is defined to consist of all edges (u,s,v) of $\mathcal{B}$ such that $L(u) \neq \emptyset$ and $\ell(u) = c'(v)$; see Figure 2 for an example. Let $\mathcal{B}'(u) \subseteq \mathcal{B}(u)$ be the set of successors of u in $\mathcal{B}'$.

We do not maintain $\mathcal{B}$ or $\mathcal{B}'$ explicitly. Nevertheless, each node $u \in U_y$ can compute the current values of $L(u)$ and $\ell(u)$ by using the following algorithm:

(i) Each $v \in U_y$ broadcasts the triplet

$$(c'(v), c(v), p(v)).$$

Let $M(s)$ be the set of messages received by $s \in S$.

(ii) Each $s \in S$ broadcasts the triplets

$$\big\{(c'(v), i, x_i(s)) : (c'(v), i, p(v)) \in M(s), \; p(v) = q_i(s)\big\}.$$

Let $M'(u)$ be the set of triplets received by $u \in U_y$.

(iii) Each $u \in U_y$ constructs

$$L(u) = \big\{c'(v) : (c'(v), i, x_i(s)) \in M'(u), \; c(u) = i, \; p(u) = x_i(s)\big\}.$$

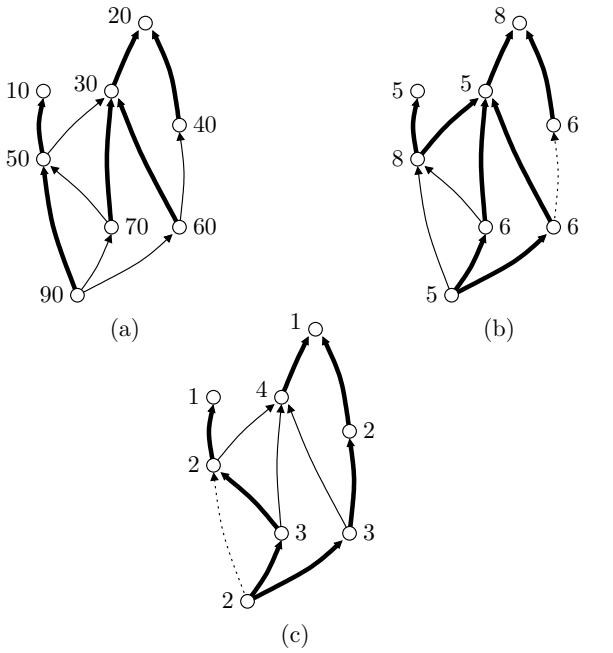

Figure 2: Weak colour reduction, two iterations. Arrows illustrate the directed acyclic graph $\mathcal{B}$, the numbers are the current values of $c'(u)$, and thick lines highlight the subgraph $\mathcal{B}'$. Dotted edges are not properly coloured; nevertheless, each node with a positive outdegree has at least one successor with a different colour.

Now we are ready to describe how the algorithm manipulates the colours c'. Initially, we set $c' \leftarrow c_1$. At this point, if $\mathcal{B}(u) \neq \emptyset$, we also have $L(u) \neq \emptyset$ and hence $\mathcal{B}'(u) \neq \emptyset$.

Then we apply repeatedly the Cole–Vishkin colour reduction technique for trees [8, 11]. In the Cole–Vishkin algorithm, each node is assumed to have at most one successor. Each node in parallel inspects the current colour of its successor and chooses a new colour; it is guaranteed that the successor chooses a different colour. In our case, each node $u \in U_y$ with $L(u) \neq \emptyset$ may have several successors in $\mathcal{B}'$, but all successors of u have the same colour $\ell(u)$. Hence we can simply proceed as if u had only one successor of colour $\ell(u)$; the Cole–Vishkin algorithm then guarantees that the new colour of u is different from the new colour of each $v \in \mathcal{B}'(u)$. In particular, if $L(u)$ was nonempty, it will be nonempty also after each colour reduction step. Hence we maintain the invariant that $\mathcal{B}(u) \neq \emptyset$ implies $\mathcal{B}'(u) \neq \emptyset$. After $O(\log^* \chi)$ iterations, we have reduced the number of colours in c' to 3.

5. VERTEX COVER IN THE BROADCAST MODEL

The edge packing algorithm from Section 3 assumed that we have a port numbering in the graph $\mathcal{G}$. Now we proceed to show how to find a maximal edge packing in the broadcast model. Naturally, we can represent an edge packing instance $(\mathcal{G}, w)$ as a fractional packing instance $(\mathcal{H}, w)$. We have $f = 2$ and $k = \Delta$; each node $v \in V$ is associated with a subset node $s(v) \in S$, and each edge $e \in E$ is associated with an element $u(e) \in U$. The algorithm $\mathcal{A}$ of Section 4 finds a maximal fractional packing in $\mathcal{H}$; hence it is sufficient to design an algorithm that simulates the execution of $\mathcal{A}$ in $\mathcal{H}$, using the broadcast model and the communication network $\mathcal{G}$. It should be noted that while the elements $u \in U$ are computational entities in $\mathcal{H}$, we do not have any internal state associated with an edge $e \in E$ in $\mathcal{G}$. Nevertheless, the simulation is possible (without increasing the number of communication rounds, but at the cost of increasing message complexity).

For each $v \in V$ and i, let $h(v, i)$ be the full history of all messages that the subset node $s(v)$ has sent during the communication rounds $1, 2, \ldots, i$ in $\mathcal{A}$; similarly, for each $e \in E$, let $h(e, i)$ be the full history of the element $u(e)$. We maintain the following invariant: after the communication round i, each node $v \in V$ knows $h(v, i)$. The base case $i = 1$ is trivial. For the general $i > 1$, we use an algorithm in which each $v \in V$ broadcasts $h(v, i-1)$. Hence for each edge $e = \{v, u\} \in E$ incident to v, the node v knows both $h(v, i-1)$ and $h(u, i-1)$, which constitute the full history of messages received by $u(e)$ before the round i. In particular, v can simulate $u(e)$ in $\mathcal{A}$ for i rounds to determine $h(e, i)$ for each incident $e \in E$, and then v can simulate $s(v)$ in $\mathcal{A}$ for 1 round to determine $h(v, i)$. Eventually, v can determine the multiset of the values $f(s(e))$ for incident edges e; in particular, v knows whether $\mathcal{A}$ saturates the subset node $s(v)$.

6. LOWER BOUNDS

In this section, we focus on the unweighted set cover problem. Let us fix the constants $k \geq 1$ and $f \geq 1$, and let $p = \min\{f, k\}$. We have already seen how to find a p-approximation of a minimum-size set cover: if $f < k$, we can apply the f-approximation algorithm from Section 4, and if $f \geq k$, we can apply the trivial k-approximation algorithm. Neither of the algorithms needs unique identifiers – port numbering is sufficient – and the running time is independent of the number of nodes.

If we assume the port-numbering model, no deterministic algorithm can achieve a better approximation ratio than p, regardless of the running time. To see this, consider the complete bipartite graph $K_{p,p}$, and choose the port numbers in a symmetric manner (see Figure 3 for an example in the case $p = 3$). Any deterministic distributed algorithm has to make the same decision for each subset node as their local views are identical. Hence the solution computed by the algorithm has size p, while there is an optimal solution of size 1.

If we had unique node identifiers, we could use them to break symmetry. However, if we focus on strictly local algorithms (with running time independent of the number of nodes), it turns out that we have the same lower bound p for the best possible approximation factor.

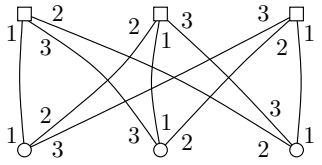

Figure 3: A symmetric set cover instance, $f = k = 3$.

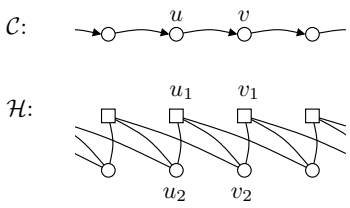

Figure 4: A local reduction from the problem of finding an independent set in a numbered directed cycle $\mathcal{C}$ to the problem of finding a set cover in the graph $\mathcal{H}$. In this example, $p = 3$.

In a *directed cycle*, each node has one incoming edge and one outgoing edge. A *numbered directed n-cycle* is a directed n-cycle in which each node is assigned a unique identifier from the set $\{1, 2, \ldots, n\}$. Czygrinow et al. [9] and Lenzen and Wattenhofer [24] show that a constant-time deterministic algorithm cannot find a large independent set in a numbered directed n-cycle; the following lemma is an adaptation of their results:

LEMMA 4 ([9, 24]). *Let $\mathcal{A}$ be a deterministic distributed algorithm that finds an independent set in any numbered directed cycle in $O(1)$ communication rounds. For any $\alpha > 1$ there exists an integer n_0 with the following property: for every $n \geq n_0$ there is a numbered directed n-cycle $\mathcal{C}$ in which $\mathcal{A}$ outputs an independent set with fewer than n/α nodes.*

We use this result and a simple local reduction to establish the lower bound. To reach a contradiction, let ε be a positive constant, and assume that there exists a deterministic local algorithm $\mathcal{A}'$ that computes a $(p - \varepsilon)$-approximation of a minimum set cover, provided that each subset node and each element is associated with a unique node identifier. The running time of $\mathcal{A}'$ must be independent of the number of nodes in the network; however, the algorithm $\mathcal{A}'$ and its running time may depend on the value of p.

We show that $\mathcal{A}'$ can be used to find a large independent set in a numbered directed cycle. Given a numbered directed n-cycle $\mathcal{C} = (V, E)$, with n divisible by p, construct a set cover instance $\mathcal{H} = (S \cup U, A)$ as follows. For each node $v \in V$ there is a subset node $v_1 \in S$ and an element $v_2 \in U$. There is an edge $\{u_1, v_2\} \in A$ iff the unique directed path from $u \in V$ to $v \in V$ in $\mathcal{C}$ has length at most $p - 1$. See Figure 4 for an illustration.

The unique node identifiers in $\mathcal{H}$ are inherited from $\mathcal{C}$ (e.g., if the identifier of $v \in V$ is i then the identifier of $v_1 \in S$ is $2i - 1$ and the identifier of $v_2 \in U$ is $2i$). Clearly, any computation in $\mathcal{H}$ can be simulated by a distributed algorithm in $\mathcal{C}$; the running time increases by a constant factor $O(p)$.

Now let us simulate the algorithm $\mathcal{A}'$ in $\mathcal{H}$, and let $C \subseteq S$ be the set cover computed by $\mathcal{A}'$. An optimal set cover C^* in $\mathcal{H}$ would take every pth subset node from S; hence $|C^*| = n/p$. By assumption, $|C| \leq (p - \varepsilon)|C^*| = (1 - \varepsilon/p)n$. This implies that the complement $S \setminus C$ has size at least $n\varepsilon/p$.

Let $X = \{v \in V : v_1 \in S \setminus C\}$; again, $|X| \geq n\varepsilon/p$. Let $\mathcal{C}'$ be the subgraph of $\mathcal{C}$ induced by the set X. The subgraph $\mathcal{C}'$ consists of paths (and isolated nodes which we consider as paths of length 0). Since C is a set cover in $\mathcal{H}$, there cannot be a path with p or more nodes in $\mathcal{C}'$. Thus there are at least $n\varepsilon/p^2$ connected components in the subgraph $\mathcal{C}'$. Let I consist of the nodes in $\mathcal{C}'$ with indegree 0 (i.e., the first node of each path). By construction, I is an independent set for $\mathcal{C}$, with $|I| \geq n\varepsilon/p^2$. Choosing $\alpha = p^2/\varepsilon$ and a sufficiently large n contradicts Lemma 4.

7. DISCUSSION

Usually, in the study of deterministic distributed algorithms, one takes for granted that each node has a unique identifier. There are only a few positive results related to anonymous networks. Mayer et al. [26] study local algorithms for weak colouring in anonymous networks with odd degrees; however, they have to assume not only a port numbering but also an orientation in order to break symmetry. Angluin [1], Attiya et al. [3], and Yamashita and Kameda [32] have studied computation in the port-numbering model, but many of their theorems are impossibility results, and the positive results focus on computability instead of time complexity. Boldi and Vigna [7] is a rare example of a work that explicitly considers the broadcast model, which is strictly weaker than the port-numbering model.

Our work shows that there are non-trivial graph problems that can be solved very efficiently in the broadcast model. Indeed, if we consider the approximability of the vertex cover problem, the broadcast model is surprisingly capable: our algorithm finds a 2-approximation in the broadcast model, and it is known that a deterministic local algorithm cannot find a better approximation even if we had unique node identifiers. Incidentally, finding a better constant factor approximation has been conjectured to be computationally hard even from the perspective of centralised algorithms [15].

While challenging to design, deterministic distributed algorithms in the broadcast model have many curious properties. If a deterministic distributed algorithm $\mathcal{A}$ uses the broadcast model, the output of $\mathcal{A}$ (together with the input) must have the same automorphisms as the graph $\mathcal{G}$ (and local inputs, if any): in a symmetric graph, the output must be symmetric. Moreover, we can apply the same reasoning to any covering graph of $\mathcal{G}$ [31, §5]. Consider, e.g., the case that $\mathcal{G}$ is the Frucht graph, which is 3-regular but has only the trivial automorphism. The universal covering graph of $\mathcal{G}$ is the infinite 3-regular tree $\mathcal{T}$. If we apply $\mathcal{A}$ to $\mathcal{T}$, then each node must produce the same output, as this is the only solution with the same automorphisms as $\mathcal{T}$. Because $\mathcal{A}$ cannot distinguish between $\mathcal{G}$ and $\mathcal{T}$, we conclude that if we apply $\mathcal{A}$ to $\mathcal{G}$, each node must produce the same output. In particular, if $\mathcal{A}$ finds a maximal edge packing, it must produce the solution $y(e) = 1/3$ for each edge e. None of this holds in the port-numbering model, as the port numbers may be used to break symmetry – for example, a prior algorithm [2] never sets $y(e) = 1/3$ in any graph.

In summary, distributed algorithms that use the broadcast model are able to produce highly symmetric solutions without explicitly identifying the symmetries in the input. This property may make such algorithms attractive also in a non-distributed setting.

8. ACKNOWLEDGEMENTS

Thanks to Christos Koufogiannakis, Valentin Polishchuk, and Joel Rybicki for discussions and comments. This work was supported in part by the Academy of Finland, Grants 116547 and 132380, by Helsinki Graduate School in Com-

puter Science and Engineering (Hecse), and by the Foundation of Nokia Corporation.

9. REFERENCES

[1] D. Angluin. Local and global properties in networks of processors. In *Proc. 12th Symposium on Theory of Computing (STOC 1980)*, pages 82–93. ACM Press, 1980.

[2] M. Åstrand, P. Floréen, V. Polishchuk, J. Rybicki, J. Suomela, and J. Uitto. A local 2-approximation algorithm for the vertex cover problem. In *Proc. 23rd Symposium on Distributed Computing (DISC 2009)*, volume 5805 of *LNCS*, pages 191–205. Springer, 2009.

[3] H. Attiya, M. Snir, and M. K. Warmuth. Computing on an anonymous ring. *Journal of the ACM*, 35(4):845–875, 1988.

[4] B. Awerbuch and M. Sipser. Dynamic networks are as fast as static networks. In *Proc. 29th Symposium on Foundations of Computer Science (FOCS 1988)*, pages 206–219. IEEE, 1988.

[5] B. Awerbuch and G. Varghese. Distributed program checking: a paradigm for building self-stabilizing distributed protocols. In *Proc. 32nd Symposium on Foundations of Computer Science (FOCS 1991)*, pages 258–267. IEEE, 1991.

[6] R. Bar-Yehuda and S. Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2(2):198–203, 1981.

[7] P. Boldi and S. Vigna. An effective characterization of computability in anonymous networks. In *Proc. 15th Symposium on Distributed Computing (DISC 2001)*, volume 2180 of *LNCS*, pages 33–47. Springer, 2001.

[8] R. Cole and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986.

[9] A. Czygrinow, M. Hańćkowiak, and W. Wawrzyniak. Fast distributed approximations in planar graphs. In *Proc. 22nd Symposium on Distributed Computing (DISC 2008)*, volume 5218 of *LNCS*, pages 78–92. Springer, 2008.

[10] S. Dolev. *Self-Stabilization*. The MIT Press, Cambridge, MA, 2000.

[11] A. V. Goldberg, S. A. Plotkin, and G. E. Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM Journal on Discrete Mathematics*, 1(4):434–446, 1988.

[12] F. Grandoni, J. Könemann, and A. Panconesi. Distributed weighted vertex cover via maximal matchings. *ACM Transactions on Algorithms*, 5(1):1–12, 2008.

[13] M. Hańćkowiak, M. Karoński, and A. Panconesi. On the distributed complexity of computing maximal matchings. *SIAM Journal on Discrete Mathematics*, 15(1):41–57, 2001.

[14] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11(3):555–556, 1982.

[15] S. Khot and O. Regev. Vertex cover might be hard to approximate to within $2-\epsilon$. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

[16] S. Khuller, U. Vishkin, and N. Young. A primal-dual parallel approximation technique applied to weighted set and vertex covers. *Journal of Algorithms*, 17(2):280–289, 1994.

[17] C. Koufogiannakis and N. E. Young. Distributed and parallel algorithms for weighted vertex cover and other covering problems. In *Proc. 28th Symposium on Principles of Distributed Computing (PODC 2009)*, pages 171–179. ACM Press, 2009.

[18] F. Kuhn. *The Price of Locality: Exploring the Complexity of Distributed Coordination Primitives*. PhD thesis, ETH Zürich, 2005.

[19] F. Kuhn, T. Moscibroda, and R. Wattenhofer. What cannot be computed locally! In *Proc. 23rd Symposium on Principles of Distributed Computing (PODC 2004)*, pages 300–309. ACM Press, 2004.

[20] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Fault-tolerant clustering in ad hoc and sensor networks. In *Proc. 26th International Conference on Distributed Computing Systems (ICDCS 2006)*. IEEE Computer Society Press, 2006.

[21] F. Kuhn, T. Moscibroda, and R. Wattenhofer. The price of being near-sighted. In *Proc. 17th Symposium on Discrete Algorithms (SODA 2006)*, pages 980–989. ACM Press, 2006.

[22] F. Kuhn and R. Wattenhofer. Constant-time distributed dominating set approximation. *Distributed Computing*, 17(4):303–310, 2005.

[23] C. Lenzen, J. Suomela, and R. Wattenhofer. Local algorithms: Self-stabilization on speed. In *Proc. 11th Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2009)*, volume 5873 of *LNCS*, pages 17–34. Springer, 2009.

[24] C. Lenzen and R. Wattenhofer. Leveraging Linial's locality limit. In *Proc. 22nd Symposium on Distributed Computing (DISC 2008)*, volume 5218 of *LNCS*, pages 394–407. Springer, 2008.

[25] N. Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[26] A. Mayer, M. Naor, and L. Stockmeyer. Local computations on static and dynamic graphs. In *Proc. 3rd Israel Symposium on the Theory of Computing and Systems (ISTCS 1995)*, pages 268–278. IEEE, 1995.

[27] M. Naor and L. Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995.

[28] A. Panconesi and R. Rizzi. Some simple distributed algorithms for sparse networks. *Distributed Computing*, 14(2):97–100, 2001.

[29] C. H. Papadimitriou and M. Yannakakis. Linear programming without the matrix. In *Proc. 25th Symposium on Theory of Computing (STOC 1993)*, pages 121–129. ACM Press, 1993.

[30] V. Polishchuk and J. Suomela. A simple local 3-approximation algorithm for vertex cover. *Information Processing Letters*, 109(12):642–645, 2009.

[31] J. Suomela. Survey of local algorithms, 2010. Manuscript submitted for publication.

[32] M. Yamashita and T. Kameda. Computing on anonymous networks: Part I – characterizing the solvable cases. *IEEE Transactions on Parallel and Distributed Systems*, 7(1):69–89, 1996.