Distributed Edge Packing

Juho Hirvonen

Helsinki November 19, 2012

MSc Thesis

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Juho Hirvonen

Työn nimi — Arbetets titel — Title

Distributed Edge Packing

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
MSc Thesis	November 19, 2012	76 pages

Tiivistelmä — Referat — Abstract

In this work we study a graph problem called edge packing in a distributed setting. An *edge packing* p is a function that associates a packing weight $p(e)$ with each edge e of a graph such that the sum of the weights of the edges incident to each node is at most one. The task is to maximise the total weight of p over all edges. We are interested in approximating a maximum edge packing and in finding maximal edge packings, that is, edge packings such that the weight of no edge can be increased.

We use the model of distributed computing known as the LOCAL model. A communication network is modelled as a graph, where nodes correspond to computers and edges correspond to direct communication links. All nodes start at the same time and they run the same algorithm. Computation proceeds in *synchronous communication rounds*, during each of which each node can send a message through each of its communication links, receive a message from each of its communication links, and then do unbounded local computation. When a node terminates the algorithm, it must produce a local output – in this case a packing weight for each incident edge. The local outputs of the nodes must together form a feasible global solution.

The *running time* of an algorithm is the number of steps it takes until all nodes have terminated and announced their outputs. In a typical distributed algorithm, the running time of an algorithm is a function of n, the size of the communication graph, and Δ, the maximum degree of the communication graph. In this work we are interested in deterministic algorithms that have a running time that is a function of Δ, but not of n.

In this work we will review an $O(\log \Delta)$-time constant-approximation algorithm for maximum edge packing, and an $O(\Delta)$-time algorithm for maximal edge packing. Maximal edge packing is an example of a problem where the best known algorithm has a running time that is linear-in-Δ. Other such problems include maximal matching and $(\Delta + 1)$-colouring. However, few matching lower bounds exist for these problems: by prior work it is known that finding a maximal edge packing requires time $\Omega(\log \Delta)$, leaving an exponential gap between the best known lower and upper bounds. Recently Hirvonen and Suomela (PODC 2012) showed a linear-in-Δ lower bound for maximal matching. This lower bound, however, applies only in weaker, anonymous models of computation. In this work we show a linear-in-Δ lower bound for maximal edge packing. It applies also in the stronger port numbering model with orientation.

Recently Göös et al. (PODC 2012) showed that for a large class of optimisation problems, the port numbering with orientation model is as powerful as a stronger, so called unique identifier model. An open question is if this result can applied to extend our lower bound to the unique identifier model.

Avainsanat — Nyckelord — Keywords
edge packing, matching, distributed algorithms, lower bounds

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction

Distributed problems arise when multiple autonomous computational entities must co-operate to solve a problem. In contrast with centralised computation, several new issues become interesting: how much communication is required? How to coordinate this communication? How to deal with communication failures? How to deal with hostile or failing parts of the system?

In this work, we are interested in the communication complexity of deterministic distributed algorithms solving classic graph problems, namely the *edge packing problem* which is also known as *fractional matching*. All algorithms are completely deterministic. In respect to models of distributed computing, we are not interested in fault tolerance, synchronisation, or security. The model of computation studied in this work, the LOCAL model [Lin92, Pel00], reflects this focus.

We model a distributed system as a graph. The nodes of a graph represent computers, and the edges between the nodes represent bidirectional communication links. Nodes communicate by sending messages through these links. The communication graph itself is also the problem instance; for example, the task is to find a maximal matching in the communication graph. In this, work we study the complexity of distributed algorithms measured in the number of communication rounds the algorithm needs until all nodes have declared an output. The running time of an algorithm is usually a function of the size of the graph, n, or the maximum degree of the graph, Δ. In general, the complexity of distributed algorithms as a function of n is quite well understood [Lin92, NS95]. However, if we look at the running time as a function of Δ, there is, in many cases, an exponential gap between the best known lower and upper bounds [KMW06, KMW10, PR01].

Maximal edge packing is a relaxed version of maximal matching. It is one of the problems, along with maximal matching and maximal independent set, for which the fastest known algorithm has a running time that is linear in the maximum degree of the graph Δ [PR01, ÅS10], and the best known lower bound is only polylogarithmic in Δ [KMW06, KMW10]. In graphs with a constant bound on the maximum degree, it is possible to find a maximal edge packing in a constant number of communication rounds.

In this work we study what is currently known about fast distributed computation of edge packings. We present fast algorithms for the general case and for some special cases. In addition we look at known lower bounds on the time complexity of

Problem	Model	Time		
Maximum (weighted) edge	P	$O(\log \Delta)$	Thm. 5.3	[Kuh05, KMW06]
packing, $O(1)$-approximation	ID	$\Omega(\log \Delta)$	Thm. 5.1	[KMW06, KMW10]
Maximal edge packing	P	$O(\Delta)$	Thm. 4.2	[ÅS10]
	P	$O(\Delta^2)$	Thm. 3.2	[ÅFP$^+$09]
	EC	$\Omega(\Delta)$	Thm. 6.1	**This work**
	PO	$\Omega(\Delta)$	Cor. 6.2	**This work**
Maximal weighted edge packing	P	$O(\Delta + \log^* W)$	Thm. 4.2	[ÅS10]
	EC	$\Omega(\Delta)$	Thm. 6.1	**This work**
	PO	$\Omega(\Delta)$	Cor. 6.2	**This work**
	PO	$\omega(1)$*	Thm. 4.1	[ÅFP$^+$09]
Maximal matching	ID	$O(\Delta + \log^* n)$	-	[PR01]
	ID	$O(\log^4 n)$	-	[HKP01]
	EC	$\Omega(\Delta)$	-	[HS12]

Table 1: A table summarising the results related to edge packings. Models are P: port numbering, PO: port numbering and orientation, ID: unique identifiers, and EC: edge colouring. For definitions of the first three, see Section 2.6, and for definition of the last, see Section 6. *: non-constant in W, the maximum weight of the graph.

finding maximal edge packings. The new contribution of this work is the first lower bound that is linear in the maximum degree of the graph on finding a maximal edge packing in an anonymous model of distributed computation. This lower bound uses techniques introduced by Hirvonen and Suomela [HS12]. Table 1 gives a summary of the results related to distributed edge packing that we will discuss in this work.

This work is structured as follows.

- In Section 2 we go through the necessary tools and concepts.
- In Section 3 we look at algorithms for the unweighted edge packing problem, and present an algorithm for finding a maximal edge packing in $O(\Delta^2)$ rounds.
- In Section 4 we look at algorithms and lower bounds for the weighted edge packing problem. We will see that any algorithm for the maximal edge packing in a weighted graph must have a running time that is a function of W, the maximum weight in the graph. Then we present an algorithm for the maximal edge packing problem with a running time $O(\Delta + \log^* W)$.
- In Section 5 we look at a simple approximation scheme for maximum edge packing [Mos06, KMW10] and a more involved approximation scheme for

general linear programs [Kuh05, KMW06, Kuh08] applied in the special case of approximating a maximum edge packing.

- In Section 6, we show that finding a maximal edge packing requires $\Omega(\Delta)$ rounds.
- Finally, in Section 7 we conclude and explore some avenues of future research.

2 Preliminaries

In this section we introduce some basic notation and the required graph theoretic concepts for the rest of the work.

2.1 Basic Notation

We use the following notation throughout this work. Denote by $[k] = \{1, 2, \ldots, k\}$ the set of k first natural numbers. All logarithms are in base 2, unless otherwise specified. The *iterated logarithm*, $\log^* n$, is defined as

$$\log^* n = \begin{cases} 0, & \text{if } n \leq 1 \\ 1 + \log^*(\log n), & \text{otherwise.} \end{cases}$$

Iterated logarithm of n is the number times one can take the logarithm of n until the result is less than or equal to 1. In practice the function grows very slowly. For example $\log^* n$ maps $2^0 = 1 \mapsto 0$, $2 \mapsto 1$, $2^2 = 4 \mapsto 2$, $2^4 = 16 \mapsto 3$, $2^{16} = 65536 \mapsto 4$, and $2^{65536} \mapsto 5$.

2.2 Graphs

A *simple graph* $G = (V, E)$ is a pair, where V is the set of *nodes* and E is the set of *edges*. An edge $e \in E$ is an unordered pair $e = \{u, v\}$, where $u, v \in V$ and $u \neq v$. Nodes u and v are called the *endpoints* of the edge e. Define $n = |V|$ and $m = |E|$ for the sizes of V and E. We use the shorthands V_G and E_G to denote the node and edge sets of graph G. A graph $H = (V_H, E_H)$ is a *subgraph* of graph G, if $V_H) \subseteq V_G$, and $E_H \subseteq E_G$. See Figure 1 for an illustration of graphs.

We say that a node v in graph G has *degree* $\deg_G(v) = |\{e \in E_G : v \in e\}|$. A graph G has maximum degree

$$\Delta(G) = \max_{v \in V_G} \deg_G(v).$$

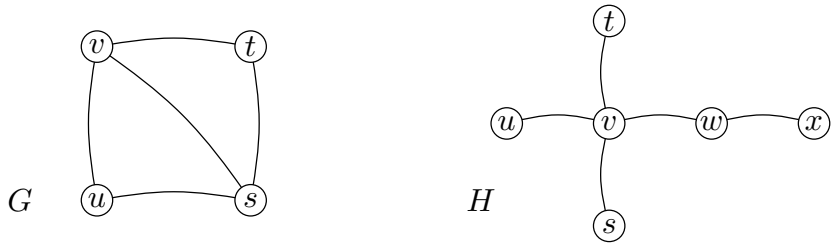

Figure 1: Two graphs, $G = (V, E)$ and $H = (V', E')$, where $V = \{s, t, u, v\}$, $V' = \{s, t, u, v, w, x\}$, $E = \big\{\{v, s\}, \{v, t\}, \{v, u\}, \{t, s\}, \{s, u\}\big\}$, and $E' = \big\{\{v, s\}, \{v, t\}, \{v, u\}, \{v, w\}, \{w, x\}\big\}$.

A family of graphs $\mathcal{F}$ has *bounded degree*, if there is a constant Δ, such that if $G \in \mathcal{F}$, then $\Delta(G) \leq \Delta$.

If a node v has $\deg_G(v) = 0$, we call node v *isolated*. Usually isolated nodes are not algorithmically interesting, and can be ignored. Therefore we assume that graphs do not have isolated nodes.

The set of *neighbours* of v is defined as

$$N_G(v) = \{u \in V_G : \{u, v\} \in E_G\}.$$

Nodes u and v are *adjacent* if they share an edge so that $\{u, v\} \in E$. Similarly two edges $e_1 = \{u, v\}$ and $e_2 = \{u', v'\}$ are adjacent if they have a common endpoint.

A *path* between two nodes is a graph $P = (V, E)$ such that $V = \{v_1, v_2, \ldots, v_{n+1}\}$, where each v_i is distinct, and $E = \{\{v_i, v_{i+1}\} : i \in [n]\}$. A path allows us to go from node v_1 to node v_n, and vice versa. The length of a path is n, the number of edges in it. There is always a trivial path from node to itself.

A *cycle* $C = (V, E)$ is a graph such that $V = \{c_1, c_2, \ldots, c_n\}$, where each c_i is distinct, and $E = \{\{c_i, c_{i+1}\} : i \in [n-1]\} \cup \{\{c_n, c_1\}\}$. A cycle has length n, the number of edges in it.

The *distance* of nodes u and v in graph G, $\mathrm{dist}_G(u, v)$, is the length of the shortest path between u and v. The distance of a node v and an edge $e = \{u, w\}$ is

$$\mathrm{dist}_G(v; e) = \min\{\mathrm{dist}_G(v, u), \mathrm{dist}_G(v, w)\} + 1.$$

Define

$$V_G[v, r] = \{u \in V_G : \mathrm{dist}_G(v, u) \leq r\},$$

that is, the set of nodes within distance r. Similarly, define

$$E_G[v, r] = \{e \in E_G : \mathrm{dist}_G(v; e) \leq r\}.$$

Now the local *r-neighbourhood* of v is a subgraph $G[v,r] = \big(V_G[v,r], E_G[v,r]\big)$. If there is a function $f\colon V_G \to Y$, we define $f[v,r]\colon V_G[v,r] \to Y$ by setting $f[v,r](u) = f(u)$ for each $u \in V_G[v,r]$. See Figure 2 for an illustration.

A *directed* graph is a pair $G = (V_G, E_G)$, where V_G is the set of nodes, as in undirected graphs, and E_G is a set of directed edges. A directed edge $e = (u,v)$ is an ordered pair. For an edge $e = (u,v)$, we say that e is directed from u to v. Define the *outdegree* of node v as

$$\deg_G^{\text{out}}(v) = |\{(v,u) : (v,u) \in E_G\}|,$$

and the *indegree* of a node v as

$$\deg_G^{\text{in}}(v) = |\{(u,v) : (u,v) \in E_G\}|.$$

If there is a simple, undirected graph $G = (V_G, E_G)$, then a directed graph $H = (V_H, E_H)$, with $V_H = V_G$, is an *orientation* of G if for each edge $\{u,v\} \in E_G$ there is exactly one edge, either (u,v) or (v,u), and no other edges in E_H.

If $X \subseteq V_G$ is a subset of nodes, then we define the subgraph of G induced by X as $H = (X, E(X))$, where

$$E(X) = \{\{u,v\} \in E_G : u \in X, v \in X\}.$$

Respectively, if $F \subseteq E_G$ is a subset of edges, then we define the subgraph of G induced by F as $H = (V(F), F)$, where

$$V(F) = \{v \in V_G : \exists u \text{ such that} \{v,u\} \in F\}.$$

There are several families of graphs that are of interest to us. A graph is *connected*, if there is a path between all pairs of nodes. A connected component of a graph G is a maximal subgraph $H = (V_H, E_H)$ of G such that H is connected. A finite graph $T = (V_T, E_T)$ is a *tree*, if

(D1) any two nodes of T are connected by exactly one simple path, or equivalently
(D2) T is connected and it has no cycles, or equivalently
(D3) T is connected and has exactly $n-1$ edges.

A *forest* F is a graph such that every connected component of F is a tree. A *cycle graph* is a graph that consists of a single cycle. A *pseudotree* is a connected graph that has at most one cycle. A *pseudoforest* S is a graph such that every connected

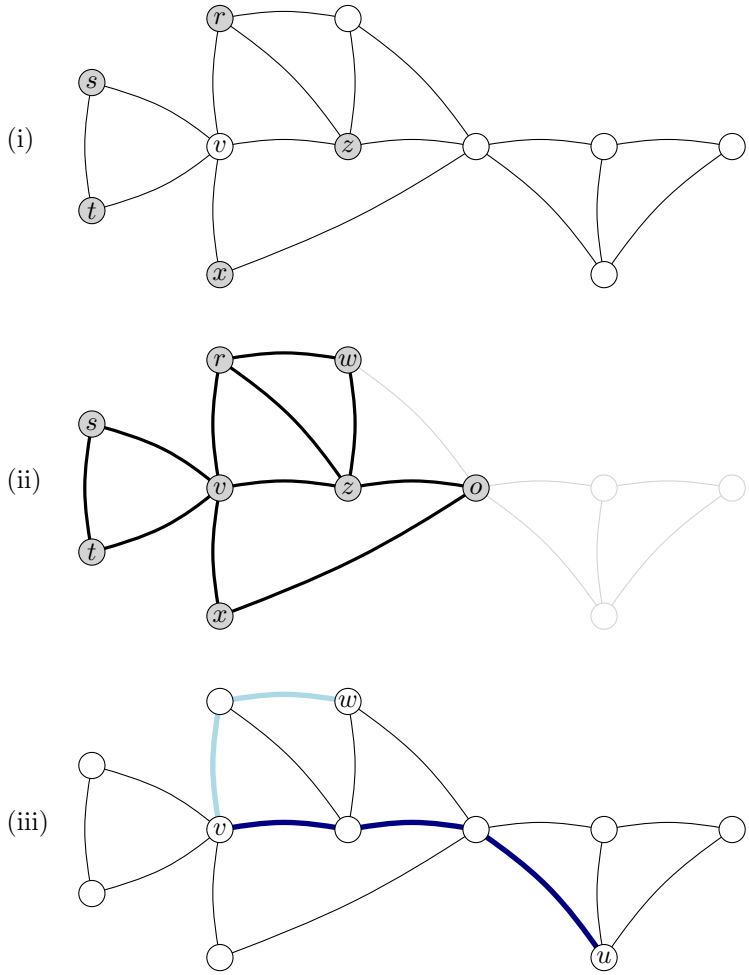

Figure 2: A graph G with maximum degree $\Delta(G) = 5$. Illustrated are (i) the set of neighbours of v is $N_G(v) = \{r, s, t, x, z\}$, (ii) nodes and edges that are within the radius 2 neighbourhood of v are $V[v, 2] = \{v, r, s, t, x, z, w, o\}$, and $E[v, 2] = \big\{\{v, r\}, \{v, s\}, \{v, t\}, \{v, x\}, \{v, z\}, \{r, w\}, \{r, z\}, \{w, z\}, \{z, o\}, \{o, x\}\big\}$, which form the 2-neighbourhood of v in G, in notation $G[v, 2] = (V[v, 2], E[v, 2])$, and (iii) shortest paths between u and v, and v and w with lengths 3 and 2, respectively, are marked with thick lines. This implies that $\text{dist}(u, v) = 3$ and $\text{dist}(v, w) = 2$.

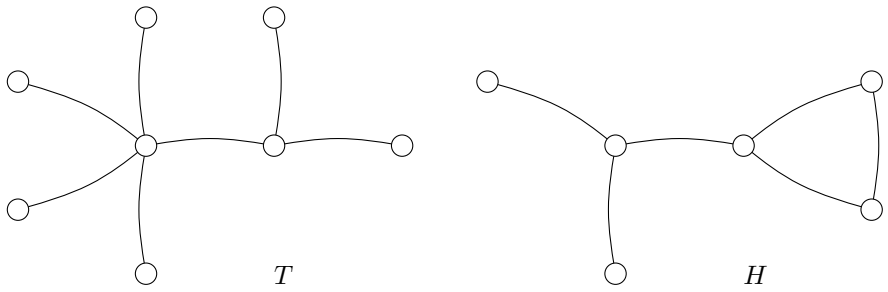

Figure 3: A tree T and a pseudotree H. As T has no cycles, it is bipartite. Pseudotree H has a cycle of length 3 and is therefore not bipartite.

component of S is a pseudotree. See Figure 3 for an illustration of a tree and a pseudotree.

A graph G is *bipartite*, if V_G can be partitioned into two sets, V_1 and V_2, such that $V_1 \cup V_2 = V_G$ and there is no edge between two nodes of V_1 or two nodes of V_2. A graph is bipartite if and only if it has no cycles of an odd length.

We say that a graph is *port-numbered*, if each node v has an ordering on its adjacent edges, or equivalently, has a bijective labelling of its incident edges with numbers from $\{1, 2, \ldots, \deg_G(v)\}$. If $\{u, v\} \in E_G$, let $\mathrm{port}(u, v)$ denote the *port number* of $\{u, v\}$ at u. We say that v is neighbour of u at port $\mathrm{port}(u, v)$. See Figure 4 for an illustration of a port-numbered graph.

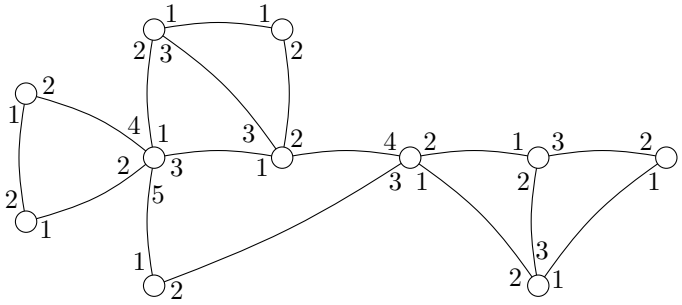

Figure 4: A port-numbered graph.

A function $f\colon V_G \to [k]$ is a *k-colouring* of G, if for each edge $\{u, v\} \in E_G$, $f(u) \neq f(v)$. If a graph has maximum degree Δ, a $(\Delta + 1)$-colouring always exists. A graph has a 2-colouring, and therefore is bipartite, if and only if there are no odd cycles in the graph. A *k-edge colouring* is a function $f\colon E_G \to [k]$ such that if $f(e) = f(e')$ for $e, e' \in E_G$, then $e \cap e' = \emptyset$. Figure 5 illustrates a node and an edge colouring.

Weighted graph is defined as a triple (V, E, w), where w is a weight function $w\colon V \to [W]$. We assume that the weights are bounded by some constant W.

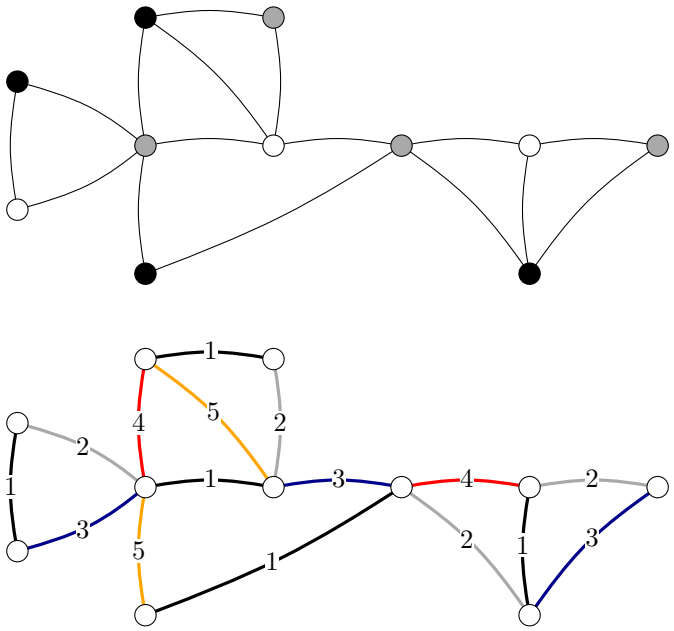

Figure 5: A graph with a 3-colouring, and the same graph with a 5-edge colouring. Both colourings are optimal; observe that there is an odd cycle in the graph, and that there is a node of degree 5.

We say that two graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic*, if there is a bijective map $\phi\colon V \to V'$ such that if $\{u, v\} \in E$, then $\{\phi(u), \phi(v)\} \in E'$, and vice versa. Isomorphisms preserve node degrees, and if the graphs are port-numbered, these are also preserved. If G and G' are isomorphic, we denote this by

$$G \simeq G'.$$

See Figure 6 for illustration.

A related notion is that of a *covering graph*. We say that a graph H is the covering graph of graph G if there is a surjective map $\phi\colon V_H \to V_G$ such that if $\{u, v\} \in E(H)$ then $\{\phi(u), \phi(v)\} \in E_G$, and vice versa. Covering maps also preserve port-numbering.

2.3 Graph Problems

In this section we give definitions for different classical graph problems, namely matchings, vertex covers and edge packings. See Figure 7 for an illustration of these.

Let $G = (V, E)$ be a simple, unweighted graph. A *matching* $M \subseteq E$ is a set of edges such that no two of them are adjacent. A node v is *matched* if there is an edge $e \in M$ such that $v \in e$. A matching is *maximal* if there are no two adjacent, unmatched

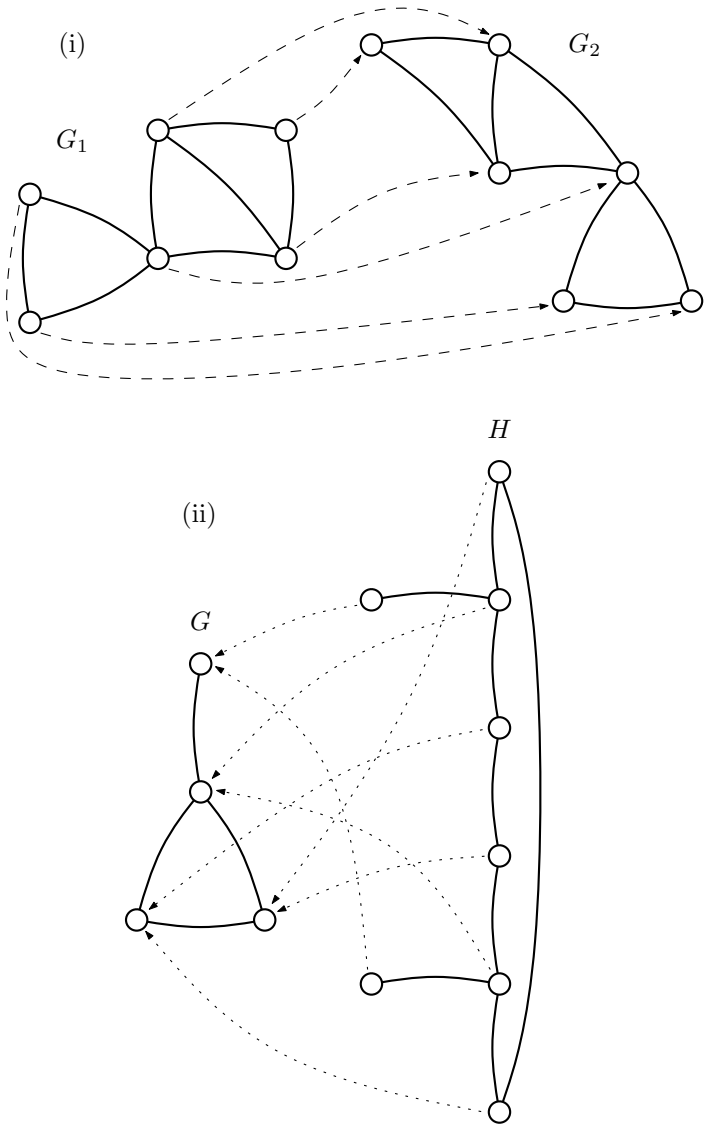

Figure 6: Two isomorphic graphs G_1 and G_2, and a graph G and its covering graph H. In (i), the dashed arrows indicate an isomorphism ϕ between the graphs G_1 and G_2. In (ii), the dashed arrows indicate a covering map from V_H to V_G.

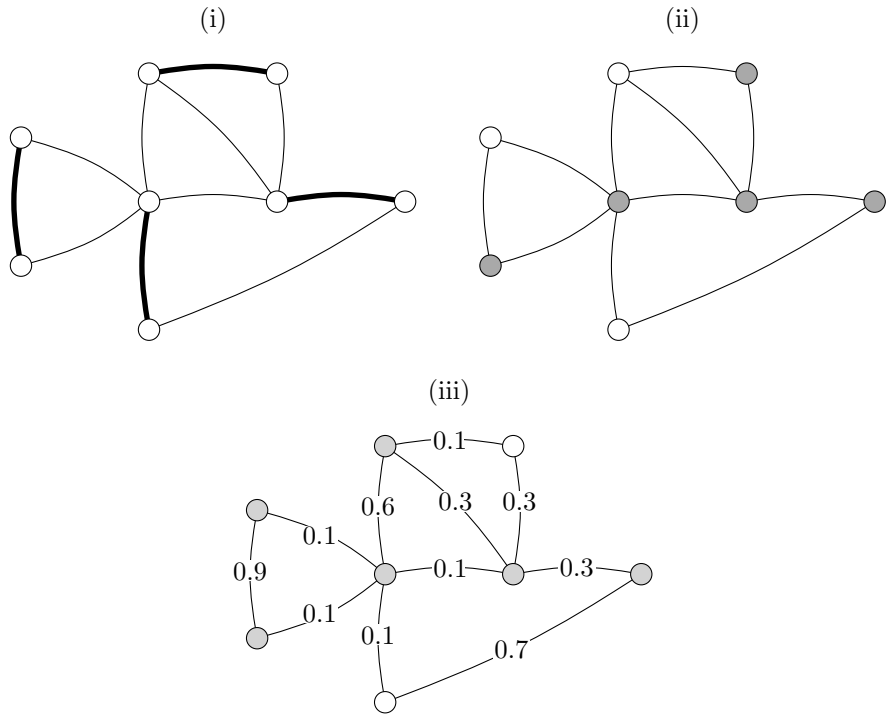

Figure 7: Three examples of classical graph problems; (i) a maximal matching, (ii) a minimum vertex cover, and (iii) a maximal edge packing. In (i), the thick black lines represent the matching, in (ii), the grey nodes represent the vertex cover, and in (iii), the numbers represent the packing p for each edge and the grey nodes represent the set of saturated nodes.

nodes. The *characteristic function* $c_M \colon E \to \{0,1\}$ associated with matching M is defined by

$$c_M(e) = \begin{cases} 1, & \text{if } e \in M \\ 0, & \text{if } e \notin M \end{cases}$$

An edge packing is a mapping $p \colon E \to [0,1]$ such that for each node $v \in V$

$$\sum_{e \in E : v \in e} p(e) \leq 1.$$

That is, the edge packing relates a value to each edge of the graph such that sum of the values related to edges adjacent to each node is less than or equal to 1. Define a shorthand

$$p[v] = \sum_{e \in E : v \in e} p(e).$$

The total weight of the edge packing is

$$p(E) = \sum_{e \in E} p(e).$$

We say that a node v is *saturated* in an edge packing, if

$$p[v] = 1.$$

An edge e is saturated if at least one of its endpoints is saturated. This means that the value $p(e)$ cannot be increased without violating the packing constraints. An edge packing is *maximal*, if all edges are saturated.

A *maximum edge packing* is an edge packing p^* such that there is no other edge packing with a larger total weight.

An edge packing is *perfect*, if all nodes are saturated.

The characteristic function of a matching in an unweighted graph is also an edge packing. In addition, the characteristic function of a maximal matching is a maximal edge packing. To see this, simply observe that

$$\sum_{e \in E : v \in e} c_M(e) = |\{e \in M : v \in e\}| \in \{0,1\}.$$

In addition, if node v is unmatched and thus $c_M[v] = 0$, all neighbours u of v are matched and have $c_M[u] = 1$. Thus each edge must have one endpoint that is saturated, and c_M is a maximal edge packing.

Weighted Edge Packing. Weighted edge packing is a generalisation of the unweighted edge packing. Let $G = (V, E, w)$ be a weighted graph. A weighted edge packing is a function $p\colon E \to [0, W)$ subject to

$$p[v] = \sum_{e \in E : v \in e} p(e) \leq w(v)$$

for all $v \in V$, where $w\colon v \to [W]$ for some constant W. Throughout this work, we assume that edge packings are unweighted unless otherwise specified.

Vertex Cover. In the minimum vertex cover problem the task is to minimize the size of a set of nodes $C \subseteq V$ while maintaining that each edge $e \in E$ must be incident to at least one node in C. We then say that C covers e. The minimum vertex cover problem is a classic NP-complete problem [GJ79].

A fractional vertex cover is a function $c\colon V \to [0, 1]$ such that for each edge $e = \{u, v\}$

$$c(u) + c(v) \geq 1.$$

In the weighted minimum vertex cover the task is to find a set of nodes $C \subseteq V$ minimising

$$\sum_{v \in C} w(v),$$

while satisfying for all edges $e = \{u, v\} \in E$ that $v \in C$ or $u \in C$.

Finally, a fractional minimum weighted vertex cover is a function $c\colon V \to [0, 1]$ minimising

$$\sum_{v \in V} c(v) w(v)$$

such that for each edge $e = \{u, v\}$

$$c(u) + c(v) \geq 1.$$

2.4 Linear Programming

In linear programming the task is to maximise or minimise a linear objective function subject to a set of linear constraints. A linear program is a problem that can be expressed as

$$\begin{aligned} \text{minimise} \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} \quad & \mathrm{A}\mathbf{x} \geq \mathbf{b} \\ \text{and} \quad & \mathbf{x} \geq 0, \end{aligned} \tag{1}$$

where A is a $(m \times n)$-matrix with known coefficients, $\mathbf{x}$ is an n-vector with the values to be computed, and $\mathbf{b}$ and $\mathbf{c}$ are m-vectors and n-vectors, respectively, with known coefficients. In this work we always assume that each element of A, $\mathbf{b}$, and $\mathbf{c}$ is non-negative.

We call this minimisation problem the *primal problem*. Each primal problem has a *dual problem*. It is a maximisation linear program that can be expressed as

$$\begin{aligned} \text{maximise} \quad & \mathbf{b}^{\mathrm{T}}\mathbf{y} \\ \text{subject to} \quad & \mathrm{A}^{\mathrm{T}}\mathbf{y} \leq \mathbf{c} \\ \text{and} \quad & \mathbf{y} \geq 0, \end{aligned} \tag{2}$$

where A, $\mathbf{b}$, and $\mathbf{c}$ are as in (1), and $\mathbf{y}$ is an m-vector with the values to be computed.

In our formulation, fractional vertex cover is a primal problem and edge packing is its dual problem. There is an important relationship between a linear program and its dual: any solution to the dual problem is a bound on the value of the original problem.

Theorem 2.1. *(Weak Duality of Linear Programming) If x is a feasible solution of a primal problem and y is a feasible solution of its dual problem, then it holds that*

$$\mathbf{b}^T\mathbf{y} \leq \mathbf{c}^T\mathbf{x}.$$

Proof. By the definitions of the two linear programs, it holds that

$$\mathbf{b}^T\mathbf{y} = \mathbf{y}^T\mathbf{b} \leq \mathbf{y}^T\mathrm{A}\mathbf{x} \leq \mathbf{c}^T\mathbf{x}.$$

The first inequality comes from the fact that $\mathrm{A}\mathbf{x} \geq \mathbf{b}$. The second inequality comes from the fact $\mathrm{A}^T\mathbf{y} \leq \mathbf{c}$, which is equivalent with $\mathbf{y}^T\mathrm{A} \leq \mathbf{c}^T$. □

2.5 Properties of Edge Packings

Unweighted edge packing and fractional vertex cover are the dual linear programs of each other. Basic proofs for approximating maximum edge packings and minimum vertex covers use the following duality.

Lemma 2.1. *Let p be an edge packing and C a vertex cover. Now $p(E) \leq w(C)$.*

Proof. We have that

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

□

It is well known that the integer version of maximal edge packing, maximal matching, gives a 2-approximation of a minimum vertex cover by taking all matched nodes. Observe that each matched edge has to be covered by one of its endpoints in a minimum vertex cover. In a maximal matching, these edges are covered by both of their endpoints. The matched nodes form a vertex cover, as if there were an uncovered edge, it could be added to the matching. In addition, it is known that a maximal matching is a 2-approximation of a maximum matching. Let M be a maximal matching and let M^* be a maximum matching. Now each for each edge $e \in M \setminus M^*$, there are at most two edges in $M^* \setminus M$. Therefore it must hold that

$$|M^* \setminus M| \leq |M \setminus M^*|.$$

Now we have that

$$|M^*| = |M^* \cap M| + |M^* \setminus M| \leq 2|M \cap M^*| + 2|M \setminus M^*| = 2|M|.$$

Bar-Yehuda and Even [BYE81] observed that a maximal edge packing gives a 2-approximation of a minimum vertex cover. It also holds for the weighted version. Also, it can be shown that a maximal edge packing is a 2-approximation of a maximum edge packing. Lemmas 2.2 and 2.3 give these results. The first proof follows Åstrand and Suomela [ÅS10].

Lemma 2.2. *If p is a maximal (weighted) edge packing, then the set of saturated nodes $C(p)$ in p is a 2-approximation of minimum vertex cover.*

Proof. The set $C(p)$ is a vertex cover by its maximality: if there was an edge $e = \{u, v\}$ such that neither of the nodes was saturated and thus not in $C(p)$, then $p(e)$ could be increased and p would not be a maximal edge packing.

Let C^* be the optimal vertex cover. Any vertex cover must include at least one endpoint of each edge. The set $C(p)$ includes at most both of the endpoints of an edge. Now the total weight is

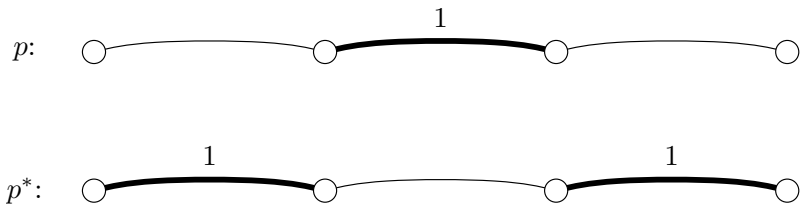

Figure 8: There is a maximal edge packing (which is a maximal matching) that is a 2-approximation of a maximum edge packing. A better approximation factor is not possible.

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

□

Lemma 2.3. *A maximal (weighted) edge packing is a 2-approximation of maximum edge packing.*

Proof. Let p and q be two maximal edge packings. By Lemma 2.2 it holds that the set of saturated nodes in p, $C(p)$, is a vertex cover. At most both endpoints of any edge are in the cover, so

$$w(C(p)) \leq 2 \sum_{e \in E} p(e).$$

By duality of linear programming, any solution to the dual problem of maximum edge packing, fractional vertex cover, is an upper bound on the value of the optimal solution for the edge packing problem. A vertex cover is also a fractional vertex cover and thus

$$\sum_{e \in E} q(e) \leq w(C(p)) \leq 2 \sum_{e \in E} p(e).$$

□

This result is tight. Let G be a path of length three, such that all nodes v have $w(v) = 1$. As in Figure 8, the optimal edge packing has total weight 2, but there is a maximal edge packing with total weight 1.

2.6 Distributed Computation

In this work we are interested in the model of distributed computation used by Linial [Lin92] and called the LOCAL model by Peleg [Pel00]. Our general view on

distributed computation is the following. A distributed system consists of computers and communication links between these computers. We model a distributed system as a simple graph, where nodes correspond to the computers, and edges correspond to communication links between those computers. Each node is a deterministic state machine. Nodes are able to communicate with their neighbours. Given a graph problem on the communication graph G itself, nodes possibly get some input, communicate with each other and then each node has to stop and announce its own local output. These outputs must together form a feasible solution to the problem.

A distributed system functions as follows. All nodes wake up at the same time and start executing the same algorithm. The algorithm is executed in synchronised communication rounds. During each round a node v can

(i) send a message to each of its neighbours,
(ii) receive a message from each of its neighbours, and
(iii) update its state.

The messages sent by a node during each round are determined as a function of the state of the node. After the node receives messages from its neighbours, it determines its next state as a function of its current state and the messages it received. A node announces its local output by going in to an end state that corresponds to a particular output.

We are interested in the situation where all computation and communication is completely deterministic. Nodes do not have access to any randomness. Nodes do not fail and communication does not fail. Synchronisation always works.

The running time of a distributed algorithm is measured in the number of synchronised communication rounds it takes for all nodes to finish and declare their output. We call a distributed algorithm a *local* algorithm if its running time is a constant. Note that in a bounded-degree graph, if an algorithm has a running time that is a function of the maximum degree Δ, but not of the size n, then the algorithm has constant running time. In this work we are mainly interested in local algorithms. For a survey of local algorithms, see Suomela [Suo11].

The amount of local computation done by the nodes is not limited. In addition, the size of the messages is not limited. Therefore nodes can flood the network with all information available to them every turn. In t rounds, the best a node can do is to gather all information about its radius-t neighbourhood. A local algorithm with running time t can therefore be seen as a mapping from the set of radius-t

neighbourhood isomorphism types to the set of possible outputs.

Common submodels of the LOCAL model are the port numbering model (P-model), and the unique identifier model (ID-model). In the port-numbering model, the input graph is port-numbered and no additional information is given. This model is *anonymous*, that is nodes have no names and are referred to by their port numbers. The port-numbering model is the standard anonymous model studied in the literature [Ang80, ASW88, YK88].

In contrast, in the ID-model each node is given a globally unique $O(\log n)$-bit identifier. This allows any computable problem on connected graphs to be solved in the model: simply gather all information to one node, locally compute the solution and distribute it back across the network. As we will see in the next section, this is not possible in the P-model.

Another anonymous model is the PO-model, where in addition to port numbering each edge is oriented from one endpoint to the other. This is simply additional information and does not affect the communication. Recent results established that for a large class of problems, the PO-model and the ID-model are equally capable from the perspective of local algorithms [GHS12].

2.7 Symmetry Breaking

In this section we show that it is not possible to find a maximal matching in the PO-model on a cycle. Therefore it is not possible to find a maximal matching in any standard anonymous model of distributed computing. It also means that to find maximal edge packings in the P-model, we have to use different techniques.

Theorem 2.2. *There is no distributed algorithm in the* PO*-model for finding a maximal matching on cycles.*

This result is directly related with the impossibility of breaking symmetry in some families of graphs in anonymous models. For example, consider the following graphs. In graph G, we have two identical nodes; the model is anonymous, both have degree exactly one and the only neighbour is connected to the nodes through port number one. Both nodes are running the same algorithm, so during the first round they will send send the same message, receive the same message and do the same local computation. By a simple induction it is easy to see that if the nodes stop, they must stop on the same round and produce the same output. Therefore it is not

possible to pick one of the nodes in graph G. In graph H this is no longer the case.

Now let us consider the problem of finding a maximal matching in a directed n-cycle $\mathcal{C} = (V, E)$. We can always construct it in the following fashion. Assign port numbers along the cycle so that if the outgoing port number at node v corresponding to edge $e = \{v, u\}$ is 1, then the incoming port at node u is 2. In addition, the edges can be directed along the cycle; in our case orient each edge from port 1 to port 2. See Figure 9 for an illustration.

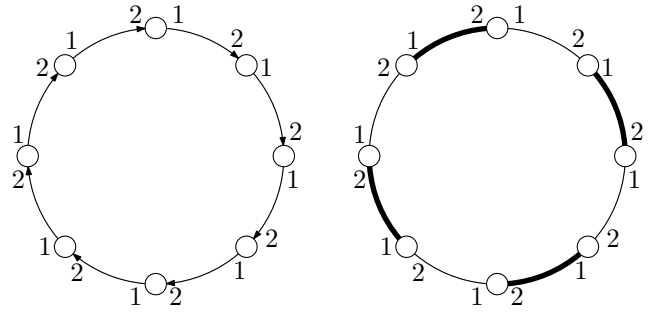

Figure 9: Two cycles: shown on the left is a cycle with a symmetric port-numbering and orientation, and on the right the optimal solution.

Now graph $\mathcal{C}$ is completely symmetric: each node has isomorphic radius T-neighbourhood for an arbitrary T. Again, the nodes start running the same algorithm. During the first round, each node sends the same message m_1 to port 1 and the same message m_2 to port 2. Then all nodes receive m_1 from port 2 and m_2 from port 1. Then nodes do local computation, in the same state with the same incoming messages. Again, a simple induction shows that after r rounds, nodes must still be in the same state. If the nodes stop, they must stop during the same round and announce the same output. We get the following useful lemma.

Lemma 2.4. *For any $n \geq 3$, there is a port-numbered, directed n-cycle $\mathcal{C}_n$ such that any algorithm in the* PO*-model must give the same output at every node.*

Now we can use this lemma to prove Theorem 2.2. Let $\mathcal{C}_n$ be an n-cycle as defined in Lemma 2.4. Each node must produce the same output. This output must encode a feasible maximal matching. The natural way to encode this output is to have a vector of length 2, where the first element encodes whether the edge with port number 1 is in the matching and the second element encodes whether the edge with port number 2 is in the matching. Now observe the following.

(a) $(0,1)$ and $(1,0)$ do not form an encoding of a matching in $\mathcal{C}_n$,
(b) $(1,1)$ is not a matching, and
(c) $(0,0)$ is not a maximal matching.

Thus for any $n \geq 3$, there is a port-numbered, directed n-cycle such that the only legal output for the algorithm is an empty matching. This proves Theorem 2.2.

While we have shown that it is impossible to find a maximal matching in any anonymous model, and we know that a maximal matching in an unweighted graph is also a maximal edge packing, we will show that it is possible to find a maximal edge packing in the models P and PO. Intuitively this is true, because the solution in the case of very symmetric graphs is trivial: in a d-regular graph we can simply output $p(e) = 1/d$ for each edge.

3 Maximal Edge Packing

We saw that a matching is in the unweighted case is also an edge packing. In addition, a maximal matching is a maximal edge packing. The best known algorithms for finding maximal matchings or good approximations of maximum matching are not local. Panconesi and Rizzi [PR01] gave an $O(\Delta + \log^* n)$ time algorithm for finding a maximal matching. For high degree graphs, there is an algorithm with running time $O(\log^4 n)$. Both algorithms also assume unique identifiers. In this work, we are mainly interested in algorithms that work in the P-model, as unique identifiers are not required to solve the edge packing problem.

In this section, we will see constant-time algorithms for the maximal edge packing in anonymous models. First, in Section 3.1 we present an algorithm for finding a maximal matching in a 2-coloured graph. This naturally gives us also a maximal edge packing. In Section 3.2 we will see an algorithm for maximal edge packing with running time $O(\Delta^2)$ that uses the algorithm from Section 3.1 as a subroutine.

3.1 Maximal Matching in 2-coloured Graphs

In an unweighted graph, a maximal matching is also a maximal edge packing. In a general graph it is not possible to find a maximal matching locally. The result by Linial [Lin92] shows that finding a maximal matching in a cycle with unique identifiers requires $\Omega(\log^* n)$ rounds. As we have seen, it is impossible to find a

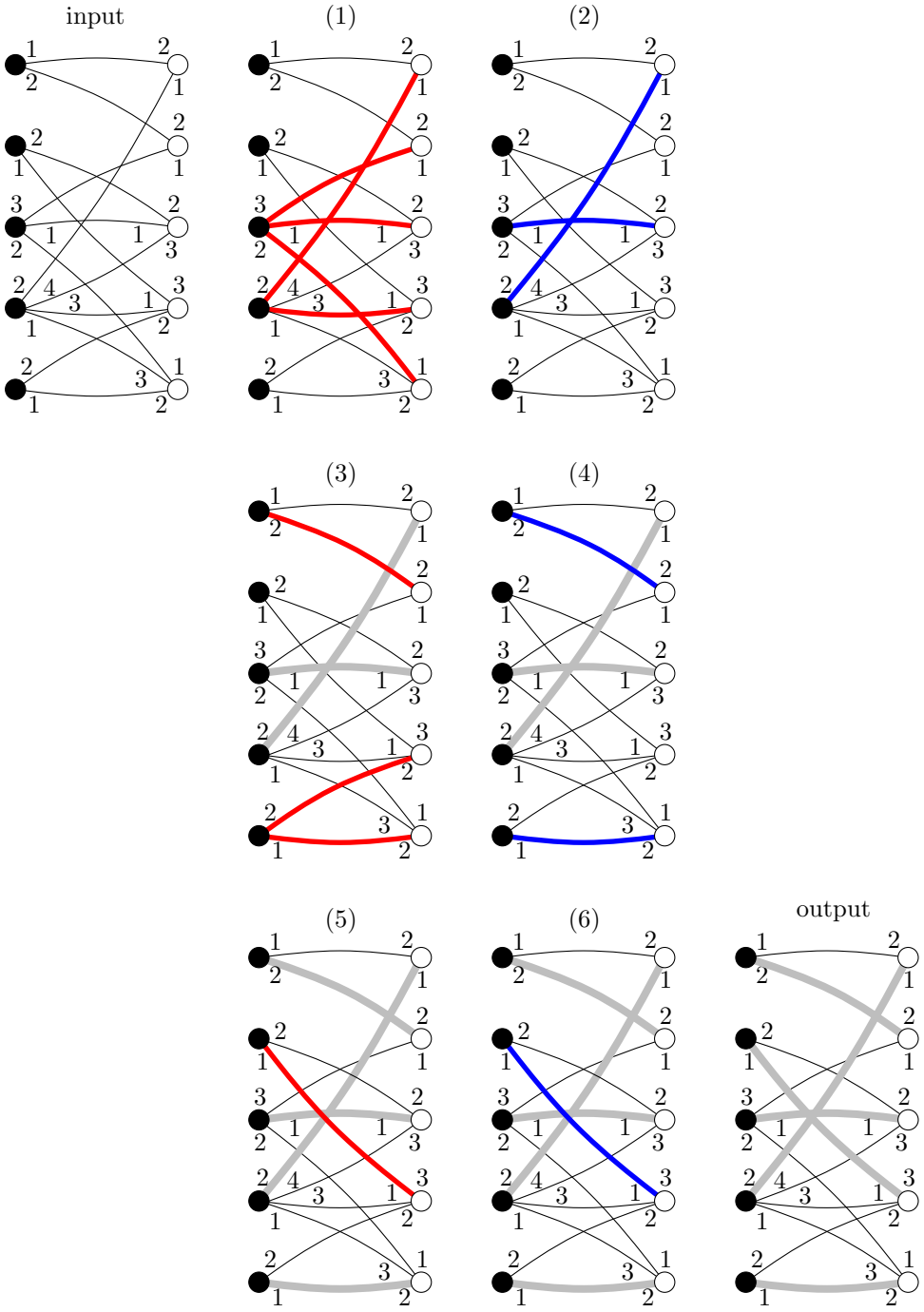

Figure 10: Simple proposal algorithm run in a 2-coloured graph. Proposals of the white nodes in odd rounds are shown by thick red edges, the acceptances of the black nodes in even rounds are shown by thick blue edges, and the matched edges are shown by the thickest gray edges.

maximal matching in an anonymous graph. However, if the graph is 2-coloured and has a bounded maximum degree there is a local algorithm for finding a maximal matching [HKP01].

Let $G = (V, E)$ be a graph and let f be a 2-colouring of G. Call colour 1 “white" and colour 2 “black". The algorithm relies completely on the symmetry breaking information provided by the 2-colouring and has different roles for the nodes of the two colours. The white nodes are active and send out proposals to the black nodes. The black nodes accept the first proposal, if any, and break ties with port numbers. Each node outputs the port number of the edge along which it is matched. If a node is not matched, it outputs $\perp$. See Figure 10 for an illustration of the execution of the algorithm.

When the algorithm starts, all nodes are active. During each round $2j$, where $j = 0, 1, 2, \ldots, \Delta$, each active white node v chooses one of the following actions; either

(i) v sends a message *‘proposal’* to its neighbour j, if v is unmatched and $j \leq \deg_G(v)$, or
(ii) v sends a message *‘matched’* to all of its neighbours and outputs $\text{port}(v, u)$ and stops, if v received a message *‘accept’* from node u during round $2j - 1$, or
(iii) v stops and outputs $\perp$, if $j > \deg_G(v)$ and v is unmatched.

During each round $2j$, each active black node u

(i) reads any *‘matched’* messages and marks those nodes as matched,
(ii) reads any *‘proposal’* messages and accepts the proposal of the node v that had the smallest port number $\text{port}(u, v)$ among those that sent proposals and
(iii) stops and outputs $\perp$ if all white neighbours have been matched.

During round $2j + 1$ each active black node u sends an *’accept’* message to node v, outputs $\text{port}(u, v)$ and stops, if u accepted the proposal of v during round $2j$.

Theorem 3.1. *In 2-coloured, bounded degree graphs there is an algorithm for computing a maximal matching in $O(\Delta)$ rounds.*

Proof. During the execution of the algorithm, each white node either sends a proposal to each of its neighbours or becomes matched. If a black node receives at least one proposal during the round, it accepts exactly one of those. Therefore if there is a

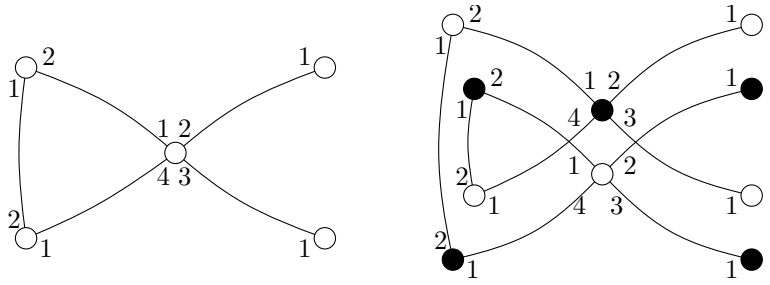

Figure 11: Graph G and its bipartite double cover.

white node that has not been matched after the algorithm has stopped, it does not have any unmatched black neighbours. The matching is maximal.

Let v be a white node with $\deg_G(v) = d$. If v remains unmatched, it sends a proposal to its neighbour d during round $2(d-1)$. If the proposal is accepted, v receives an accept message during round $2(d-1)+1$, sends out *'matched'* messages and stops during round $2d$. Now let u be a black node such that the neighbour of u with the largest degree, v has $\deg_G(v) = d'$. If u remains unmatched and does not receive any proposals at round $2(d'-1)$ then u must have received a *'matched'* message from each of its neighbours and it will stop.

After round 2Δ all nodes have stopped and produced output. Thus the algorithm runs in total for at most $2\Delta + 1$ rounds. □

Note that a maximal matching is a maximal edge packing and therefore a 2-approximation of the maximum edge packing.

3.2 An $O(\Delta^2)$ Time Algorithm for Maximal Edge Packing

Next we look at a local algorithm given by Åstrand et al. [ÅFP+09] for finding a 2-approximation of vertex cover in an unweighted graph. This algorithm finds a maximal edge packing and uses Lemma 2.2 to turn it into a 2-approximation of vertex cover. The algorithm is based on the use of *bipartite double covers* and the simple algorithm for finding a maximal matching in a 2-coloured graph from Section 3.1.

Bipartite double cover. Let $G = (V, E)$ be simple, bounded degree graph with a port numbering. The bipartite double cover of G is a 2-coloured covering graph of G. More specifically, the bipartite double cover $H(G) = G \times K_2 = (V_b \cup V_w, E')$ is constructed as follows. First, take two copies of each node v, a black copy v_b and

a white copy v_w. Now connect the white copy of u and the black copy of v if and only if $\{u, v\} \in E$. Both copies retain port numbers from the original graph, so if u is the neighbour of v at port 1 in G, then u_w and u_b are the neighbours of v_b and v_w, respectively, at port 1 in H. As a result, each edge $e = \{u, v\} \in E$ has two copies, $\{u_w, v_b\}$ and $\{u_b, v_w\}$ in $H(G)$. Figure 11 illustrates a bipartite double cover.

Formally there is a surjective graph homomorphism $\phi\colon H(G) \to G$ that maps the copies of each node to the original node, that is $\phi(v_b) = \phi(v_w) = v$. Now ϕ is clearly a homomorphism, as by definition if $\{u_b, v_w\} \in E'$ then $\{\phi(u_b), \phi(v_w)\} \in E$. Bipartite double covers retain many properties of the original graph, such as the maximum degree Δ. Distances are preserved to the following extent: if $\operatorname{dist}_G(u, v) = 2m - 1$ for any $m \in \mathbb{N}$, then $\operatorname{dist}_{H(G)}(u_b, v_w) = \operatorname{dist}_{H(G)}(u_w, v_b) = 2m - 1$. If $\operatorname{dist}_G(u, v) = 2m$, then $\operatorname{dist}_{H(G)}(u_b, v_b) = \operatorname{dist}_{H(G)}(u_w, v_w) = 2m$. A bipartite double cover of a connected graph is not necessarily connected. In addition, due to the bipartiteness, there are no odd cycles in $H(G)$.

If there is a local algorithm that works in 2-coloured graphs, such as the algorithm for finding a maximal matching from Section 3.1, then the computation of this algorithm in $H(G)$ can be simulated by nodes in G. The simulation scheme is simple. Each node v simulates its two virtual copies, v_b and v_w. If during round i the node v_b would send a message to the node u_w, then the node v sends the same message to node u and appends it with the information that it is from the black copy.

Bipartite double covers do not allow us to overcome the impossibility of breaking symmetry in an anonymous graph. It is not possible to find a maximal matching using only constant time in the original graph G. A maximal matching M in the bipartite double cover is not necessarily mapped by ϕ into a matching in G. Both the black and the white copy of node v could be matched to different neighbours u and u'. Then the corresponding image $\phi(M)$ in G has node v, which is matched to both $\phi(u)$ and $\phi(u')$. It can be, however, mapped into an edge packing. Let $e = \{u, v\} \in E$ and define $p\colon E \to [0, 1]$ such that

$$p(e) = \frac{f_M(\{u_b, v_w\}) + f_M(\{u_w, v_b\})}{2}. \tag{3}$$

Now we will define *half-integral* edge packings. Edge packing p is an example of such an edge packing.

Definition 1. An edge packing p is *half-integral* if for each $e \in E$ $p(e) \in \{0, \frac{1}{2}, 1\}$.

Half-integral edge packings include what we will call *almost saturating* edge packings.

Definition 2. A half-integral edge packing p is *almost saturating*, if the following conditions hold for all $v \in V$:

- If $p[v] = 0$, then for each neighbour u of v, we have that $p[u] = 1$.
- If $p[v] = 1/2$, then there is at least one neighbour u of v such that $p[u] = 1$.

See Figure 12 for an illustration of a maximal matching in the bipartite double cover, and the corresponding half-integral edge packing.

This means that each node v with $p[v] = 0$ or $p[v] = 1$ is saturated. These properties will be essential for the maximal edge packing algorithm. In fact, we will now show that the packing p, as given in (3), is almost saturating in G.

Lemma 3.1. *An almost saturating edge packing can be found in $O(\Delta)$ rounds by a deterministic distributed algorithm.*

Proof. If $G = (V, E)$ is a simple, bounded degree graph and H is its bipartite double cover, we will show that finding a maximal matching in H gives us an almost saturating edge packing in G by (3). We have already seen in Section 3.1 that it is possible to find a maximal matching in any 2-coloured graph in $O(\Delta)$ rounds.

First, let v be a node such that $p[v] = 0$. This means that neither v_b nor v_w is matched. Therefore each $u_w \in N_H(v_b)$ and $u_b \in N_H(v_w)$ must be matched, and we have that $p[v'] = 1$ for all $v' \in N_G(u)$.

Second, let v be a node such that $p[v] = 1/2$. Now one copy of v, either v_b or v_w, is matched and the other is not. Let v_b be the matched copy; the case when v_w is matched, is symmetric. Now each node u_b in $N_H(v_w)$ is matched, as otherwise the matching would not be maximal. Because v_b is matched, there is a node $u \in N_G(v)$ such that both its copies are matched in H. Therefore $p[u] = 1$. □

An Algorithm for Maximal Edge Packing. We first describe the algorithm informally. It consists of iterating the almost saturating edge packing algorithm repeatedly. Each time the algorithm is run, the resulting edge packing is added to the edge packing already calculated, and some nodes are discarded. We will show that each node that is not discarded has at least one neighbour which is discarded and thus the maximum degree of the remaining graph is strictly smaller than the maximum degree of the original graph. This will result in an empty graph after running the algorithm at most Δ times. To maintain a proper edge packing, the

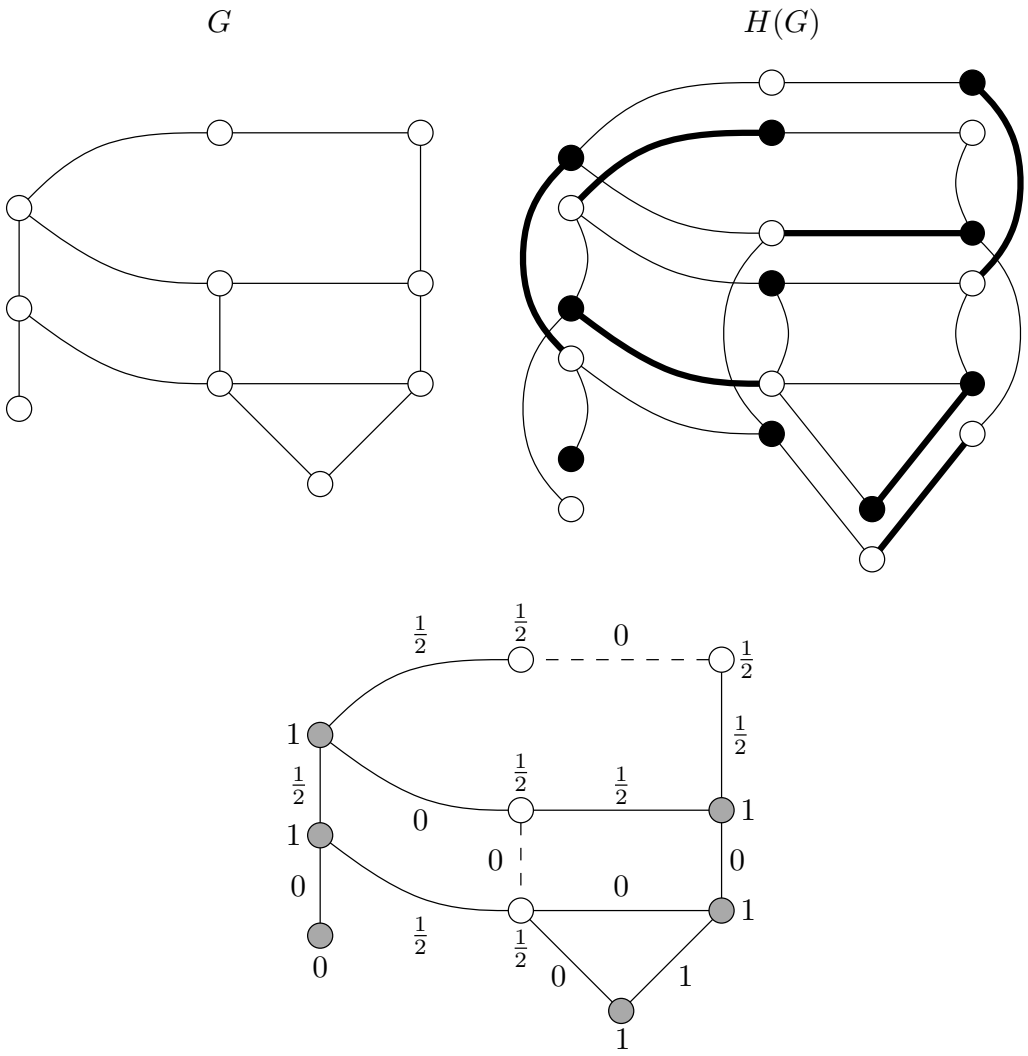

Figure 12: A graph G and its bipartite double cover H with a maximal matching in bold. Below, we have the same graph G with an almost saturating edge packing. Saturated nodes are in gray, and half-saturated edges are dashed. Each node has the weight $p[v]$ in p and each edge has the weight $p(e)$ in p written next to it.

packing weights added each round must be scaled properly. We will show that the resulting edge packing is maximal.

First we show that the algorithm terminates after constant number of iterations. At the beginning of each iteration $i \in \{0, 1, \ldots, \Delta\}$ we have a graph G_i. During each iteration the algorithm finds an almost saturating edge packing p_i in G_i. We say that an edge is *half-saturated* in an almost saturating edge packing p if both of its endpoints v and u have $p[v] = p[u] = 1/2$. The algorithm obtains G_{i+1} as the graph induced by the half-saturated edges in p_i. Let $e = \{u, v\}$ be a half-saturated edge in p_i. Thus both u and v are ,by the definition of almost saturating edge packing, adjacent to a saturated node in p_i. As we discard all saturated nodes to obtain G_{i+1}, it must hold that $\Delta(G_{i+1})$ is at most $\Delta(G_i) - 1$. If we iterate the algorithm $\Delta(G) + 1$ times, starting with $G_0 = G$, then $G_{\Delta(G)}$ is an empty graph.

Second we show how to construct the function p, which is a maximal edge packing. Start with an empty edge packing $p(e) = 0$ for all $e \in E$. Let $G_0 = G$. Now compute an almost saturating edge packing p_0 in G and let $p(e) \leftarrow p(e) + p_0(e)$. By the definition of almost saturating edge packings, the following hold:

- if $p_0[v] = 0$, each neighbour u of v must have $p[u] = 1$, and thus edges incident to v are saturated in p_0,
- if $p_0[v] = 1$, edges adjacent to v are saturated in p_0 by definition, and
- if $p_0[v] = 1/2$, there is at least one neighbour u of v such that $p[u] = 1$, and thus $\{u, v\}$ is saturated in p_0.

This means that if we discard each node v with $p_0[v] = 0$ or $p[v] = 1$, and each node v with $p_0[v] = 1/2$ such that each neighbour of v is saturated, the remaining graph must be induced by the half-saturated edges. We denote this graph by G_1. Now the original algorithm for finding almost saturating edge packings can be run again in G_1.

Now we generalise this idea. Define $p_i(e) = 0$, if $e \notin E_i$, that is one endpoint of edge e has been saturated in some p_j, $j < i$ and it is not in G_i. Otherwise $p_i(e)$ is the value of the edge packing for e in the ith iteration of the algorithm. Now construct the function

$$p = \sum_{i=0}^{\Delta-1} 2^{-i} p_i.$$

Lemma 3.2. *Function p computed by the algorithm is a maximal edge packing.*

Proof. We have to show that p is an edge packing and that it is maximal. First we show that p is an edge packing. Let v be a node in G_0 and k be the largest integer such that $v \in G_k$. Now for all iterations $i < k$, we know that by definition $p_i[v] = 1/2$. Finally, in iteration k, node v is saturated and $p_k[v] \leq 1$. Summing over all i, we obtain

$$p[v] = \sum_{i=0}^{k-1} 2^{-1}2^{-i} + p_k[v]^{-k} \leq \sum_{i=1}^{k} 2^{-i} + 2^{-k} = 1. \tag{4}$$

Therefore p is an edge packing. This also shows that if $p_k[v] = 1$ for some $k < \Delta$, then $p[v] = 1$.

Second, we show that each edge is saturated in p and therefore p is maximal. An edge $e = \{u, v\}$ is saturated, if at least one of its endpoints is saturated by p. Now let k be the largest integer such that $e \in G_k$. This means that either u or v must be saturated in p_k. If $p_k[u] = 1/2$, then it must be that $p_k[v] = 1$. This is due to the fact that if $p_k[v] = 0$, p_k would not be almost saturating and if $p_k[v] = 1/2$, e would be half-saturated and also in G_{k+1}. If $p_k[u] = 0$, then by definition of almost saturating edge packing $p_k[v] = 1$. Finally, if $k = \Delta - 1$, both u and v have degree 1 and therefore $p_k(e) = p_k[v] = p_k[u] = 1$.

Now by (4), function p is such that for each edge there is an endpoint v with $p[v] = 1$. Thus p is a maximal edge packing. □

Theorem 3.2. *There is a local deterministic distributed algorithm in the* P*-model for finding a maximal edge packing that runs in $O(\Delta^2)$ synchronous communication rounds.*

Proof. By Lemma 3.2 function p is a maximal edge packing. Function p is constructed in Δ iterations of the almost saturating edge packing algorithm. Each iteration $i \in \{0, 1, \ldots, \Delta - 1\}$ of the algorithm can be completed in $2(\Delta - 1)$ rounds. Thus in total the algorithm runs in $O(\Delta^2)$ communication rounds. □

4 Weighted Edge Packing

In this section we will see how adding weights to the nodes affects the maximal edge packing problem. As we have seen, in the unweighted case and in bounded-degree graphs it possible to solve the problem in constant time. However, in the case of the weighted maximal edge packing, this is no longer the case. In Section 4.1 we will

show that the running time of an algorithm for the weighted maximal edge packing can not be a function of only the maximum degree of the graph. In Section 4.3 we will see a fast algorithm for the weighted maximal edge packing problem, with a running time that is both a function of Δ, the maximum degree of the graph, and W, the maximum weight of a node in the graph. Now if both the weights and the maximum degree of the graph are bounded by a constant, this gives us a constant time algorithm for the weighted maximum edge packing problem.

4.1 Lower Bound

In this section we show the following theorem. Both the theorem and the proof are due to Åstrand et al. [ÅFP+09].

Theorem 4.1. *There is no local algorithm in the* ID*-model for the weighted maximal edge packing problem with running time $T = f(\Delta)$.*

The proof of the Theorem 4.1 uses Ramsey's Theorem [Ram30]. Ramsey's Theorem considers labellings of subsets of a larger set. Let $N \in \mathbb{N}$ and let $S = [N]$ and denote the set of k-subsets of S by

$$Y_S^{(k)} = \{Y : Y \subseteq S \text{ and } |Y| = k\}.$$

A c-labelling of a set $Y_S^{(k)}$ is a mapping

$$f \colon Y_S^{(k)} \to [c].$$

Ramsey's Theorem states that for all positive integers n, k, and c there exists a least integer, called a *Ramsey number*, denoted by $R_c(n; k)$, such that the following holds: if $N \geq R_c(n; k)$, then in any set S of size N, any c-labelling of $Y_S^{(k)}$ contains a *monochromatic* subset X of size at least n. That is, there is a set $X \subseteq S$ such that $f(Z) = f(Z')$ for each pair $Z, Z' \subseteq X$ with $|Z| = |Z'| = k$.

Ramsey's Theorem can be seen as a theorem about monochromatic cliques on complete graphs. Let $K_N = (V, E)$ be a complete graph on N nodes. Fix $k = 2$ so that each element of $Y_V^{(2)}$ corresponds to an edge in K_N. Now Ramsey's Theorem states that if $N \geq R_c(n; 2)$, then there is a clique of size n in K_N such that each edge of that clique has the same colour. If we set $c = 2$, and let one colour represent the existence of an edge, and the other the lack of an edge between a pair of nodes, Ramsey's Theorem can be used to show that a sufficiently large graph has either a large clique or a large independent set.

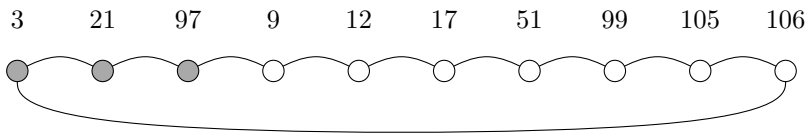

Figure 13: An example of a cycle construction C_X, where $N = \{3, 9, 12, 17, 21, 51, 97, 99, 105\}$ and $X = \{3, 21, 97\}$.

A Ramsey argument has also been used by Czygrinow et al. [CHW08] to argue that there is no constant-time, constant-approximation algorithm for the maximum independent set problem. We can assume that each node has unique entifiers and the proof still holds.

Now we are ready to show Theorem 4.1. Assume that A is a distributed algorithm for the weighted maximal edge packing problem with running time T. For simplicity, we assume without loss of generality that T is even.

To apply Ramsey's Theorem, let $n \gg T$ be a natural number, which will be fixed later. Let $N = \{1, 2, \ldots, n\}$. Set N corresponds to our set of nodes and each node $v \in N$ has unique identifier $\mathrm{id}(v) = v$. If $X \subseteq N$, with $|X| = \ell$, denote by $\mathcal{C}_X = (V, E_X)$ a n-cycle, which is constructed as follows. First, let

$$X = \{x_1, x_2, \ldots x_\ell\},$$

such that

$$x_1 < x_2 < \cdots < x_\ell.$$

Also, let

$$N \setminus X = \{x_{\ell+1}, x_{\ell+2}, \ldots, x_n\},$$

such that

$$x_{\ell+1} < x_{\ell+2} < \cdots < x_n.$$

Now let $V = N$, and let

$$E = \big\{\{x_i, x_{i+1}\} : i \in [n]\big\} \cup \big\{\{x_n, x_1\}\big\}.$$

For each node v, the unique identifier is $id(v) = v$ and the weight of v is $w(v) = v$. For an example of C_X, see Figure 13.

Next we will construct a labelling f of the $(2T+1)$-subsets of N as follows. For each $X \subset N$ with $|X| = 2T + 1$, construct the cycle $\mathcal{C}_X$ and run algorithm A in $\mathcal{C}_X$. Now

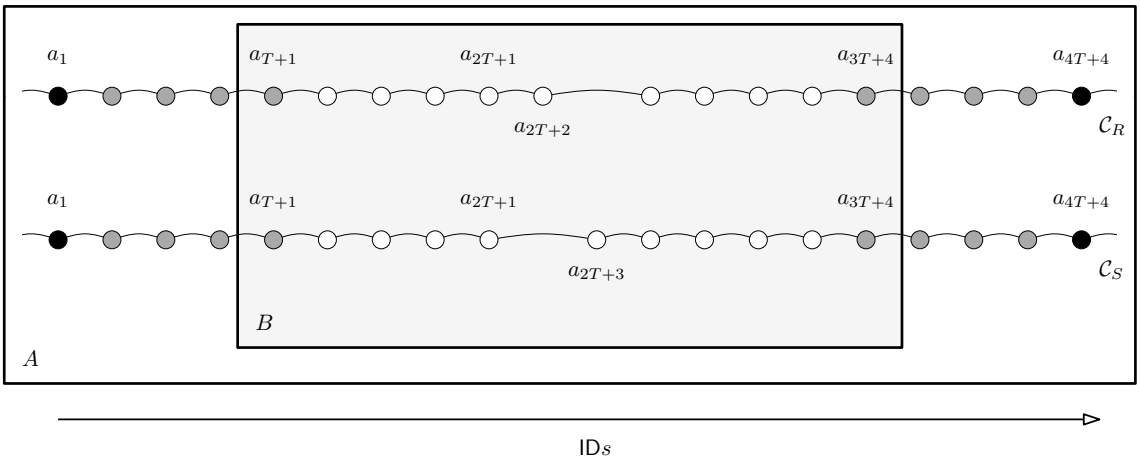

Figure 14: Two cycles, $\mathcal{C}_R$ and $\mathcal{C}_X$, used in our lower bound construction for $T = 4$. Nodes are ordered in the growing order of unique identifiers. The outer box shows the monochromatic subset A. We get sets R and S by removing a_{2T+3} and a_{2T+2}, respectively, from A. The subset B is another monochromatic subset such that the smallest node in a_{T+1} and the largest node a_{3T+4} in B are the first ones that cannot detect the difference between $\mathcal{C}_R$ and $\mathcal{C}_S$.

consider the central node x_{T+1}. Its local T-neighbourhood consists of only nodes in X and its local output depends only on its T-neighbourhood. Let $f(X) = 1$, if the node x_{T+1} is saturated by A and set $f(X) = 0$ otherwise.

By Ramsey's Theorem we can choose n such that there is a monochromatic set $A \subseteq N$, with $|A| = 4T + 4$, that is for all $X \subseteq A$, with $|X| = 2T + 1$, either all X have $f(X) = 0$, or all X have $f(X) = 1$. Now let

$$A = \{a_1, a_2, \ldots a_{4T+4}\},$$

where $a_1 < a_2 < \cdots < a_{4T+4}$. Let $R = A \setminus \{a_{2T+3}\}$ and $S = A \setminus \{a_{2T+2}\}$. Now we can use $\mathcal{C}_R$ and $\mathcal{C}_S$ to show that algorithm A can not compute a maximal edge packing in both. See Figure 14 for an illustration on $\mathcal{C}_R$ and $\mathcal{C}_S$.

Lemma 4.1. *Algorithm* A *cannot produce a maximal matching in both* $\mathcal{C}_R$ *and* $\mathcal{C}_S$.

Proof. We prove the Lemma by contradiction. Assume that A produces a maximal edge packing in both $\mathcal{C}_R$ and $\mathcal{C}_S$. We will use this maximality to construct a situation, where the nodes are saturated with growing weights along the cycle. Now nodes that are more than T hops away cannot detect if they are in $\mathcal{C}_R$ or $\mathcal{C}_S$ and this will lead into a contradiction.

Define $B = \{a_{T+1}, a_{T+2}, \ldots, a_{3T+4}\}$. Let $I \in \{R, S\}$ and let y_I be the edge packing computed by A in $\mathcal{C}_I$. First, we will show that every node $v \in I \cap B$ must be saturated in y_I. Let X be the set of nodes within distance T of v in $\mathcal{C}_I$. Now by construction, $X \subseteq A$, and either each node v must be saturated or node v is saturated. The T-neighbourhoods of v in $\mathcal{C}_I$ and $\mathcal{C}_X$ are equal. Thus each node $v \in I \cap B$ must have the same output both in $\mathcal{C}_I$ and in $\mathcal{C}_X$. Each node v is saturated if and only if the corresponding $f(X) = 1$. If each node in $I \cap B$ is not saturated, then y_I is not maximal, and thus every node in $I \cap B$ must be saturated.

Now the subset $I \cap B$ forms a path where the identifiers and weights of the nodes are strictly increasing. Let $x_I = a_{2T+2}$, if $I = R$ and $x_I = x_{2T+3}$, if $I = S$. We can represent the weights of the edge packing y_I as

$$\begin{aligned} y_I(\{a_{3T+3}, a_{3T+4}\}) &= a_{3T+3} - y_I(\{a_{3T+2}a_{3T+3}\}) \\ &= a_{3T+3} - a_{3T+2} + y_I(\{a_{3T+1}, a_{3T+2}\}) \\ &= \sum_{i=0}^{T-1} (-1)^i a_{3T+3-i} + x_I + \sum_{j=0}^{T-1} (-1)^{j+1} a_{2T+1-j} \\ &\quad - y_I(\{a_{T+1}, a_{T+2}\}). \end{aligned} \tag{5}$$

The weights of the nodes do not depend on whether $I = R$ or $I = S$. In addition, as the T-neighbourhoods of a_{T+1} in $\mathcal{C}_R$ and $\mathcal{C}_S$, and the T-neighbourhoods of a_{3T+4} in $\mathcal{C}_R$ and $\mathcal{C}_S$ are equal, their values do not depend on whether $I = R$ or $I = S$. We get

$$\begin{aligned} y_R(\{a_{3T+3}, a_{3T+4}\}) = y_S(\{a_{3T+3}, a_{3T+4}\}) \text{ and} \\ y_R(\{a_{T+1}, a_{T+2}\}) = y_S(\{a_{T+1}, a_{T+2}\}). \end{aligned} \tag{6}$$

This implies $x_{2T+2} = x_{2T+3}$, a contradiction. □

Theorem 4.1 follows, as the algorithm cannot produce a maximal edge packing in time T, for any constant T.

4.2 A simple algorithm for ε-maximal edge packing

Distributed edge packing has also been studied within other models than the local model. Khuller et al. [KVY94] present a parallel, simple algorithm for finding an *ε-maximal* edge packing. The algorithm uses a "safe" technique for incrementing the edge packing, also presented by Papadimitrou and Yannakis [PY93]. Note that this algorithm is not a constant-time algorithm. In Section 4.3 we will see another

algorithm, that uses this safe algorithm as a subroutine, but runs it only for a constant number of rounds.

An edge packing p is ε-maximal, for any $\varepsilon \geq 0$, if the set of nodes

$$C(p) = \{v \in V : p[v] \geq (1 - \varepsilon)w(v)\}$$

is a vertex cover. That is, the packing weight $p(e)$ of any edge cannot be increased by more than ε-fraction without violating the packing constraints.

While the algorithm is not introduced for the LOCAL model, it can be turned into a distributed algorithm in this model. The main idea of the algorithm is to distribute the free weight of each node equally among its unsaturated adjacent edges. The algorithm is safe in the sense that for each edge, the smaller proposal is accepted so that the edge packing always remains valid.

Now we define an algorithm we call ε-MAX in the LOCAL model. Let p be the edge packing at the beginning of round i and let $r_p(v) = w(v) - p[v]$ be the *residual weight* of node v. Let $G_p = (V_p, E_p)$ be the subgraph of G induced by the set of nodes $V_p = V \setminus C(p)$. Let $d_p(v)$ be the degree of v in the subgraph G_p. The main idea is that during each round, each node that is not in the partial vertex cover $C(p)$, sends as an offer to each of its neighbours also still not in $C(p)$ its *safe offer*

$$s_p(v) = \frac{w(v) - p[v]}{d_p(v)}.$$

Then for each edge $e = \{u, v\}$, the nodes select a new packing weight

$$p'(e) = p(e) + \min\{s_p(u), s_p(v)\}.$$

The algorithm itself does not record the packing weights of the edges, but of the nodes.

The algorithm proceeds as follows. First, set $p[v] \leftarrow 0$ for each $v \in V$, $G_p = G$, and $r(v) = w(v)$. During each odd round $2i - 1$

(i) each node $v \in V_p$ sends $s_p(v)$ to each of its neighbours,
(ii) each node $v \in V_p$ sets $p[v] \leftarrow p[v] + \min\{s_p(v), s_p(u)\}$ for each neighbour $u \in V_p$, and
(iii) each node $v \in V_p$ sets $r_p(v) = w(v) - p[v]$.

After each odd round $2i - 1$, nodes have just computed a new edge packing p'. Then, during round $2i$, each node $v \in V_p$, if $r_p(v) \leq \varepsilon w(v)$, adds itself to $C(p)$, sends a

message to each of its neighbours and stops. Then each node that did not stop, reconstructs the set of neighbours still in $V_{p'}$. We call a single odd round followed by a single even round an *iteration* of the algorithm ε-MAX. The algorithm always maintains the invariant that edge packing p is feasible after each two-round iteration of algorithm. To see this, observe that each node only offers a sum of weights that is equal to its residual weight, and for each edge the smaller weight of the offered is picked.

The problem with this algorithm is that the safe approach becomes very slow as the remaining weights decrease. In the next section we see an algorithm that overcomes this issue by running the iterated safe algorithm for a constant number of rounds and then switching to another algorithm.

4.3 Fast Algorithm for Maximal Edge Packing

Now we look at the fastest currently known algorithm for maximal edge packing [ÅS10]. It works also in the weighted problem. The running time is linear in Δ but depends also on the weights of the graph. By the proof in Section 4.1, this is unavoidable.

The algorithm consists of two phases. In the first phase, the algorithm uses the safe edge packing algorithm seen in Section 4.2. This is run for Δ rounds. It turns out that this edge packing is maximal in the areas of the graph that are regular. In those areas that are not saturated, computing the edge packing allows us to extract symmetry breaking information from the graph. This information can then be used to partition the graph into directed forests and 3-colour these forests. The second phase closely resembles the colouring algorithm by Goldberg et al. [GPS88] and the edge colouring algorithm by Panconesi and Rizzi [PR01]. In addition it employs colour reduction techniques due to Cole and Vishkin [CV86].

In the first phase the algorithm constructs an improper colouring c in a graph $G = (V, E, w)$. An edge $e = \{u, v\}$ is said to be *multicoloured* in c if $c(u) \neq c(v)$. It is assumed that the weights are integers from the set $\{1, 2, \ldots, W\}$.

4.3.1 Phase I

The first phase of the algorithm consists of running the safe algorithm from Section 4.2 for Δ rounds. In addition, the nodes construct an improper colouring c, which is a

sequence of rational numbers, such that the following holds after Δ rounds: each edge is either saturated or multicoloured.

At the start of the first phase, we have a weighted graph $G = (V, E, w)$ and an empty edge packing p. Each node v keeps track of its remaining weight $r(v) = w(v) - p[v]$ and colour $c(v)$. In the beginning, we set $r(v) = w(v)$ and $c(v) = \emptyset$. During the algorithm we maintain a subgraph $G(p, c)$ of G. It is the subgraph induced by the edge set

$$E(p, c) = \big\{\{u, v\} \in E : r(u) > 0,\ r(v) > 0,\ c(u) = c(v)\big\}.$$

At the start $G(p, c) = G$. Finally, $\deg_{G(p,c)}(v)$ is the degree of v in $G(p, c)$.

Now the algorithm proceeds by repeating the following three steps Δ times.

1. Each node sets its proposal $x(v)$ to
$$x(v) = \frac{r(v)}{\deg_G(v, p, c)}.$$
2. The edge packing is incremented for each edge $e = \{u, v\} \in E(p, c)$ by $\min\{x(u), x(v)\}$.
3. Each node $v \in V(p, c)$ sets its colour to $c(v) \leftarrow (c(v), x(v))$ and each node $u \in V \setminus V(p, c)$ sets its colour to $c(v) \leftarrow c(v) \cup \{1\}$.

Lemma 4.2. *During each round of execution of Phase I, the maximum degree of $G(p, c)$ decreases by at least one.*

Proof. Consider any node v with $\deg_{G(p,c)}(v) = \Delta(G(p, c))$. Now during the execution of the next round, there are two possibilities. Either

(i) for each edge $\{v, u\} \in E(p, c)$, $x(v) = \min\{x(v), x(u)\}$ and v is saturated or
(ii) there is an edge $\{v, u\} \in E(p, c)$ such that $x(v) > x(u)$; v is not saturated, but now $c(v) \neq c(u)$ and the degree of v in $G(p, c)$ is decreased by one. □

At the start of Phase I, it holds that $\Delta(G(p, c)) = \Delta(G)$. By Lemma 4.2, during each round of execution $\Delta(G(p, c))$ is decreased by at least one. As a result, after $\Delta(G)$ rounds, it must hold that $\Delta(G(p, c)) = 0$ and that $G(p, c)$ is empty.

Corollary 4.1. *After Phase I, each node $v \in V_G$ is either saturated, or there is a multicoloured edge $\{v, u\} \in E_G$.*

After Phase I, the colours of the nodes are sequences of rational numbers. To represent these colours as a mapping

$$c\colon V \to \{1, 2, \ldots, \chi\},$$

for some χ to be fixed later, we want the elements of the colours to be integral. To do this, we multiply the weights of the nodes by $(\Delta!)^\Delta$. We assume that all nodes know a constant Δ that is an upper bound for $\Delta(G)$.

Lemma 4.3. *For each $v \in V$ and each element q of $c(v)$, we have $0 < q \leq W$ and $q(\Delta!)^\Delta \in \mathbb{N}$.*

Proof. We show by induction that if we multiply each weight $w(v)$ by $(\Delta!)^k$, then after k iterations of the algorithm, the elements q of $c(v)$ are integral for each v.

Let $k \geq 1$. During the first round the offers of each node are integral, as

$$\frac{w(v)(\Delta!)}{\deg_G(v)} \in \mathbb{N}.$$

Because the offers are integral, the element q_1 of $c(v)$ is integral. In addition, the remaining weight $r(v)$ and packing weights $p(e)$ are integral. Specifically, the remaining weight $r(v)$ is of the form

$$(\Delta!)^{k-1} \cdot C,$$

where C is some integral number.

Now assume that $i < k$ and each q of $c(v)$, $r(v)$ and $p(e)$ for each $e \in E$ are integral after i iterations of the algorithm. In addition, assume that $r(v) = (\Delta!)^{k-i} \cdot C$ for some integral C. The degree of v in $G(p, c)$ is $\deg_{G(p,c)}(v) \in [\Delta]$. Because $i < k$ and $r(v) = (\Delta!)^{k-i}$, the offer $x(v)$ and thus the element q_i of $c(v)$ must be integral. Finally, the remaining weight will be integral, as the new remaining weight

$$r'(v) = r(v) - \sum_{\{u,v\} \in E(p,c)} \min\{x(u), x(v)\},$$

where

$$x(u) = (\Delta!)^{k-(i+1)} C'$$

for some C'. Thus the remaining weight is integral, and of the form

$$r'(v) = (\Delta!)^{k-(i+1)} C'',$$

for some integral C''. □

If we set $k = \Delta$, then the elements q of $c(v)$ are integral after the execution of the algorithm. Now we can define an injection from the possible values of c to $[\chi]$, where

$$\chi = (W(\Delta!)^{\Delta})^{\Delta}.$$

4.3.2 Phase II

By Corollary 4.1, after Phase I edges can be partitioned into two classes: saturated edges and edges that are not saturated but are multicoloured. Denote the latter class by $E(c) = \{\{u, v\} \in E_G : c(u) \neq c(v)\}$ and let $G(c)$ be the subgraph of G induced by $E(c)$. Because an edge is saturated if and only if one of its endpoints is saturated, there are only non-saturated edges and nodes in $G(c)$.

Now form an orientation of $G(c)$, denoted by $H(c)$ by setting

$$E(H(c)) = \{(u, v) : \{u, v\} \in E(c) \text{ and } c(u) < c(v)\},$$

that is, orient edges of $G(c)$ from the endpoint with a smaller colour to the endpoint with a higher colour. We have that $V(H(c)) = V(G(c))$.

Now observe that $H(c)$ is acyclic; if there was a directed cycle in $H(c)$, we could start at an arbitrary node and follow this cycle to the direction growing colours. Eventually we would arrive back to the starting node, which must have a colour larger than the previous node.

Now partition the edge set into $\Delta(G)$ forests, $F_1, F_2, \ldots F_{\Delta(G)}$. For each edge $(u, v) \in E(H(c))$, the node u assigns (u, v) to edge set E_i if and only if $\text{port}(u, v) = i$. Let $F_i = (V(H(c)), E_i)$ be the forest i. Now observe the following properties:

(P1) the outdegree of each node v in F_i is at most one: by definition the node assigns at most one outgoing edge into each forest and

(P2) each tree of each forest is rooted: there is a single root node such that each edge is oriented towards that node.

To show property (P2), combine (P1) and the fact that $H(c)$ is acyclic. Starting at any node v in any forest F_i, follow the outgoing edges until we reach a node with outdegree 0. If no such node exists, there must be a cycle in $H(c)$, which is a contradiction. Now the node reached from v is the root of the tree that contains v in F_i. To see that this root is unique, observe that if there were two root nodes in a single tree, then there would be a path between these nodes. This path however,

can not be oriented from either node to the other, so there must be a node with outdegree 2 on that path, which is a contradiction.

Next we will apply a techniques due to Cole and Vishkin [CV86], and Goldberg et al. [GPS88] to 3-colour each forest F_i. As we have a χ-colouring of $H(c)$, it takes $O(\log^* \chi)$ rounds to 3-colour each forest F_i in parallel.

Now let F_{ij} be the forest induced by the set of edges (u, v) such that u has colour j in F_i. Now each tree of F_{ij} is a rooted star, that is a rooted tree of height 1. This is by definition of F_{ij}: if there were two consecutive edges, (u, v) and (v, w) in F_{ij}, this would imply that the colour of u and v in F_i is equal and the colouring is not proper.

Now we saturate all edges greedily. Go through each combination of i and j, starting with $i, j = 1$. At each rooted star, if v is the root node of that star and L is the set of leaves (nodes of degree 1) in that star, each leaf $u \in L$ sends its remaining weight $r(u)$ to v. Then, if

$$r(v) \geq \sum_{u \in L} r(u),$$

set $p(\{u, v\}) = r(u)$, saturating every $u \in L$. Otherwise saturate v by setting

$$p(\{u, v\}) = \frac{r(u)}{\sum_{w \in L}(r(w)/r(v))}$$

for each $u \in L$.

Because the edge sets E_i partition $E(H(c))$, we have saturated all edges. This was achieved in $O(\Delta + \log^* \chi)$ rounds.

Next we show how the running time of the algorithm depends on Δ and W.

Theorem 4.2. *There is a deterministic algorithm in the* P*-model for finding a maximal edge packing in* $O(\Delta + \log^* W)$ *communication rounds.*

Proof. Phases I and II saturate all edges of the graph. Phase I takes 2Δ rounds. Phase II takes $O(\Delta + \log^* \chi)$ rounds. We show that $\log^* \chi = O(\log^* \Delta + \log^* W)$. Let $M = \max\{W, \Delta, 4\}$. By definition, $\chi = (W(\Delta!)^\Delta)^\Delta$. Next we show that

$\log\log\chi \leq 4\log M$:

$$\begin{aligned}
\log\log\chi &= \log\log\left(W(\Delta!)^{\Delta}\right)^{\Delta} \\
&= \log\left(\Delta\log(W(\Delta!)^{\Delta})\right) \\
&= \log\left(\Delta(\log W + \Delta\log\Delta!)\right) \\
&\leq \log\left(\Delta(\log W + \Delta^2\log\Delta)\right) \qquad (7) \\
&\leq \log\left(M(M^2+1)\log M\right) \\
&= \log\left((M^3+M)\log M\right) \\
&< \log M^4 = 4\log M.
\end{aligned}$$

Now by (7),

$$\log^*\chi \leq \log^* M^4 + 1 = O(\log^*\Delta + \log^* W). \qquad \square$$

Now if W is bounded by some constant, the algorithm computes a weighted maximal edge packing in constant time.

5 Approximation Scheme for Edge Packing

In this section we first present an approximation algorithm for the maximum edge packing problem [Mos06, KMW10]. Then we give a simplified version of a more involved general approximation scheme for general linear programs [Kuh05, KMW06]. Both are based on constructing simultaneously solutions to the primal and the dual linear programs, maintaining that the solutions have the same weight and dual problem is almost feasible. Then we make the dual solution feasible by dividing each variable by some small number. We bound this number and get approximation guarantees of these solutions based on the duality of linear programming.

Best known lower bound for edge packing is due to Kuhn et al. [KMW06, KMW10]. This same lower bound also applies for approximating a maximum matching. We will not prove this theorem.

Theorem 5.1 (Kuhn et al. [KMW10])**.** *Finding a constant approximation of a maximum edge packing requires at least*

$$\Omega(\log\Delta) \text{ and } \Omega(\sqrt{\log n})$$

communication rounds.

Both of the algorithms that we will see in this section produce a constant approximation of a maximum edge packing in time $O(\log\Delta)$.

5.1 Approximation Algorithm for Maximum Edge Packing

The first algorithm works roughly as follows. The algorithm computes a vertex cover (the characteristic function) x and an infeasible edge packing y. The algorithm does the following iteration for $O(k)$ rounds. During each round, nodes with many active neighbours join the vertex cover, and set their covering weight $x(v)$ to 1. We maintain the invariant that both the vertex cover x and the edge packing y have equal value. Therefore when a node v is added to the vertex cover, the packing weights of its adjacent edges are increased in total by 1. After the algorithm completes, x is a vertex cover. Dividing the packing weight $y(\{u, v\})$ of each edge by $\max\{y[u], y[v]\}$, we make y a feasible edge packing. If we can bound the weights of the edges by a constant, we can guarantee a constant approximations of both problems by the weak duality of linear programming.

We will show the following theorem.

Theorem 5.2. *There is a deterministic distributed algorithm that runs in $O(k)$ rounds and computes a $3 + \Delta^{1/k}$-approximation of minimum vertex cover and maximum edge packing.*

Using $k = O(\log \Delta)$ we get the following corollary.

Corollary 5.1. *There is a deterministic distributed algorithm that runs in $O(\log \Delta)$ rounds and computes a constant factor approximation of the minimum vertex cover and maximum edge packing.*

Proof. Assuming Theorem 5.2, and $k = \alpha \log \Delta$, we calculate

$$\Delta^{1/k} = \Delta^{1/(\alpha \log \Delta)} = 2^{1/\alpha}, \tag{8}$$

where β is some constant. The second equality of (8) follows from the fact that

$$\log \Delta^{1/(\alpha \log \Delta)} = \frac{1}{\alpha \log \Delta} \log \Delta = \frac{1}{\alpha} = \log 2^{1/\alpha}. \qquad \square$$

The specifics of the algorithm are as follows. Each node stores its own packing weight $x(v)$ and the packing weight $y(e)$ for each e of its adjacent edges. First we set $x(v) = 0$ and $y(e) = 0$ for each v and e.

The pseudocode for the algorithm is given by Algorithm 1. Each node v stores its own covering weight $x(v)$ and the packing weight $y(e)$ for each adjacent edge e. Let $E_G(v)$ be the set of edges adjacent to node v, and let $\tilde{E}_G(v, x)$ be the set of uncovered

Algorithm 1 A $(3 + \Delta^{1/k})$-approximation algorithm for minimum vertex cover and maximum edge packing for node v.

s

1: $x(v) \leftarrow 0$
2: **for each** $e \in E_G(v)$ set $y(e) \leftarrow 0$
3: **for** $\ell = k-1, k-2, \ldots, 0$ **do**
4: $\quad \tilde{\delta}(v,x) \leftarrow |\tilde{E}_G(v,x)|$
5: $\quad \tilde{\delta}_{\max}(v,x) \leftarrow \max_{u \in N_G(v)}\{\tilde{\delta}(u,x)\}$
6: $\quad$ **if** $\tilde{\delta}(v,x) \geq \tilde{\delta}_{\max}(v,x)^{\ell/(\ell+1)}$ **then**
7: $\quad\quad x(v) \leftarrow 1$
8: $\quad\quad$ **for each** $e \in \tilde{E}_G(v)$ set $y(e) \leftarrow y(e) + 1/\tilde{\delta}(v,x)$
9: $\quad$ **end if**
10: $\quad$ **if** $x(v) = 0$ and $y[v] \geq 1$ **then**
11: $\quad\quad x(v) \leftarrow 1$
12: $\quad\quad$ **for each** $e \in E_G(v)$ set $y(e) \leftarrow y(e) + y(e)/y[v]$
13: $\quad$ **end if**
14: **end for**
15: **for each** $e = \{u,v\} \in E_G(v)$ set $y(e) \leftarrow y(e)/\max\{y[u], y[v]\}$

edges adjacent to v. The dynamic degree $\tilde{\delta}(v,x)$ of node v is the size of $\tilde{E}_G(v,x)$. The maximum dynamic degree among the neighbours of v is denoted by $\tilde{\delta}_{\max}(v,x)$.

The algorithm is divided into k iterations of the same basic algorithm. During each iteration there are two ways a node can join the vertex cover. First, if during round ℓ node v has dynamic degree

$$\tilde{\delta}(v,x) \geq \left(\tilde{\delta}_{\max}(v,x)\right)^{\ell/(\ell+1)},$$

then the node joins the cover and the weight of each uncovered, adjacent edge is increased by $1/\tilde{\delta}(v,x)$. Second, if the node is not in the vertex cover, but the packing weights of the edges adjacent to it total at least 1, node joins the vertex cover and distributes the weight $x(v) = 1$, proportionally to the weights $y(e)$, to its adjacent edges. The algorithm computes a vertex cover, as in the last iteration, $\ell = 0$, on line 6 nodes that have dynamic degree $\tilde{\delta} \geq 1$, or equivalently have uncovered adjacent edges, join the vertex cover.

The initial edge packing computed by the algorithm is not feasible, however. To take

this into account, the algorithm maintains the invariant that

$$\sum_{v \in V_G} x(v) = \sum_{e \in E_G} y(e).$$

By LP duality, the dual solutions give bounds to the optimum values of each other. Therefore, as we will show that dividing each $y(e)$ by

$$\alpha = 3 + \Delta^{1/k}$$

yields a feasible solution to the edge packing problem, we also get an approximation bound on the solutions computed by the algorithm.

To prove the correctness of the algorithm, we first show that the dynamic degrees of the nodes must decrease as the algorithm is iterated. Then we show a bound on the packing weights of the edges adjacent to each node. Putting these together gives us Theorem 5.2.

Lemma 5.1. *At the beginning of each iteration ℓ for each $v \in V_G$ it holds that $\tilde{\delta}(v, x) \leq \Delta^{(\ell+1)/k}$.*

Proof. We show the claim by induction. During the first iteration, ℓ equals $k - 1$ and by definition it holds that $\tilde{\delta}(v, x) \leq \Delta$. Now we will show that if a node has $\tilde{\delta}(v, x) \geq \Delta^{\ell/k}$ during iteration ℓ, it will join the vertex cover. Nodes join on line 6 if

$$\tilde{\delta}(v, x) \geq \tilde{\delta}_{\max}(v, x)^{\ell/(\ell+1)}.$$

Therefore we must show that if a node does not join, then its dynamic degree is not too large, that is it holds that

$$\tilde{\delta}_{\max}(v, x)^{\ell/(\ell+1)} \leq \Delta^{\ell/k}$$

after iteration ℓ.

We know by induction assumption that $\tilde{\delta}(v, x) \leq \Delta^{(\ell+1)/k}$ at the beginning of the iteration. Now $\tilde{\delta}_{\max}(v, x)$ is the dynamic degree of some node u. Therefore it holds that $\tilde{\delta}_{\max}(v, x) \leq \Delta^{(\ell+1)/k}$. Now we get that

$$\tilde{\delta}_{\max}(v, x)^{\ell/(\ell+1)} \leq \Delta^{\frac{\ell+1}{k} \cdot \frac{\ell}{\ell+1}} = \Delta^{\ell/k}.$$

We have shown that all nodes with a large dynamic degree join the vertex cover and therefore the claim holds. □

Next we bound the sum of the increases of the packing function y during the execution of the algorithm.

Lemma 5.2. *After the execution of the algorithm, for each node $v \in V_G$ it holds that*

$$y[v] \leq 3 + \Delta^{1/k}.$$

Proof. We show the lemma by studying the different cases of a node can joining the vertex cover and how much the edge packing can increase in each case.

(i) If a node v does not join the vertex cover, then during the last iteration, when $\ell = 0$, the dynamic degree $\tilde{\delta}(v, x)$ must be 0 as otherwise the node would join the vertex cover. Therefore all neighbours of v have already joined the vertex cover before the last iteration and the packing weight cannot be increased by neighbouring nodes. Also, as any node v with $y[v] \geq 1$ joins the vertex cover, it must hold that $y[v] < 1$ for a node that does not join the vertex cover.

(ii) If a node v joins the vertex cover on line 7 during any iteration ℓ, it increases its adjacent packing weights by 1. Before this increase, it must have been that $y[v] < 1$. In addition neighbours of v may simultaneously join the vertex cover. By the joining condition on line 6, any such neighbour u has dynamic degree

$$\tilde{\delta}(u, x) \geq \tilde{\delta}_{\max}(u, x)^{\ell/(\ell+1)} \geq \tilde{\delta}(v, x)^{\ell/(\ell+1)}.$$

Let $\tilde{N}(v, x)$ denote the set of neighbours of v that have not joined the vertex cover. By Lemma 5.1 it holds that

$$\tilde{\delta}(v, x) \leq \Delta^{(\ell+1)/k}$$

and thereby the contribution of the active neighbours of v is bounded by

$$\sum_{u \in \tilde{N}(v,x)} \frac{1}{\tilde{\delta}(u, x)} \leq \frac{\tilde{\delta}(v, x)}{\tilde{\delta}(v, x)^{\ell/(\ell+1)}} = \tilde{\delta}(v, x)^{1/(\ell+1)} \leq \Delta^{1/k}.$$

After this iteration, no node can add weight to these edges by joining the vertex cover on line 7.

Finally, nodes that are not part of the vertex cover may join it on line 11. Any edge that has its packing weight increased in this way is only covered by at most one of its endpoints prior to the execution of line 11. Therefore these edges are only covered by v and the total weight of these edges before the other

endpoints join is at most 1. As the weights are increased proportionally, their sum is at most doubled during the execution of the algorithm. Therefore we get the total weight of the edges adjacent to a node v is at most $y[v] \leq 3 + \Delta^{1/k}$.

(iii) If a node v joins the vertex cover on the line 11, during iteration ℓ, it again has $y[v] < 1$ before the start of the iteration. Now as in the previous case, the adjacent, active nodes may join and increase the weights of edges adjacent to v during iteration ℓ by at most $\Delta^{1/k}$. The joining of v increases the weights by 1. Now any edge that was covered by v in this way was either already covered by another node u and will not have its weight increased or was not previously covered and therefore will have weight 0, which cannot be increased by nodes joining on line 11. This gives a total of increases at most $2 + \Delta^{1/k}$. □

Now to make the packing constraints feasible, the weight of each edge $\{u, v\}$ is divided by $\max\{y[u], y[v]\} \leq 3 + \Delta^{1/k}$. This, together with the weak duality of linear programming gives Theorem 5.2.

Proof of Theorem 5.2. We have to show that x is a characteristic function of a vertex cover, that y is an edge packing, that

$$\frac{\sum_{v \in V_G} x(v)}{\sum_{e \in E_G} y(e)} \leq 3 + \Delta^{1/k},$$

and finally that the running time of the algorithm is $O(k)$.

By construction, any node v can set $x(v) = 1$ only once during the execution of the algorithm. Now we have to show that each edge is covered. During the last iteration of the algorithm, any node v that has dynamic degree $\tilde{\delta}(v, x) \neq 0$ will join the vertex cover. Thus any otherwise uncovered edges will be covered during the last iteration and x is the characteristic function of a vertex cover.

By Lemma 5.2 for each $v \in V_G$ the sum of the adjacent packing weights is at most $3 + \Delta^{1/k}$. On line 15 the packing weight $y(e)$ of each edge $e = \{u, v\}$ is divided by $\max\{y[u], y[v]\}$. This will make the packing constraints feasible and y is therefore a feasible edge packing. By the weak duality of linear programming y is now a $(3 + \Delta^{1/k})$-approximation of the maximum edge packing.

The running time of the algorithm is $O(k)$, as each iteration of the main loop can be done in a constant number of communication rounds. Assume that at the beginning of the round the nodes know their own dynamic degree $\tilde{\delta}(v, x)$. First the nodes communicate this to their neighbours, which allows them to compute the local

maximum dynamic degree $\tilde{\delta}_{\max}(v, x)$. Second, nodes may join the vertex cover on line 7 and communicate the increases of y to their neighbours before line 10. Third, nodes may join the vertex cover on line 11 and communicate the increase of y to their neighbours. After the third communication round each node knows the packing weights of its adjacent edges and its own dynamic degree. Thus the algorithm runs in total in $3k$ communication rounds. □

5.2 General Linear Program Approximation Scheme

In this section we present an approximation scheme for general linear programs adapted to the special case of edge packing. Our presentation follows that of Kuhn et al. [KMW06, Kuh05]. The algorithm computes a constant approximation of the primal problem, the minimum fractional vertex cover, and of the dual problem, the maximum edge packing. For simplicity and readability, we assume that the maximum degree Δ is known by the algorithm. This requirement can be removed [Kuh05, KMW06].

Let $k_p > 1$ and $k_d > 1$ be parameters for our algorithm A. We start with the main theorem.

Theorem 5.3. *There is a distributed algorithm* A *that approximates minimum fractional vertex cover and maximum edge packing by a factor*

$$\Delta^{4/k_p} \max\{\Delta^{1/k_p}, 2^{1/k_d}\}.$$

The running time of algorithm A *is*

$$O\left(k_p k_d \left(1 + \frac{1}{\Delta^{1/k_p} - 1}\right)\left(1 + \frac{k_p}{\Delta^{1/k_p} \log \Delta}\right)\right).$$

This leads directly to the following corollary.

Corollary 5.2. *There is deterministic distributed algorithm that finds a constant approximation of maximum edge packing with running time* $O(\log \Delta)$.

Proof. Let $k_p = \beta_1 \log \Delta$ and $k_d = \beta_2 \log 2 = \beta_2$. By Theorem 5.3 we get that the algorithm computes an approximation of factor

$$\Delta^{4/(\beta_1 \log \Delta)} \Delta^{1/(\beta_1 \log \Delta)} = e^{4/\beta_1} e^{1/\beta_1},$$

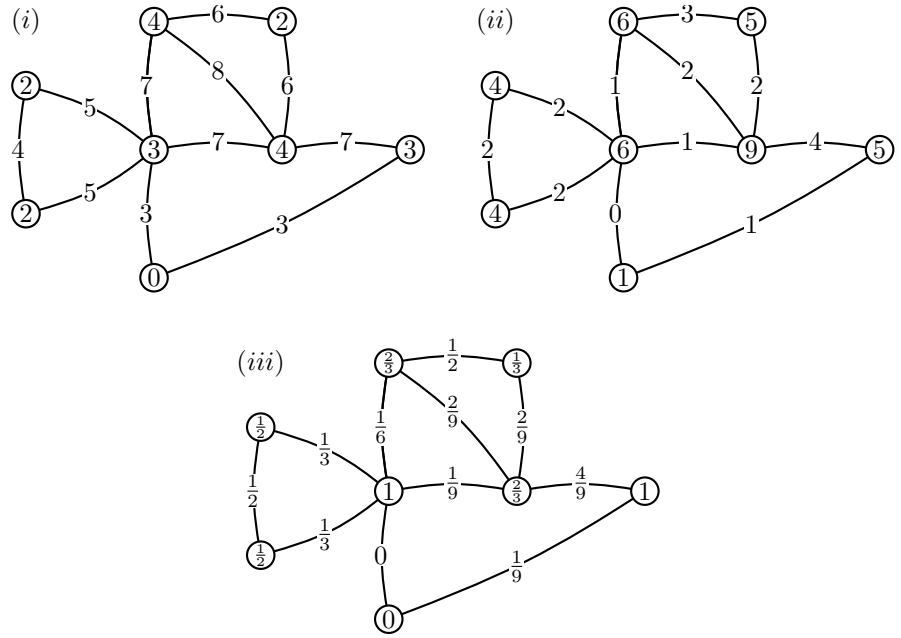

Figure 15: An example of (i) a fractional vertex cover (primal problem), and (ii) of an infeasible edge packing. In (i) each node is labelled with the covering weight $x(v)$ of node v, and each edge $\{u, v\}$ is labelled with $x(u) + x(v)$. In (ii) each edge e is labelled with the (infeasible) packing weight $y(e)$, and each node v is labelled with $y[v]$. The total weights of x and y are equal: $x(V) = y(E)$. Note that in (i), each edge is covered at least three times, giving $f = 3$, and in (ii) each node has $y[v]$ at most 9. This gives $\alpha = 3$. In (iii) nodes are labelled with the normalised x, that is computed as $x(v)/(\min_{e:v\in e} \sum_{u\in e} x(u))$, and edges are labelled with $y(e)/(\max_{v\in e} y[v])$.

if $\Delta^{1/k_p} \geq 2^{1/k_d}$, and

$$\Delta^{4/(\beta_1 \log \Delta)} 2^{1/(\beta_2 \log 2)} = e^{4/\beta_1} e^{1/\beta_2},$$

if $2^{1/k_d} > \Delta^{1/k_p}$.

Again, from Theorem 5.3 we get that the total running time of the algorithm is

$$\begin{aligned}
&O\left(k_p k_d \left(1 + \frac{1}{\Delta^{1/k_p} - 1}\right)\left(1 + \frac{k_p}{\Delta^{1/k_p} \log \Delta}\right)\right) \\
&= O\left(\beta_1 \log \Delta \beta_2 \log 2 \left(1 + \frac{1}{2^{1/\beta_1} - 1}\right)\left(1 + \frac{1}{2^{1/\beta_1}}\right)\right) \\
&= O(\log \Delta).
\end{aligned}$$

□

First we present the algorithm. It computes a solution to the primal and to the dual linear programs. The basic idea is to compute both solutions in parallel such

that both have the same objective value at all times. When the main part of the algorithm terminates, each primal variable is covered at least f times and each dual variable is only a small, αf-factor away from being feasible. We then normalize the values of both variables by dividing each primal variable by at least f and each dual variable by at most αf. Now we are able to bound the ratio of the value of the two solutions by α, which by the duality of linear programming gives an approximation guarantee. See Figure 15 for an illustration of this principle.

Let G be a simple graph. Each node $v \in V_G$ is associated with a *primal variable* $x(v)$ and each edge $e \in E_G$ is associated with a *dual variable* $y(e)$. We also call nodes *primal nodes* and edges *dual nodes*. Formally we solve two linear programs, a covering LP

$$\begin{aligned} \text{minimise} \quad & \sum_{v \in V_G} x(v) \\ \text{subject to} \quad & \sum_{v \in e} x(v) \geq 1 \text{ for each } e \in E_G \\ \text{and} \quad & x(v) \geq 0 \text{ for each } x \in V_G, \end{aligned}$$

and a packing LP

$$\begin{aligned} \text{maximise} \quad & \sum_{e \in E_G} y(e) \\ \text{subject to} \quad & \sum_{e \in E : v \in e} y(e) \leq 1 \text{ for each } v \in V_G \\ \text{and} \quad & y(e) \geq 0 \text{ for each } y \in E_G. \end{aligned}$$

We present the algorithm as if both the nodes and the edges were computers and both were running a different algorithm, one for the primal nodes and one for the dual nodes. The dual algorithm can be simulated by the endpoints of each edge, if they gather all the information the edge would gather in the algorithm.

The algorithm is different for the primal and the dual nodes. The primal nodes run Algorithm 2 and the dual nodes run Algorithm 3. Both parts consist of three nested loops. The iterations of the loops for the primal and the dual nodes are synchronised. The main idea is that inside the inner loop a primal node becomes active only if it has many dual neighbours that have not yet been covered during that iteration of the outer loop. Each edge e has a requirement, $r(e) \leq 1$, which is decreased every time the dual node is saturated, and a counter $f(e)$, which counts the number of times e has been saturated. A third variable, $w(e)$, counts the amount of leftover

weight from the previous iteration of the two inner loops and the added weight of the current iteration.

The primal nodes compute a *efficiency per cost ratio* $\gamma(v)$, defined as

$$\gamma(v) = \sum_{\substack{e \in E_G: \\ v \in e}} r(e),$$

where $r(e)$ is the requirement of the edge e. Note that during the iterations of the inner loops nodes compute a dynamic version of this ratio, $\tilde{\gamma}(v)$, defined as

$$\tilde{\gamma}(v) = \sum_{\substack{e \in E_G: \\ v \in e}} \tilde{r}(e),$$

where $\tilde{r}(e)$ is a temporary requirement used inside the inner loops.

The function increaseduals() takes care of computing the proper values for the dual variables and decreasing the requirement of each dual variable. If an edge e has been covered during an iteration of the middle loop, the edge will increase its value $y(e)$, compute a new $f(e)$, and decrease $r(e)$. If it has been covered f times, the requirement is immediately dropped to 0 and the node will no longer add weight to its packing. Otherwise the node has been either covered once, or more than once. These situations are considered separately

The algorithm knows some global quantities. We assume that $\Delta(G) = \Delta$ is known. In addition, we fix parameters k_p and k_d which affect the running time and approximation ratio of the algorithm. Essentially larger k_p and k_d give a slower algorithm with a better approximation ratio. In the original algorithm, the parameter k_d is related to the degree of the dual nodes. In our simplified case, these dual nodes are always edges and have degree 2. Finally, two global quantities, f and h, which require knowledge of Δ, are defined as

$$f = \left\lceil \frac{k_p + 1}{\Delta^{1/k_p} - 1} \right\rceil$$

and

$$h = \left\lceil 1 + \frac{k_p}{\Delta^{1/k_p} \ln \Delta} \right\rceil.$$

5.3 Analysis of the Algorithm

In this section we show the necessary lemmas to prove Theorem 5.3.

Algorithm 2 Primal node v approximation scheme.

1: $x(v) \leftarrow 0$
2: **for** $e_p \leftarrow k_p - 2$ **to** $-f - 1$ **by** -1 **do**
3: **for** 1 **to** h **do**
4: **for** $e_d \leftarrow k_d - 1$ **to** 0 **by** -1 **do**
5: $\tilde{\gamma}(v) \leftarrow \sum_{e:v\in e} \tilde{r}(e)$
6: **if** $\tilde{\gamma}(v) \geq \Delta^{e_p/k_p}$ **then**
7: $x^+(v) \leftarrow 1/\Delta^{e_d/k_d}; x(v) \leftarrow x(v) + x^+(v)$
8: **end if**
9: **send** $x^+(v), \tilde{\gamma}(v)$ to dual neighbours
10: **receive** $\tilde{r}(e)$ from dual neighbours
11: **end for**
12: **receive** $r(e)$ from dual neighbours
13: **end for**
14: **end for**
15: $x(v) \leftarrow x(v)/\min_{e:v\in e} \sum_{u\in e} x(u)$

Lemma 5.3. *Let v be a primal node and let*

$$y[v] = \sum_{\substack{e\in E_G:\\ v\in e}} y(e)$$

be the sum of the packing weights of its adjacent edges. After the algorithm has finished all the loops on line 14 of Algorithm 2 and on line 19 of Algorithm 3, $y[v]$ is bounded by

$$y[v] \leq (k_p + f + 1)\Delta^{3/k_p} \max\left\{\Delta^{1/k_p}, 2^{1/k_d}\right\}.$$

To show Lemma 5.3, we need to show some auxiliary lemmas, which bound the increase of the packing function y. First, however, we bound the efficiency per cost ratio of a node.

Lemma 5.4. *For each primal node v, at all times, the efficiency per cost ratio $\gamma(v)$ is bounded by*

$$\gamma(v) \leq \Delta^{(e_p+2)/k_p}.$$

Proof. Recall that the efficiency per cost ratio is

$$\gamma(v) = \sum_{\substack{e\in E_G:\\ v\in e}} r(e)$$

Algorithm 3 Dual node (edge) e approximation scheme

1: $y(e) \leftarrow y^+(e) \leftarrow w(e) \leftarrow f(e) \leftarrow 0; r(e) \leftarrow 1$
2: **for** $e_p \leftarrow k_p - 2$ **to** $-f - 1$ **by** -1 **do**
3: **for** 1 **to** h **do**
4: $\tilde{r}(e) \leftarrow r(e)$
5: **for** $e_d \leftarrow k_d - 1$ **to** 0 **do**
6: **receive** $x^+(v)$, $\tilde{\gamma}(v)$ from primal neighbours
7: $y^+(e) \leftarrow y^+(e) + \tilde{r}(e) \sum_{v \in e} x^+(v)/\tilde{\gamma}(v)$
8: $w^+(e) \leftarrow \sum_{v \in e} x^+(v)$
9: $w(e) \leftarrow w(e) + w^+(e)$
10: $f(e) \leftarrow f(e) + w^+(e)$
11: **if** $w(e) \geq 1$ **then**
12: $\tilde{r}(e) \leftarrow 0$
13: **end if**
14: **send** $\tilde{r}(e)$ to primal neighbours
15: **end for**
16: increaseduals()
17: **send** $r(e)$ to primal neighbours
18: **end for**
19: **end for**
20: $y(e) \leftarrow y(e)/\max_{v \in e} y[v]$

Algorithm 4 Procedure increaseduals()

```
1: if w(e) ≥ 1 then
2:     if f(e) ≥ f then
3:         y(e) ← y(e) + y⁺(e); y⁺(e) ← 0
4:         r(e) ← 0; w(e) ← 0
5:     else if w(e) ≥ 2 then
6:         y(e) ← y(e) + y⁺(e); y⁺(e) ← 0
7:         r(e) ← r(e)/Δ^{⌊w(e)⌋/k_p}
8:     else
9:         λ ← max{2^{1/k_d}, Δ^{1/k_p}}
10:        y(e) ← y(e) + min{y⁺(e), r(e)λ/Δ^{e_p/k_p}}
11:        y⁺(e) ← y⁺(e) − min{y⁺(e), r(e)λ/Δ^{e_p/k_p}}
12:        r(e) ← r(e)/Δ^{1/k_p}
13:    end if
14:    w(e) ← w(e) − ⌊w(e)⌋
15: end if
```

and that the requirement $r(e)$ is updated only during increaseduals() at the end of the middle loop. We prove the claim by induction. The claim holds at the beginning of the execution of Algorithm 2 by the definition of Δ. Now assume that the claim holds at the beginning of the iteration.

We look at what happens inside the procedure increaseduals(). It is called after the last iteration of the inner loop. Now it must hold that

$$\tilde{\gamma}(v) \leq \Delta^{e_p/k_p}$$

because otherwise v would have set $x^+(v) = 1$ during the last iteration of the inner loop. This in turn would have led to each neighbouring edge e setting $w(e) \geq 1$ and $\tilde{r}(e) = 0$, which in turn implies that $\tilde{\gamma}(v) = 0$ after the last iteration of the inner loop, when increaseduals() is called. Edges that become covered, set $\tilde{r}(e) = 0$, and otherwise do not change this value. Edges that become covered divide their requirement $r(e)$ by at least a factor Δ^{1/k_p}, so after a call of increaseduals() it holds that

$$\gamma'(v) \leq \tilde{\gamma}(v) + \frac{\gamma(v) - \tilde{\gamma}(v)}{\Delta^{1/k_p}} \leq \Delta^{e_p/k_p} + \frac{\gamma(v) - \Delta^{e_p/k_p}}{\Delta^{1/k_p}},$$

where $\gamma(v)$ and $\gamma'(v)$ denote the values before and after the execution of increaseduals(), respectively. Before e_p is decreased and the algorithm proceeds to the next iteration of the main loop, the middle loop is executed h times. By our induction hypothesis

it holds that

$$\gamma(v) \leq \Delta^{(e_p+2)/k_p}$$

at the beginning of the main loop. After h iterations of the middle loop we have that

$$\gamma(v) \leq \Delta^{e_p/k_p} + \frac{\Delta^{(e_p+2)/k_p} - \Delta^{e_p/k_p}}{\Delta^{h/k_p}}. \tag{9}$$

We have to show that the right side of (9) is at most $\Delta^{(e_p+1)/k_p}$. We have that

$$\Delta^{e_p/k_p} + \frac{\Delta^{(e_p+2)/k_p} - \Delta^{e_p/k_p}}{\Delta^{h/k}} \leq \Delta^{(e_p+1)/k_p},$$

which implies that

$$\Delta^{(e_p+2)/k_p} - \Delta^{e_p/k_p} \leq \Big(\Delta^{(e_p+1)/k_p} - \Delta^{e_p/k_p}\Big)\Delta^{h/k_p},$$

which is equivalent to

$$\Delta^{2/k_p} - 1 \leq \Big(\Delta^{1/k_p} - 1\Big)\Delta^{h/k_p},$$

which finally implies that

$$\Big(\Delta^{1/k_p} - 1\Big)\Big(\Delta^{1/k_p} + 1\Big) \leq \Big(\Delta^{1/k_p} - 1\Big)\Delta^{h/k_p}. \tag{10}$$

Taking a logarithm preserves inequalities, so we have that

$$\ln\Big(\Delta^{1/k_p} + 1\Big) \leq h \ln \Delta^{1/k_p},$$

yielding

$$h \geq \frac{\ln\Big(\Delta^{1/k_p} + 1\Big)}{\ln \Delta^{1/k_p}}.$$

Logarithm is a concave function, so it holds that $\ln(x+1) \leq \ln x + 1/x$. Now we have that (10) holds if

$$\begin{aligned}\frac{\ln\Big(\Delta^{1/k_p} + 1\Big)}{\ln \Delta^{1/k_p}} &\leq 1 + \frac{1}{\Delta^{1/k_p} \ln \Delta^{1/k_p}} \\ &= 1 + \frac{k_p}{\Delta^{1/k_p} \ln \Delta} \leq h\end{aligned}$$

by the definition of h. □

The next lemma bounds the increase of y during each iteration of the middle loop. This is required to show that the packing constraint of any node is not too far away from being feasible.

Lemma 5.5. *In Algorithm 3 when a node enters* increaseduals(), *the increase of* $y(e)$, $y^+(e)$ *is bounded by*

$$y^+(e) \leq r(e)\frac{w(e)}{\Delta^{e_p/k_p}}, \tag{11}$$

and

$$y^+(e) \leq r(e) \cdot \frac{\Delta^{1/k_d}+1}{\Delta^{e_p/k_p}}. \tag{12}$$

Proof. We start with first inequality. Outside the procedure increaseduals(), both $y^+(e)$ and $w(e)$ are increased in equal amounts on lines 7 and 9 of the dual algorithm. The term Δ^{e_p/k_p} is only decreased between iterations, which in turn only increases the right side of (11). In addition, $r(e)$ is not changed outside increaseduals(). As it is, the inequality can only be violated inside increaseduals().

During increaseduals(), we have three different cases, depending on the value of $w(e)$. If $w(e) < 1$, nothing happens and (11) is not violated. If $w(e) \geq 2$ or $f(e) \geq f$ then $y^+(e)$ is set to zero and the inequality holds trivially. Finally, we look at the case when $1 \leq w(e) < 2$. Denote by $w'(e)$ the fractional part of $w(e)$, $w'(e) = w(e) - 1$. Inside increaseduals(), $\lambda r(e)/\Delta^{e_p/k_p}$ is subtracted from $y^+(e)$ and $r(e)$ is divided by Δ^{1/k_p}. Finally, $w(e)$ is set to $w'(e)$. Assuming that (11) holds before the beginning of increaseduals(), the following inequality must hold after the execution of increaseduals() for the lemma to hold:

$$y^+(e) - \frac{r(e)\lambda}{\Delta^{e_p/k_p}} \leq r(e)\frac{w(e)}{\Delta^{e_p/k_p}} - r(e)\frac{\lambda}{\Delta^{e_p/k_p}} \leq \frac{r(e)}{\Delta^{1/k_p}}\frac{w'(e)}{\Delta^{e_p/k_p}}.$$

This gives us

$$\begin{aligned}
&\frac{r(e)}{\Delta^{1/k_p}}\frac{w'(e)}{\Delta^{e_p/k_p}} - r(e)\frac{1+w'(e)-\lambda}{\Delta^{e_p/k_p}} \\
&= \lambda\Delta^{1/k_p} - \Delta^{1/k_p} - w'(e)\Delta^{1/k_p} + w'(e) \\
&\geq \Delta^{2/k_p} - \Delta^{1/k_p} - w'(e)\Delta^{1/k_p} + w'(e) \\
&= \left(\Delta^{1/k_p} - 1\right) \cdot \left(\Delta^{1/k_p} - w'(e)\right) \geq 0
\end{aligned}$$

because $\lambda \geq \Delta^{1/k_p} \geq 1$ and $w'(e) < 1$.

Next we look at the second inequality. We use the first inequality to get that

$$y^+(e) \leq r(e)\frac{w(e)}{\Delta^{e_p/k_p}} \leq r(e)\frac{2^{1/k_d}+1}{\Delta^{e_p/k_p}},$$

that is equivalent to

$$w(e) \leq 2^{1/k_d} + 1.$$

During previous iteration of the inner loop, either $w(e)$ was set to $w(e) \geq 1$, and then $\tilde{r}(e)$ was set to 0, or it held that $w(e) < 1$ after the previous iteration of the inner loop. In the first case, during the current iteration $\tilde{r}(e) = 0$ and therefore $y^+(e) = 0$ and the inequality holds. In the second case, we know that the value of $w(e)$ after the previous iteration, denoted by $w'(e)$, was at most 1. We want to show that

$$w^+(e) \leq 2^{1/k_d}$$

or $\tilde{r}(e) = 0$ after line 8 of the dual algorithm. Assume now that $w^+(e) > 2^{1/k_d}$ and $\tilde{r}(e) > 0$. Denote by $w'^+(e)$ the increase of $w(e)$ during the previous iteration of the inner loop. If the primal values of nodes adjacent to e are increased during the current iteration, then they were also increased during the previous iteration, as $\tilde{\gamma}(v)$ can only decrease, and Δ^{e_p/k_p} is constant during the iterations of the inner loop. Given our assumption, we get that

$$w'^+(e) \geq \frac{w^+(e)}{2^{1/k_d}} > \frac{2^{1/k_d}}{2^{1/k_d}} = 1.$$

This is a contradiction with the fact that we assumed $\tilde{r}(v) > 0$ and $w^+(e) > 0$.

To complete the proof, we consider the case when the current iteration is the first. We have $e_d = k_d - 1$ and therefore

$$w^+(e) = \sum_{v:v\in e} x^+(v) \leq \sum_{v:v\in e} \frac{1}{2^{(k_d-1)/k_d}} = 2^{1/k_d}. \qquad \square$$

Next we proceed to bound the increase of the dual values related to a single primal node. Denote by $y^+[v]$ the increase of $y[v]$ during the last execution of increaseduals(), and by $\gamma^-(v)$ the corresponding decrease of $\gamma(v)$.

Lemma 5.6. *For each node v it holds after the execution of* increaseduals() *that*

$$y^+[v] \leq \frac{\Delta^{3/k_p} \max\{\Delta^{1/k_p}, 2^{1/k_d}\}}{\gamma(v)\big(\Delta^{1/k_p} - 1\big)} \cdot \gamma^-(v).$$

Proof. We show the lemma by showing another inequality, namely that for each adjacent edge e of v, the increase $y^+(e)$ of $y(e)$ is bounded by

$$y^+(e) \leq \frac{\Delta^{1/k_p} \max\{\Delta^{1/k_p}, 2^{1/k_d}\}}{\Delta^{e_p/k_p}\big(\Delta^{1/k_p} - 1\big)} \cdot r^-(e), \tag{13}$$

where $r^-(e)$ is the decrease of $r(e)$.

We know by Lemma 5.4 that $\gamma(v) \leq \Delta^{e_p+2)/k_p}$. In addition, it holds by definition that

$$y^+[v] = \sum_{e:v\in e} y^+(e)$$

and

$$\gamma^-(v) = \sum_{e:v\in e} r^-(e).$$

Now we get that

$$\begin{aligned} y^+[v] = \sum_{e:v\in e} y^+(e) &\leq \sum_{e:v\in e} \frac{\Delta^{1/k_p}\max\{\Delta^{1/k_p}, 2^{1/k_d}\}}{\Delta^{e_p/k_p}\left(\Delta^{1/k_p}-1\right)} \cdot r^-(e) \\ &= \frac{\Delta^{1/k_p}\max\{\Delta^{1/k_p}, 2^{1/k_d}\}}{\Delta^{e_p/k_p}\left(\Delta^{1/k_p}-1\right)} \sum_{e:v\in e} r^-(e) \\ &\leq \frac{\Delta^{2/k_p}\Delta^{1/k_p}\max\{\Delta^{1/k_p}, 2^{1/k_d}\}}{\gamma(v)\left(\Delta^{1/k_p}-1\right)}\gamma^-(v). \end{aligned}$$

To show (13), we consider the two possible cases inside increaseduals(). First, if $w(e) \geq 2$, we know by Lemma 5.5 that

$$y^+(e) \leq r(e)\frac{\Delta^{1/k_p}+1}{\Delta^{e_p/k_p}}.$$

The requirement $r(e)$ is divided at least by Δ^{2/k_p}, so we get that

$$r^-(e) \geq r(e) - \frac{r(e)}{\Delta^{2/k_p}} = r(e)\frac{\Delta^{2/k_p}-1}{\Delta^{2/k_p}}.$$

This gives us

$$\begin{aligned} y^+(e) &\leq \frac{\Delta^{1/k_p}+1}{\Delta^{e_p/k_p}}\frac{\Delta^{2/k_p}}{\Delta^{2/k_p}-1} \cdot r^-(e) \\ &\leq \frac{\left(1+\Delta^{1/k_p}\right)\Delta^{1/k_p}\max\{\Delta^{1/k_p}, 2^{1/k_d}\}}{\Delta^{e_p/k_p}\left(\Delta^{1/k_p}+1\right)\left(\Delta^{1/k_p}-1\right)} \cdot r^-(e) \\ &= \frac{\Delta^{1/k_p}\max\{\Delta^{1/k_p}, 2^{1/k_d}\}}{\Delta^{e_p/k_p}\left(\Delta^{1/k_p}-1\right)} \cdot r^-(e). \end{aligned}$$

Now the second case is very similar. We have by Lemma 5.5 that

$$y^+(e) \leq \frac{r(e)\cdot\max\{\Delta^{1/k_p}, 2^{1/k_d}\}}{\Delta^{e_p/k_p}}$$

and by definition of the algorithm that

$$r(e) \leq \frac{\Delta^{1/k_p}}{\Delta^{1/k_p}-1}r^-(e).$$

This gives us again (13)

$$y^+(e) \leq \frac{\max\{\Delta^{1/k_p}, 2^{1/k_d}\}}{\Delta^{e_p/k_p}} \frac{\Delta^{1/k_p}}{\Delta^{1/k_p}-1} r^-(e). \qquad \square$$

Next we bound the factor by which the duals weights have to be divided to make the dual constraints feasible.

Lemma 5.7. *After the main part of the algorithm, that is after line 14 of Algorithms 2 and 3, it holds for any node v that*

$$y[v] \leq (k_p + f + 1)\Delta^{3/k_p} \max\{\Delta^{1/k_p}, 2^{1/k_d}\}. \tag{14}$$

Proof. We define

$$Q = \Delta^{3/k_p} \max\{\Delta^{1/k_p}, 2^{1/k_d}\}.$$

Before $\gamma(v)$ is decreased for the last time, there must be at least one adjacent edge e of v that has $r(e) > 0$. As each $r(e)$ is set to 0 after e has been covered f times, it must hold that $r(e) \geq 1/\Delta^{(f-1)/k_p}$ and therefore also $\gamma(v) \geq 1/\Delta^{(f-1)/k_p}$. Assume that after the last decrease $\gamma(v)$ is decreased only to

$$\gamma(v) = \frac{1}{\Delta^{(f+1)/k_p}}.$$

Lemma 5.6 holds and the analysis is the same as in the case $w(e) \geq 2$. Now we can use Lemma 5.6 to bound the total weight $y[v]$ after the execution of the algorithm. By the lemma, decreasing $\gamma(v)$ by $\gamma^-(v)$ increases $y[v]$ by

$$y^+[v] \leq \frac{Q}{\Delta^{1/k_p}} \frac{\gamma^-(v)}{\gamma(v)}.$$

This has a geometric interpretation as the area of a rectangle. The total sum $y[v]$ can then be bounded by the sum of the areas of the rectangles as an integral

$$\begin{aligned} y[v] &\leq \frac{Q}{\Delta^{1/k_p}} \int_{\Delta^{-(f+1)/k_p}}^{\Delta} \frac{1}{x}\,\mathrm{d}x \\ &= \frac{Q}{\Delta^{1/k_p}} \ln \Delta^{\frac{k_p+f+1}{k_p}} \\ &= \frac{(k_p+f+1)Q \ln \Delta^{1/k_p}}{\Delta^{1/k_p}-1} \\ &\leq (k_p+f+1)Q. \end{aligned} \tag{15}$$

Last inequality follows from the fact that $\ln(1+x) \leq x$. $\square$

We have bounded the final gap between the feasible primal and dual solutions. We still have to show that the original, infeasible solutions computed by the algorithm have the same value.

Lemma 5.8. *After the main part of the algorithm, on line 14 of Algorithms 2 and 3, for each edge e it holds that $f(e) \geq f$. In addition it holds that*

$$\sum_{v \in V_G} x(v) = \sum_{e \in E_G} y(e).$$

Proof. When a node v enters the main outer loop for the last time, by Lemma 5.4 it must hold that

$$\Delta^{(-f+1)/k_p} \geq \gamma(v) = \sum_{e:v \in e} r(e).$$

Now if $\gamma(v)$ is greater than 0, then there must be exactly one adjacent edge e of v with $r(e) > 0$, as non-zero requirement has to be larger than or equal to $\Delta^{(-f+1)/k_p}$. During the last iteration of the inner loop, when $e_d = 0$, if $r(e)$ is still non-zero, $x(v)$ is increased by 1, and e becomes covered f times, forcing $r(e) = 0$.

The term $f(e)$ counts how many times e has been covered. Every time $w(e)$ is increased on line 9 of Algorithm 3, $f(e)$ is increased as well. In addition, every time that the integer part of $w(e)$ is increased, $r(e)$ is divided by $\Delta^{\lfloor w(e) \rfloor / k_p}$. The requirement $r(e)$ becomes zero only when $f(e) \geq f$, which happens exactly when e has been covered at least f times.

To show that

$$\sum_{v \in V_G} x(v) = \sum_{e \in E_G} y(e),$$

we simply note that every time $x(v)$ is increased, the weight is also divided among its adjacent edges so that $x^+(v) = y^+[v]$. Finally, because for each e, $f(e) \geq f$ at the end of the algorithm, each $y^+(e)$ from the last increase is added to $y(e)$ and $y^+(e)$ is set to zero. □

6 Lower Bound for Maximal Edge Packing

In this section we will show the first linear-in-Δ lower bound for the maximal edge packing problem. The proof will use techniques that are similar to those used by Hirvonen and Suomela [HS12]: we will study edge coloured graphs, a model that is stronger than our usual port numbering model. In Section 6.8 we will extend the

lower bound to the PO-model. This is a non-trivial extension, as the orientation on edges gives the PO-model symmetry breaking power that is not available in the edge colouring model.

The following theorem gives our result in its most genereral form.

Theorem 6.1. *Finding a maximal edge packing in a k-edge coloured graph requires $k-1$ communication rounds.*

Now if we consider the case of bounded-degree graphs, we get the following corollary.

Corollary 6.1. *Finding a maximal edge packing in anonymous edge coloured graphs with maximum degree Δ requires $\Omega(\Delta)$ communication rounds.*

This lower bound is matched by the upper bound from Section 4.3.

Our lower bound construction uses graphs where nodes have degree $d = k-1$ or $d = k$. Note that in [HS12] the lower bound constructions are d-regular graphs for $d = k-1$. In the case of maximal edge packings, regular graphs can no longer function as a lower bound construction as there is a trivial solution to the problem in a regular graph.

To model graphs that are highly symmetrical we will use edge coloured multigraphs. These are simply a tool in the proof and we are still interested in running our algorithms in simple graphs. In a simple graph, a k-edge colouring can be seen as a function that maps each edge to a colour in $[k]$, denoted by $c\colon E \to [k]$, such that no two edges that share an endpoint have the same colour. In the case of multigraphs, we have to use stronger notation to express edge colourings. This notation is given Section 6.1. In Section 6.2 we introduce *templates*, which are a restricted class of edge-coloured multigraphs. In Sections 6.3 and 6.4 we define how multigraphs and simple graphs are related and what it means to run a distributed algorithm on an edge-coloured multigraph. In Section 6.5 we formalise the situation that is difficult for any algorithm: two graphs that are almost isomorphic, at least locally, but have the property that the algorithm produces different outputs on these graphs. Finally, in Sections 6.6 and 6.7 we use these tools to inductively show Theorem 6.1.

6.1 Multigraphs

A *k-edge coloured multigraph* is a tuple

$$M = (V, E_1, E_2, \ldots, E_k),$$

where V is a set of nodes and each E_i is a symmetric relation $E_i \subseteq V \times V$. We call E_i the set of edges of colour i. As each E_i is symmetric, we use the unordered notation $\{u, v\} \in E_i$ for an edge. The edges are properly coloured, that is for each node v and each colour i, there is at most one node u such that $\{v, u\} \in E_i$. We use the shorthands $V(M)$, and $E_i(M)$ for the nodes and edges of colour i of multigraph M.

We divide the sets of edges into *proper edges* and *loops*. An edge $\{u, v\}$ is proper, if $u \neq v$. Otherwise $u = v$ and we call it a loop. Denote a loop $\{u, u\}$ by $\{u\}$. We denote the sets of proper and loop edges of colour i by

$$E_i^P(M) = \{\{u, v\} \in E_i(M) : u \neq v\}$$

and

$$E_i^L(M) = \{\{u, v\} \in E_i(M) : u = v\},$$

respectively. In general we will omit M if it is clear from the context.

Conversely, let the set of colours associated with a node v be denoted by

$$C_M(v) = \{i \in [k] : \{v, u\} \in E_i\},$$

the set of *proper colours* by

$$C_M^P(v) = \{i \in [k] : \{v, u\} \in E_i, \text{ and } v \neq u\}$$

and the set of *loop colours* by

$$C_M^L(v) = \{i \in [k] : \{v\} \in E_i^L\}.$$

Basic graph theoretic concepts are naturally extended to the case of multigraphs. The degree of a node v in a multigraph M is

$$\deg_M(v) = \sum_i |\{e \in E_i(M) : v \in e\}|.$$

The *proper degree* of a node v on a multigraph M is

$$\deg_M^P(v) = \sum_i |\{e \in E_i^P(M) : v \in e\}|.$$

A walk in a k-edge coloured multigraph M is a defined by a sequence of nodes $(v_1, v_2, v_3, \ldots, v_p)$ and a sequence of colours $(c_1, c_2, \ldots, c_{p-1})$ such that for each $i \in [p-1]$, if $c_i = j$ then $\{v_i, v_{i+1}\} \in E_j(M)$. A walk is *non-backtracking*, if for each

$i \in [p-1]$, it holds that $c_i \neq c_{i+1}$. A walk can therefore take loops but it cannot take the same edge consecutively.

Graph properties such as connectivity and distance are defined naturally as in simple graphs. The local r-neighbourhood of a node v in a k-edge coloured multigraph M is a subgraph

$$M[v,r] = (V[v,r], E_1[v,r], E_2[v,r], \ldots, E_k[v,r]),$$

where $V[v,r]$ is the set of nodes within distance r of v and each $E_i[v,r]$ is the set of edges of colour i within distance r of v. An edge is within distance r of v if at least one of its endpoints is within distance $r-1$ of v. See Figure 16 for illustration.

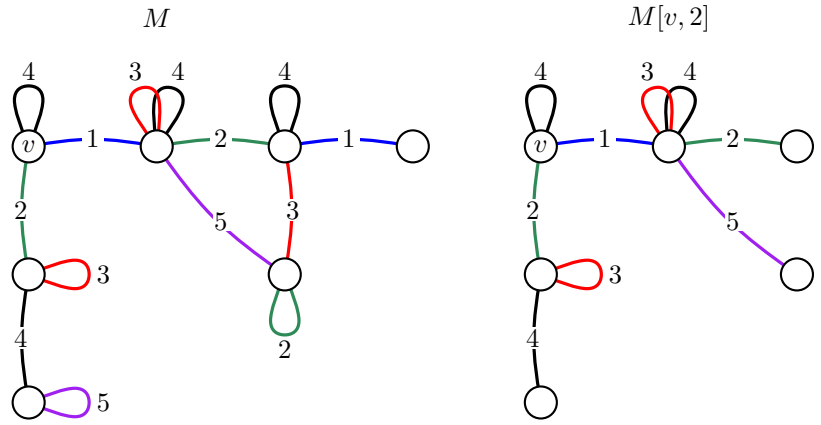

Figure 16: A 5-edge coloured multigraph M and the local 2-neighbourhood $M[v,2]$ of node v in M.

The notion of graph isomorphisms is also naturally extended to multigraphs. We say that two edge-coloured multigraphs M and M' are isomorphic, if there is a bijection $\phi\colon V(M) \to V(M')$ that preserves adjacencies, degrees and edge colours. Formally a bijection $\phi\colon V(M) \to V(M')$ is a multigraph isomorphism if and only if

(i) $\deg_M(v) = \deg_{M'}(\phi(v))$ for each v in $V(M)$ and
(ii) $\{u,v\} \in E_i(M)$ if and only if $\{\phi(u), \phi(v)\} \in E_i(M')$.

We can generalise the notion of covering graphs to edge-coloured multigraphs. Let M and M' be two edge-coloured multigraphs. We say that M is the covering graph of M', if there is a covering map $\phi\colon V(M) \to V(M')$ such that if $\{u,v\} \in E_c(M)$, then $\{\phi(u), \phi(v)\} \in E_c(M')$. Covering graphs are important in our proof. We will also define the output of an algorithm on a multigraph such that if $\phi(u) = v$, for $u \in V(M)$ and $v \in M'$, then the outputs of u and v must be the same.

6.2 Templates

In this section we introduce *templates*, which are a restricted class of multigraphs.

An *almost* (h, j)*-regular template* T is a k-edge coloured, connected multigraph, such that each node has proper degree h and degree j or $j-1$. A (h, j)*-regular template* is an almost (h, j)-regular template such that each node has proper degree h and degree j.

Each node v that has degree $k-1$ has a missing colour $F(v) = c$ such that there is no edge $\{v, u\} \in E_c$ for any $u \in V$. If a node has degree k, then the node is *colourless* and $F(v) = \perp$.

In this work the subgraph $T' \subseteq T$ induced by the proper edges of the template T is always a tree. This will lead to the fact that all of our constructions are infinite trees.

6.3 Unfoldings

Let $T = (V, E_1, E_2, \ldots, E_k)$ be an almost (h, k)-regular template. A *p-unfolding function* f of T maps each node v to a p-subset $f(v)$ of $[k]$ such that for each $i \in f(v)$, $\{v\} \in E_i$. Intuitively, f picks p incident loops for each node.

Next we define X, the *f-unfolding* of an almost (h, k)-regular template T for a p-unfolding function f. First pick an arbitrary node v_0 in $V(T)$. This does not affect the isomorphism type of the unfolding X, but does affect the labelling of $V(X)$ that we construct as we unfold T. Let $V(X)$ equal the set of all non-backtracking walks in T, starting at v_0 that use proper edges of T or loops picked by f. Formally each $x \in V(X)$ is a non-backtracking walk

$$w = (v_1 c_1 v_2 c_2 v_3 \cdots c_{\ell-1} v_\ell),$$

where $v_i \in V(T)$, and $c_i \in C_M^P(v_i) \cup f(v_i)$ for each i. We call this W, the set of f-unfolding walks in T.

Next we define a map $\phi_f \colon W \to V(T)$ as follows. For each walk w that ends in node $v_\ell \in V(T)$, we set $\phi_f(w) = v_\ell$.

Now let w and w', defined as

$$w' = (v'_1 c'_1 v'_2 c'_2 v'_3 \cdots c'_\ell v'_{\ell+1}),$$

be unfolding walks in T. The walk w' is a *successor* of w, if for each $i \in [\ell - 1]$ it holds that $v_i = v'_i$ and $c_i = c'_i$, and $v_\ell = v'_\ell$.

Now we are ready to construct the almost $(h+p,k)$-regular template X. Set $V(X) = W$. Define a missing colour function F' for each $x \in V(X)$ as $F'(x) = F(\phi_f(x))$. Now we can define $E_i(X)$ for each i. First, there is a proper edge $\{u,v\} \in E_i^P(X)$ if and only if u or v is the successor of the other and the successor's last edge colour is i. Second, there is a loop $\{v\} \in E_i^L$ if and only if $i \in C_T^L(\phi_f(v)) \setminus f(\phi_f(v))$.

Note that the function ϕ_f is a covering map. If v and u are nodes in $V(X)$ connected by an edge of colour c, and v is the successor of u, then there are two options: either $\phi_f(v) = \phi_f(u)$, or $\phi_f(v) \neq \phi_f(u)$. In the first case, the last edge of v is a loop edge at $\phi_f(v)$ and $\{\phi_f(v), \phi_f(u)\} \in E_c(T)$. In the second case, the last edge of v is a proper edge of colour c in T, and therefore $\{\phi_f(u), \phi_f(v)\} \in E_c(T)$. See Figure 17 for an illustration of an f-unfolding and the related covering map ϕ_f.

For ease of notation, if X is the f-unfolding of T at t, then denote by t the node in X that corresponds to the empty walk in T. In addition, if w is a non-backtracking walk to $u \in V(T)$ using only proper edges of T, we say that $w = u \in V(X)$.

Let f and g be two unfolding functions for template T such that for each $v \in V(T)$, it holds that $f(v) \cap g(v) = \emptyset$. Now define the $(g \circ f)$-unfolding of T at v as follows. First, take the f-unfolding of T at v, and denote this unfolding by X. Denote by ϕ the covering map from X to T related to the unfolding. Now take the g-unfolding of X at v, where we extend g to X as

$$g(u) = g(\phi(u))$$

for any $u \in V(X)$.

Lemma 6.1. *Let S and T be templates such that there are isomorphic radius-h neighbourhoods $S[s,h]$ and $T[t,h]$ and there is an isomorphism $\phi\colon V(S[s,h]) \to V(T[t,h])$ such that $\phi(s) = t$. Now if f is an unfolding function of S and f' is an unfolding function of T such that $f(v) = f'(\phi(v))$ for each $v \in V(S[s,h])$, and X is the f-unfolding of S at s and Z is the f'-unfolding of T at t, then*

$$X[s,h] \simeq Z[t,h].$$

Proof. Each node in X and in Z corresponds to a walk in S and in T, respectively. A node $v \in V(X)$ with $\mathrm{dist}_X(v,s) = d$ corresponds to a walk of length exactly d. To see this, recall that the proper edges of our templates always induce a tree. Now each node within distance h of s in X and of t in Z corresponds to a walk of length at most h in S or T.

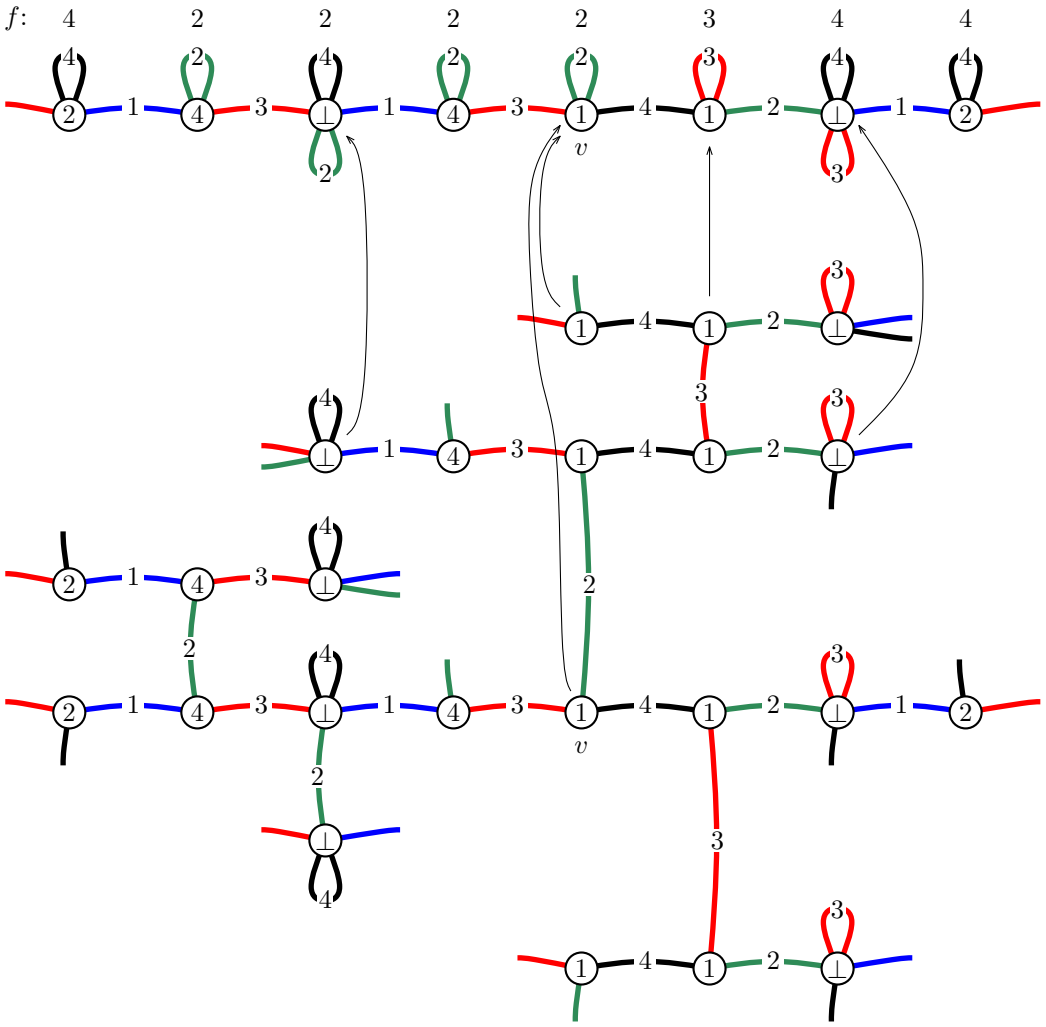

Figure 17: An almost $(2,4)$-regular template and its f-unfolding at node v. Nodes are labelled with missing colour function F. The arrows give examples of the covering map ϕ_f. The function f is a 1-unfolding function, and $k = 4$.

We can construct an isomorphism $\xi\colon V(X[s,h]) \to V(Z[t,h])$ as follows. First, set $\xi(s) = \xi(t)$. Then, for each unique, non-backtracking walk w in S, denoted by

$$w = (x_0, c_0, \ldots, c_{d-1}, x_d),$$

there exists, by the existence of ϕ, a corresponding walk w' in T, denoted by

$$w' = (\phi(x_0), c_0, \ldots, c_{d-1}, \phi(x_d)).$$

For each such pair of walks, we define $\xi(w) = w'$. We have constructed an isomorphism ξ from $X[s,h]$ to $Z[t,h]$, which concludes our proof. □

Lemma 6.2. *If f, f' and g are three unfolding functions for template T such that for each $v \in V(T)$ it holds that*

$$f(v) \cap f'(v) = \emptyset,$$

and

$$f(v) \cup f'(v) = g(v),$$

then the g-unfolding of T at v is isomorphic to the $(f \circ f')$-unfolding of T at v.

Proof. Pick an arbitrary node $u \in V(T)$. Let X be the f'-unfolding of T at u. Let ϕ be the covering map from $V(X)$ to $V(T)$. By definition, $V(X)$ is equal to the set of non-backtracking walks in T, starting at u. For each $v \in V(X)$, we defined $f(v) = f(\phi(v))$.

We will show that there is a bijective mapping between the g-unfolding walks in T starting at u, and f-unfolding walks starting at u in X. Now first, let $W(f)$ be the set of f-unfolding walks in X, and let $w \in W(f)$. At any node v in $V(X)$, the walk can take a proper edge of X such that $\phi(v)$ has an adjacent proper edge of the same colour, a proper edge of X of colour c at $\phi(v)$ such that $c \in f'(\phi(v))$, or a loop edge of colour χ at v such that $\chi \in f(v) = f(\phi(v))$. As by definition $g(v) = f(v) \cup f'(v)$ for each $v \in V(T)$, there exists a corresponding g-unfolding walk in T, constructed by following the sequence of edges with corresponding colours to w. This walk is unique as there is at most one edge of any colour at any node.

Now let $W(g)$ be the set of g-unfolding walks in T starting at u, and let $w' \in W(g)$. Again, at any node v in $V(T)$ along the walk w', the walk can take a proper edge of T or a loop edge of colour c at v such that $c \in g(v)$. Consider two nodes, v in T and v' in X such that $\phi(v') = v$. By the fact that ϕ is a covering map, we have that if there is an edge of colour c in $C_T^P(v) \cup g(v)$, then there is an edge of colour c

in $C_X^P(v') \cup f(\phi(v'))$. Now if the walk w' takes the edge of colour c and ends up at node ν in T, then by the properties of ϕ the edge of colour c at v' goes to node ν' such that $\phi(\nu') = \nu$. Therefore, for any g-unfolding walk there is a corresponding f-unfolding walk in X. This gives us that $W(g) = W(f)$. Finally, observe that by definition in the unfolding a node (an unfolding walk) is connected to each of its successors. Now holds for g-unfolding of T at u, denoted by Y, and X that

$$Y \simeq X. \qquad \square$$

Finally we observe that it can be shown that the isomorphism type of the graph is not affected by the choice of the node at which we unfold. Our proof does not require this knowledge.

6.4 Realisations

Thus far we have discussed only templates and have not specified how these relate to simple graphs, which we are interested in. Informally, a realisation $\mathsf{real}(T, v)$ of template T at v is a complete unfolding of T, that is an f-unfolding for an f defined as

$$f(v) = \{C^L(v)\}$$

for each $v \in V(T)$.

A realisation of a template is a simple, k-edge coloured graph. As it is an unfolding, we have a covering map $\phi\colon V(\mathsf{real}(T, v)) \to V(T)$.

In our proof we will use the notion of running an algorithm on a template that is not a simple graph. In this case, it is assumed that the output $\mathsf{A}(T, u, c)$ of a node $u \in V(T)$ for an edge of colour c is defined as taking the realisation R of T and then for a node $v \in V(R)$, such that $\phi(v) = u$, setting

$$\mathsf{A}(T, u, c) = \mathsf{A}(R, v, c).$$

Next we will note some useful properties of realisations. Recall that a perfect edge packing p is defined as an edge packing such that each node $v \in V(G)$ has $p[v] = 1$.

Lemma 6.3. *Let T be an almost $(h, k-1)$-regular template for any $h < k-1$. Any algorithm for maximal edge packing must produce a perfect edge packing on T.*

Proof. Let $v \in V(T)$ be an arbitrary node. Because $h < k - 1$, in the realisation of T, v has a neighbour v', such that

$$\mathsf{real}(T, v)[v, d] \simeq \mathsf{real}(T, v)[v', d]$$

for an arbitrary d. If it holds that

$$\sum_{c \in [k]} \mathsf{A}(T, v, c) < 1,$$

then also

$$\sum_{c \in [k]} \mathsf{A}(\mathsf{real}(T, v), v', c) < 1,$$

and the edge packing produced by the algorithm is not maximal. Therefore the algorithm must produce a perfect edge packing. □

6.5 Compatible and Critical Pairs

Let T and S be two multigraphs. We say that T and S are h-compatible if there are nodes $u \in V(T)$ and $v \in V(S)$ such that the h-neighbourhoods of u and v are isomorphic and there is an isomorphism ϕ that maps u to v. Figure 18 illustrates a 2-compatible pair.

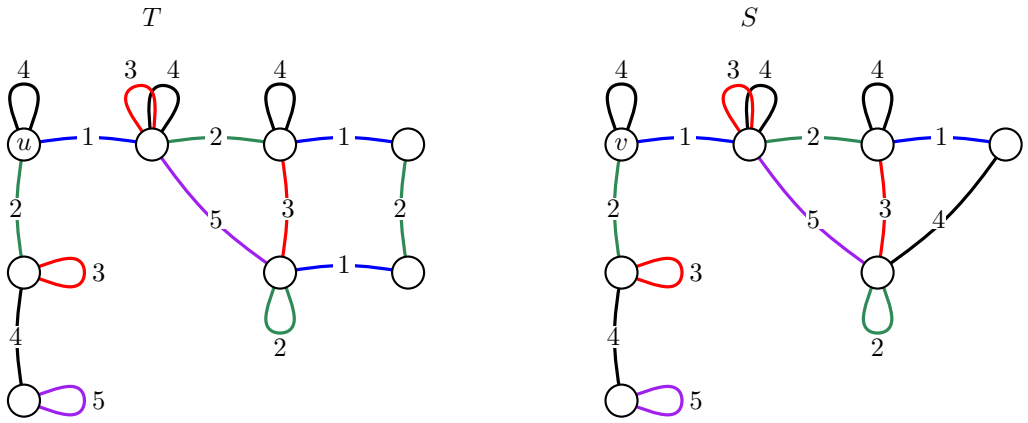

Figure 18: A 2-compatible pair T and S.

Now let A be a distributed algorithm for the maximal edge packing problem. Multigraphs T and S form an *h-critical pair* if

(i) T and S are h-compatible,

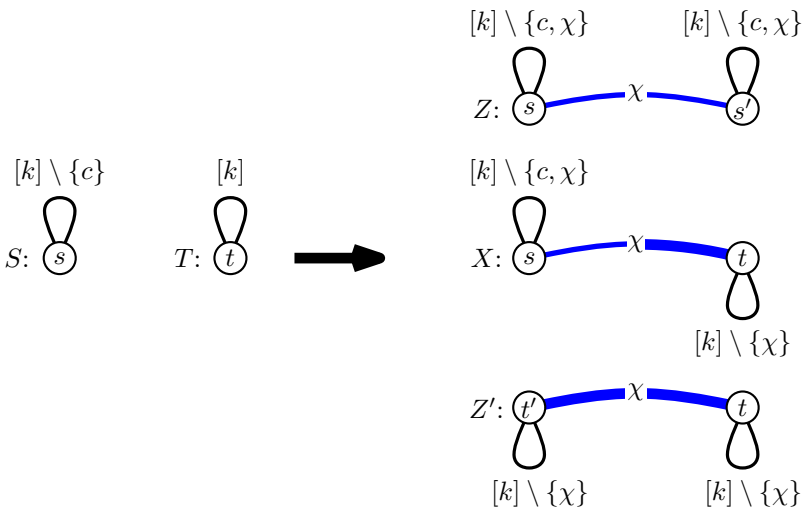

Figure 19: The construction of a 0-critical pair. The set above and below each loop corresponds to the set of loop colours at each node. Node s has missing colour c and node t has no missing colour. Because $\mathsf{A}(T, t, c) \neq 0$, there is a colour χ such that $\mathsf{A}(S, s, \chi) \neq \mathsf{A}(T, t, \chi)$. The 0-neighbourhoods of nodes s and t in X are isomorphic.

(ii) $u \in V(T)$ and $v \in V(S)$ are nodes with isomorphic h-neighbourhoods such that isomorphism $\phi\colon V(T)[v, h] \to V(S)[u, h]$ maps $\phi(u) = v$, and

(iii) there is a colour i such that the output of A in respect with colour i is different at v and at u.

Now we are ready to proceed with the proof of Theorem 6.1. We will prove the theorem by induction on the proper degree of our templates.

6.6 Base Case

First we will show that for any algorithm A, there is a 0-critical pair and thus the output of A must depend on the 1-radius neighbourhood of each node.

Lemma 6.4. *For any algorithm A for the maximal edge packing problem, there is a 0-critical pair.*

Proof. Let T be a $(0, k)$-regular template, such that $V(T) = \{t\}$ and the missing colour of t is $F_T(t) = \perp$. Now there must be a colour c such that $\mathsf{A}\big(T, t, c\big) \neq 0$. Now let S be another, $(0, k-1)$-regular template, such that $V(S) = \{s\}$ and the missing colour of s is $F_S(s) = c$. Now there must be another colour χ such that $\mathsf{A}\big(T, t, \chi\big) \neq \mathsf{A}\big(S, s, \chi\big)$. To see this, simply observe that A must produce a maximal

edge packing at t and s, and that

$$\sum_{i \in [k] \setminus \{c\}} \mathsf{A}\big(T, t, i\big) < 1$$

and

$$\sum_{i \in [k] \setminus \{c\}} \mathsf{A}\big(S, s, i\big) = 1.$$

Now construct a new, almost $(1, k)$-regular template X as follows. Let $V(X) = \{t, s\}$. Let $\{s, t\} \in E_\chi(Z)$, and let

$$C_X(t) = [k-1] \setminus \{\chi\}$$

and let

$$C_X(s) = [k-1] \setminus \{c, \chi\}.$$

Finally, set $F_X(s) = c$ and $F_X(t) = \perp$. See Figure 19 for an illustration.

Now by construction it holds that $T[t, 0] \simeq X[t, 0]$ and $S[s, 0] \simeq X[s, 0]$. As $\mathsf{A}(T, t, \chi) \neq \mathsf{A}(S, s, \chi)$, this implies that it must hold that either

$$\mathsf{A}\big(T, t, \chi\big) \neq \mathsf{A}\big(X, t, \chi\big)$$

or

$$\mathsf{A}\big(S, s, \chi\big) \neq \mathsf{A}\big(X, s, \chi\big).$$

Now assume that the output of A is different for node s in S and X. We construct another, $(1, k-1)$-regular template Z by unfolding S along edge of colour χ. Formally let $V(Z) = \{s, s'\}$, let $E_\chi(Z) = \big\{\{s, s'\}\big\}$ and let $C_Z(s) = C_Z(s') = [k] \setminus \{c, \chi\}$. Now by construction, $\mathsf{A}(S, s, \chi) \neq \mathsf{A}(Z, s, \chi)$. As the edge packing constructed by A is assumed to be maximal, there must be a loop colour $c' \in C_Z^L(s), C_X^L(s)$ such that

$$\mathsf{A}(X, s, c') \neq \mathsf{A}(Z, s, c').$$

We have that X and Z form a 0-critical pair.

The proof is similar if t changes output, but we are in that case left with a template that has a higher minimum degree. As we will see, this would only help us with our lower bound construction. □

Now given that we have a 0-critical pair for algorithm A, the algorithm must look at least one hop in the communication network to produce a feasible output.

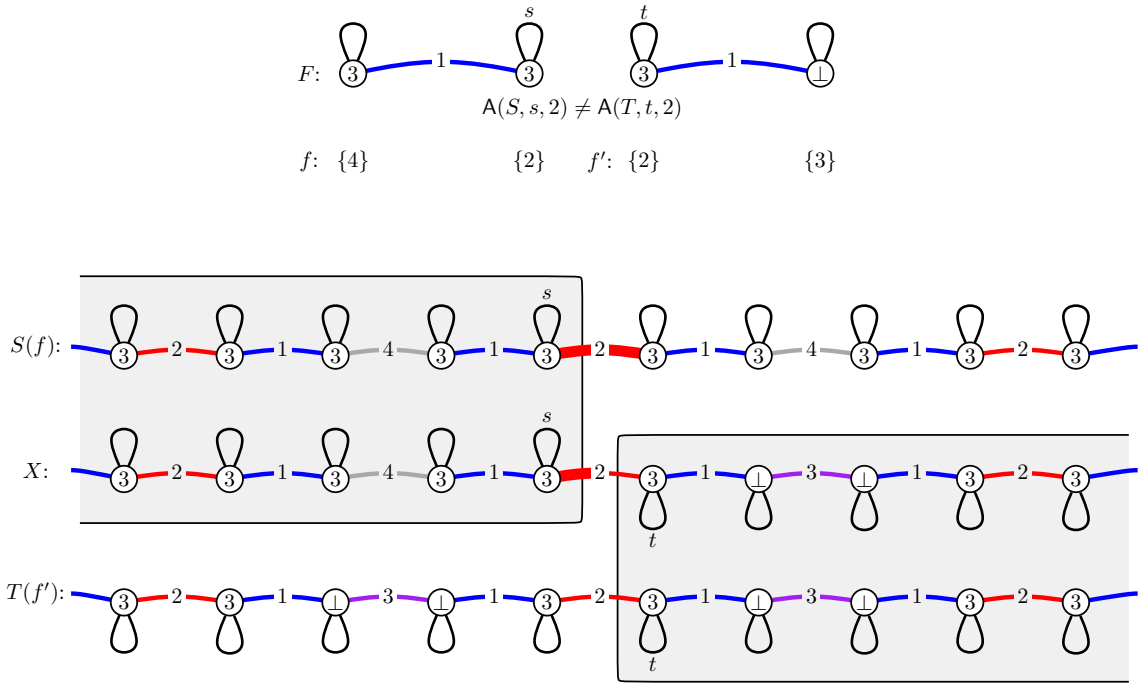

Figure 20: The inductive step for $h = 2$. Almost $(1, k)$-regular templates S and T form a 0-critical pair. The outputs of s and t are different for the edge of colour 2. By unfolding along the edge of colour 2 at s and t we construct templates X, $S(f)$, and $T(f')$ such that X, and $S(f)$ and $T(f')$, respectively, are pairwise 1-compatible. At least one of the pairs will form a 1-critical pair.

6.7 Inductive Step

Next we will show that given an h-critical pair, for any positive $h < k - 2$, we can construct an $(h + 1)$-critical pair.

Lemma 6.5. *Assume that for algorithm* A *there is a* h*-critical pair* T*,* S*, where* T *and* S *are almost* $(h + 1, k)$ *regular templates and* $h < k - 2$*. Then there is a* $(h + 1)$*-critical pair.*

Proof. As in the statement of the lemma, assume that there is a h-critical pair T, S. Now by definition, there are nodes $t \in V(T)$ and $s \in V(S)$ such that

$$T[t, h] \simeq S[s, h],$$

there is a colour c such that there is a loop $\{t\} \in E_\chi(T)$ and loop $\{s\} \in E_\chi(S)$, and

$$\mathsf{A}(T, t, \chi) \neq \mathsf{A}(S, s, \chi).$$

The fact that $h \leq k-3$ implies that for each node $v \in V(T), V(S)$, it holds that $C_T^L(v), C_S^L(v) \neq \emptyset$. Now define 1-unfolding functions $f\colon V(T) \to [k]$ and $f'\colon V(S) \to [k]$ as follows. First, let ϕ be a graph isomorphism from $T[t,h]$ to $S[s,h]$ that maps $\phi(t) = s$. Let $f(t) = f'(s) = \chi$. For each $v \in V(T)[t,h] \setminus \{t\}$ pick an arbitrary $c \in C_T^L(v)$ and set $f(v) = f'(\phi(v)) = c$. For each $v \notin V(T)[t,h]$ and each $u \notin V(S)[s,h]$ pick arbitrary colours with loop edges at v and u, respectively.

We construct a third, almost $(h+1,k)$-regular template X as follows. Informally, we take certain f and f'-unfoldings of S and T, and cut both at s and t, respectively, along the edges of colour χ. Then we take the connected components of s and t and add an edge of colour χ between s and t.

Formally, let $S(f)$ be the f-unfolding of S at s and let $T(f')$ be the f'-unfolding of T at t. Let ψ be the covering function $\psi\colon V(S(f)) \to V(S)$ and let ξ be the covering function $\xi\colon V(T(f')) \to V(T)$. Now pick two nodes, $\sigma \in V(S(f))$ and $\tau \in V(T(f'))$ such that $\psi(\sigma) = s$ and $\xi(\tau) = t$. By Lemma 6.1 it holds that

$$S(f)[\sigma, h] \simeq T(f')[\tau, h].$$

Specifically, observe that there are nodes σ' and τ' such that $\{\sigma, \sigma'\} \in E_\chi(S(f))$, $\{\tau, \tau'\} \in E_\chi(T(f'))$, and

$$S(f)[\sigma, \ell] \simeq S(f)[\sigma', \ell] \text{ and } T(f')[\tau, \ell] \simeq T(f')[\tau', \ell] \tag{16}$$

for an arbitrarily large ℓ.

For a template T, a node t, and a colour c we define the function $\mathrm{cut}(T, t, c)$, which returns a multigraph M such that we remove the edge of colour c at t and then return the connected component containing t. Now set

$$S' \leftarrow \mathrm{cut}(S(f), \sigma, \chi),$$

and

$$T' \leftarrow \mathrm{cut}(T(f'), \tau, \chi).$$

It still holds that

$$T'[\tau, h] \simeq S'[\sigma, h],$$

as we cut the edge of the same colour at both nodes.

Now construct an almost $(h+1,k)$-regular template by setting

$$V(X) = V(S') \cup V(T'),$$

$$E_\chi(X) = E_\chi(S') \cup E_\chi(T') \cup \{\sigma, \tau\}, \text{ and}$$
$$E_i(X) = E_i(S') \cup E_i(T') \text{ for } i \neq \chi.$$

By construction, it will hold that

$$X[\sigma, h] \simeq X[\tau, h],$$

by (16).

Recall that by construction,

$$\mathsf{A}(T', \tau, \chi) \neq \mathsf{A}(S', \sigma, \chi).$$

We will argue, that there is a node u and a loop edge of colour c at u such that $u \in V(U) \cap V(X)$, where $U \in \{S(f), T(f')\}$, and

$$\mathsf{A}(U, u, c) \neq \mathsf{A}(X, u, c).$$

First, let D_i be the set of nodes such that for each $v \in D_i$ it holds that $v \in V(S') \cap V(X)$ and $\mathrm{dist}_X(v, \sigma) = i$, or $v \in V(T') \cap V(X)$ and $\mathrm{dist}_X(v, \tau) = i$. Now for any algorithm with a running time at most r, the outputs of nodes in D_{r+1} are the same in X, and, respectively, $S(f)$ and $T(f')$. Note that the proper edges in X induce a tree and as such X has a unique path from any node in D_{r+1} to σ and τ. Now consider the outputs of the nodes in D_{r+1} to be fixed. We will use induction on i. For any set D_i, assume that the outputs of the set D_{i+1} have already been fixed. Each node $v \in D_i$ has exactly one proper incident edge that goes to a node in D_{i-1}, and rest of the proper incident edges go to nodes in D_{i+1}. The outputs of these edges have been already fixed. There are two options for each node: either change the output of a loop edge, or keep the outputs of all edges fixed. If any of the outputs for the loop edges change, our proof is complete. Therefore assume that the outputs for the loop edges remain the same. Now the output of the remaining edge to D_{i-1} must be fixed as well, because by Lemma 6.3 the algorithm must produce a perfect edge packing.

By this argument, the outputs of all nodes are fixed until we reach $D_0 = \{\sigma, \tau\}$. Recall that $\mathsf{A}(S(f), \sigma, \chi) \neq \mathsf{A}(T(f'), \tau, \chi)$, and the outputs of the proper edges, except for the critical edge $\{\sigma, \tau\}$, are fixed. Therefore, if no loop edge changes its output, the output of the algorithm is not an edge packing and we have reached a contradiction.

Now without loss of generality, assume that there is a node $\kappa \in V(S')$ such that κ changes its output for a loop edge of colour π. The following proof goes similarly, if

κ were in $V(T')$. Now we claim that X and the original unfolded graph $S(f)$ form an $(h+1)$-critical pair. In particular, we claim that

$$X[\sigma, h] \simeq S(f)[\sigma, h].$$

Denote by $\sigma' \in S(f)$ the neighbour of σ along the edge of colour χ. We know that it holds that

$$S(f)[\sigma, h] \simeq S(f)[\sigma', h] \simeq T(f')[\tau, h].$$

After cutting the edge of colour χ, and connecting σ and τ with an edge of colour χ, we get that

$$X[\sigma, h+1] \simeq S(f)[\sigma, h+1],$$

showing that X and $S(f)$ form an $(h+1)$-critical pair. □

This induction can continue until there is a node with proper degree equal to $k-1$. This happens when we would otherwise have a $(k-2)$-critical pair. The critical $(k-2)$-neighbourhoods of the templates are isomorphic, but the algorithm must produce different outputs. Therefore the algorithm must use $k-1$ steps to distinguish the two neighbourhoods. Together, Lemmas 6.4 and 6.5 imply Theorem 6.1.

6.8 Maximal Edge Packing Lower Bound for PO-model

In this section we will extend the lower bound from the previous section to the anonymous PO-model with edge orientations. Recall that in the PO-model, the communication graph has a port numbering, and in addition, each edge is oriented towards one of the nodes. Also, the model is anonymous, so the nodes do not have access to unique identifiers.

The reduction of our lower bound from the previous section to the regular port-numbering model is simple: observe that an edge colouring can be transformed into a port-numbering by nodes picking the ports for each edge in the order of the edge colours. Now if there was a faster algorithm for the maximal edge packing problem in the port-numbering model, we could simply transform any edge colouring into a port numbering locally and run this algorithm to get a better running time.

This does not hold with the PO-model. To see this, consider the case of a graph of two nodes connected by a single edge. In the PO-model, symmetry is trivially broken and we can find for example an independent set. In the anonymous model with k-edge colouring, it is not possible to break symmetry.

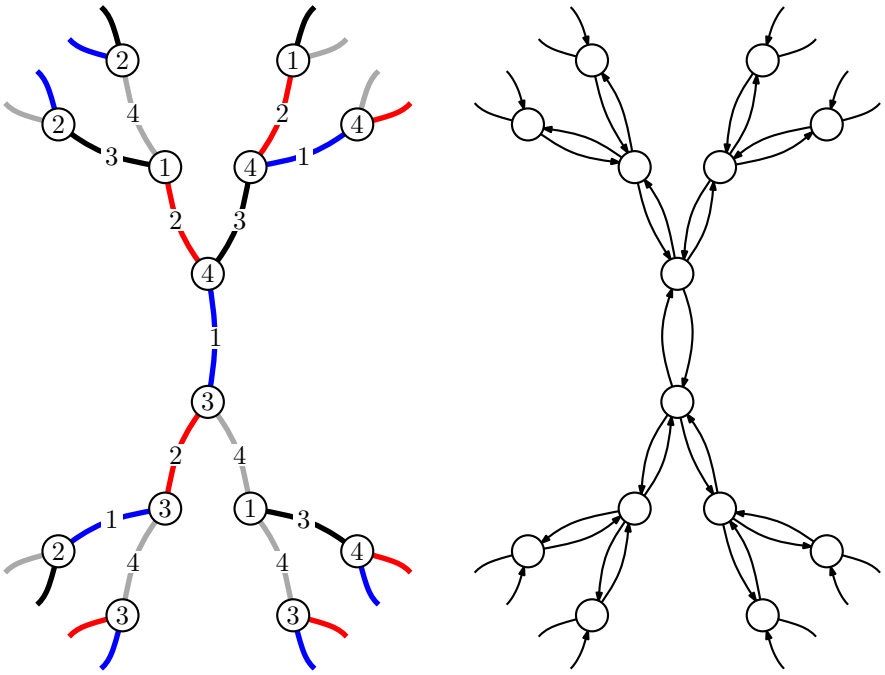

Figure 21: An edge coloured multigraph and the port-numbered construction with orientation in the proof of Corollary 6.2. The port-numbering is arbitrary (but fixed for each node), and is not shown. The maximum degree of the graph is doubled.

Now we will show the following corollary to Theorem 6.1.

Corollary 6.2. *Finding a maximal edge packing in the* PO*-model requires* $r = \left\lceil \frac{\Delta-2}{2} \right\rceil$ *communication rounds.*

Proof. Assume that there is an algorithm A that finds a maximal edge packing in the PO-model in less than r rounds. Now we can use this algorithm to break our previous lower bound as follows. Let G be a simple k-edge coloured graph. We will construct a directed multigraph M based on G. First we set $V(M) = V_G$. Then for each colour c and each edge $\{u, v\} \in E_c(G)$ add two edges to $E(M)$: (u, v) and (v, u). Each node can then assign an arbitrary port numbering to its incident edges. See Figure 21 for illustration.

Next we can unfold M to get a simple, edge-oriented graph. Let U be the unfolding of M, and let $\phi\colon V(U) \to V(M)$ be the covering map associated with this unfolding. Note that while U is possibly infinite, our algorithm has constant running time in bounded-degree graphs and therefore must announce an output even in an infinite graph. Now each node $u \in V(M)$ can simulate the execution of A at each node $v \in V(U)$ such that $\phi(v) = u$. Each node $u \in V(M)$ sends messages through each of

its edges, receives messages and updates its state. The existence of the covering map ϕ guarantees, that each node in $\phi^{-1}(u)$ takes the exactly same steps during each round.

After the simulation of A finishes, each node $u \in V(M)$ announces its output. For a node v, we define that the outputs of node v for edges (u, v) and (v, u) are $\mathsf{A}(M, v, (u, v))$ and $\mathsf{A}(M, v, (v, u))$, respectively. These outputs form a maximal edge packing in U and can be mapped back to a maximal edge packing p in G: for each edge $\{u, v\}$ we set

$$p(\{u, v\}) = \mathsf{A}(M, u, (u, v)) + \mathsf{A}(M, v, (v, u)).$$

By definition, we have

$$p[v] = \sum_{e \in E(M) : v \in e} \mathsf{A}(M, v, e) = \sum_{f \in E_G : v \in e} \mathsf{A}(G, v, e).$$

Therefore if the edge packing is maximal in M, then it must be maximal in G as well.

By construction, we have that $\Delta(M) = \Delta(U) = 2\Delta(G)$. To conclude our proof, now observe that we have computed a maximal edge packing in G in less than

$$r = \left\lceil \frac{\Delta(M) - 2}{2} \right\rceil = \Delta(G) - 1,$$

which is a contradiction. □

7 Conclusions

We have surveyed what is currently known about the distributed computation of edge packings: best known upper and lower bounds. We saw that a constant-approximation of maximum edge packing can be computed in $O(\log \Delta)$ rounds. There exists a matching lower bound. We saw that there is an algorithm for finding a weighted maximal edge packing in $O(\Delta + \log^* W)$ rounds. The requirement on W is necessary, and we showed a matching lower bound of $\Omega(\Delta)$ for the PO-model.

As we have seen, finding a maximal edge packing is directly connected with finding a good approximation of a minimum vertex cover. It is quite surprising that a local algorithm can find a 2-approximation of a minimum vertex cover, when no fast $(2 - \varepsilon)$-approximation algorithms are known even in the centralised setting.

Maximal edge packing is one of the many classical distributed graph problems, such as maximal matching, maximal independent set, and $(\Delta + 1)$-colouring, that have an algorithm that runs in time that is linear in Δ, but so far almost no corresponding lower bounds have existed. The lower bound in this work, and the similar lower bound by Hirvonen and Suomela [HS12] are only shown for anonymous models of distributed computation. It is a major open problem in the field to show linear-in-Δ lower bound for any of these problems in the ID-model. Recently Göös et al. [GHS12] that for a large class of graph optimisation problems, the PO-model and the ID-model are equivalent. It is no coincidence that we showed our lower bound also in the PO-model: combining these two results would yield the first linear-in-Δ lower bound for the ID-model for maximal edge packing.

References

ÅFP+09 Åstrand, M., Floréen, P., Polishchuk, V., Rybicki, J., Suomela, J. and Uitto, J., A local 2-approximation algorithm for the vertex cover problem. *Proc. 23rd International Symposium on Distributed Computing (DISC, Elche, Spain, September 2009)*, volume 5805 of *Lecture Notes in Computer Science*, Berlin, Germany, 2009, Springer, pages 191–205.

Ang80 Angluin, D., Local and global properties in networks of processors. *Proc. 12th Annual ACM Symposium on Theory of Computing (STOC, Los Angeles, CA, USA, April 1980)*, New York, NY, USA, 1980, ACM Press, pages 82–93.

ÅS10 Åstrand, M. and Suomela, J., Fast distributed approximation algorithms for vertex cover and set cover in anonymous networks. *Proc. 22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA, Santorini, Greece, June 2010)*, New York, NY, USA, 2010, ACM Press, pages 294–302.

ASW88 Attiya, H., Snir, M. and Warmuth, M. K., Computing on an anonymous ring. *Journal of the ACM*, 35,4(1988), pages 845–875.

BYE81 Bar-Yehuda, R. and Even, S., A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2,2(1981), pages 198–203.

CHW08 Czygrinow, A., Hańćkowiak, M. and Wawrzyniak, W., Fast distributed approximations in planar graphs. *Proc. 22nd International Symposium on Distributed Computing (DISC, Arcachon, France, September 2008)*, volume 5218 of *Lecture Notes in Computer Science*, Berlin, Germany, 2008, Springer, pages 78–92.

CV86 Cole, R. and Vishkin, U., Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70,1(1986), pages 32–53.

GHS12 Göös, M., Hirvonen, J. and Suomela, J., Lower bounds for local approximation. *Proc. 31st Annual ACM Symposium on Principles of Distributed Computing (PODC, Madeira, Portugal, July 2012)*, New York, NY, USA, 2012, ACM Press, pages 175–184.

GJ79 Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, NY, USA, 1979.

GPS88 Goldberg, A. V., Plotkin, S. A. and Shannon, G. E., Parallel symmetry-breaking in sparse graphs. *SIAM Journal on Discrete Mathematics*, 1,4(1988), pages 434–446.

HKP01 Hańćkowiak, M., Karoński, M. and Panconesi, A., On the distributed complexity of computing maximal matchings. *SIAM Journal on Discrete Mathematics*, 15,1(2001), pages 41–57.

HS12 Hirvonen, J. and Suomela, J., Distributed maximal matching: greedy is optimal. *Proc. 31st Annual ACM Symposium on Principles of Distributed Computing (PODC, Madeira, Portugal, July 2012)*, New York, NY, USA, 2012, ACM Press, pages 165–174.

KMW06 Kuhn, F., Moscibroda, T. and Wattenhofer, R., The price of being near-sighted. *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA, Miami, FL, USA, January 2006)*, New York, NY, USA, 2006, ACM Press, pages 980–989.

KMW10 Kuhn, F., Moscibroda, T. and Wattenhofer, R., Local computation: Lower and upper bounds, 2010. Manuscript, arXiv:1011.5470 [cs.DC].

Kuh05 Kuhn, F., *The Price of Locality: Exploring the Complexity of Distributed Coordination Primitives*. Ph.D. thesis, ETH Zurich, 2005.

Kuh08 Kuhn, F., Local approximation of covering and packing problems. In *Encyclopedia of Algorithms*, Kao, M.-Y., editor, Springer, New York, NY, USA, 2008.

KVY94 Khuller, S., Vishkin, U. and Young, N., A primal-dual parallel approximation technique applied to weighted set and vertex covers. *Journal of Algorithms*, 17,2(1994), pages 280–289.

Lin92 Linial, N., Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21,1(1992), pages 193–201.

Mos06 Moscibroda, T., *Locality, Scheduling, and Selfishness: Algorithmic Foundations of Highly Decentralized Networks*. Ph.D. thesis, ETH Zurich, 2006.

NS95 Naor, M. and Stockmeyer, L., What can be computed locally? *SIAM Journal on Computing*, 24,6(1995), pages 1259–1277.

Pel00 Peleg, D., *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

PR01 Panconesi, A. and Rizzi, R., Some simple distributed algorithms for sparse networks. *Distributed Computing*, 14,2(2001), pages 97–100.

PY93 Papadimitriou, C. H. and Yannakakis, M., Linear programming without the matrix. *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC, San Diego, CA, USA, May 1993)*, New York, NY, USA, 1993, ACM Press, pages 121–129.

Ram30 Ramsey, F. P., On a problem of formal logic. *Proceedings of the London Mathematical Society*, 30(1930), pages 264–286.

Suo11 Suomela, J., Survey of local algorithms. *ACM Computing Surveys*, (2011). URL `http://www.cs.helsinki.fi/local-survey/`. To appear.

YK88 Yamashita, M. and Kameda, T., Computing on anonymous networks. *Proc. 7th Annual ACM Symposium on Principles of Distributed Computing (PODC, Toronto, Ontario, Canada, August 1988)*, New York, NY, USA, 1988, ACM Press, pages 117–130.