Karlsruhe Institute of Technology

On the Distributed Computation of Fractional Connected Dominating Set Packings

Bachelor Thesis of

Matthias Wolf

At the Department of Informatics
Institute of Theoretical Computer Science

Reviewers:	Prof. Dr. Dorothea Wagner
	Prof. Dr. Peter Sanders
Advisor:	Dipl.-Inform. Fabian Fuchs

Time Period: 1st July 2014 – 30th September 2014

Acknowledgement

I would like to thank Prof. Dr. Dorothea Wagner for giving me the possibility to write my thesis at her institute. I also want to thank my advisor Fabian Fuchs for interesting discussions, his guidance and constructive feedback. Finally, I would like to thank my parents for their support.

Statement of Authorship

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Karlsruhe, 30th September 2014

Abstract

The fundamental goal of communication networks is to transfer messages from one node to other nodes. Often, one is interested in maximizing the information flow, which is limited by the connectivity of the network. If we focus on vertex-connectivity, connected dominating sets (CDS) are a valuable tool. Having multiple (fractionally) vertex-disjoint connected dominating sets, which we call a fractional CDS packing, allows to get an information flow matching the size of the packing.

In this thesis, we present a distributed algorithm that given a communication network with n nodes and vertex-connectivity k computes a fractional CDS packing of size $\Omega(k/\log n)$. It takes $O(\log^2 n \cdot (D + \sqrt{n \log n} \log^* n + k\Delta))$ rounds in the V-CONGEST model, where D denotes the diameter of the network and Δ the maximum degree. For networks of not too large vertex-connectivity, our algorithm achieves a better runtime than the currently best known algorithm of Censor-Hillel, Ghaffari, and Kuhn [CHGK14a].

We also show how the runtime can be improved to $O(\log^2 n \cdot (D + \sqrt{n \log n} \log^* n + k))$ if we work in the less restricted E-CONGEST model.

Deutsche Zusammenfassung

Das Ziel von Kommunikationsnetzwerken ist es, Nachrichten von einem Knoten zu anderen zu schicken. Häufig interessiert man sich dafür, den Informationsfluss zu maximieren. Dieser wird durch den Zusammenhang des Netzwerkes begrenzt. Bei der Betrachtung des Knotenzusammenhangs haben sich „connected dominating sets“ (CDS) als hilfreich erwiesen. Packt man mehrere connected dominating sets in das Netzwerk, erhält man ein „fractional CDS packing“. Dieses kann verwendet werden, um einen Informationfluss in der Größe der Packung zu erhalten.

In dieser Arbeit stellen wir einen verteilten Algorithmus vor, der zu einem Netzwerk mit n Knoten und Knotenzusammenhang k eine solche Packung der Größe $\Omega(k/\log n)$ bestimmt. Dieser benötigt $O(\log^2 n \cdot (D + \sqrt{n \log n} \log^* n + k\Delta))$ Runden im V-CONGEST-Modell, wobei D den Durchmesser und Δ den größten auftretenden Knotengrad bezeichnet. In Netzwerken, deren Knotenzusammenhang nicht zu groß ist, erreicht unser Algorithmus eine bessere Laufzeit als der bislang schnellste bekannte Algorithmus von Censor-Hillel, Ghaffari und Kuhn [CHGK14a].

Darüber hinaus erläutern wir, wie die Laufzeit auf $O(\log^2 n \cdot (D + \sqrt{n \log n} \log^* n + k))$ Runden verringert werden kann, wenn wir das mächtigere E-CONGEST-Modell annehmen.

Contents

1. Introduction

The goal of communication networks is to transfer information from one node to another. Normally, there is more than one node that wants to send a message at one time. Therefore, one is interested in a method that optimizes the information flow, i.e. the number of messages sent across the network per round. A fundamental measure that limits the information flow is the connectivity of the network. There are two measures for connectivity, edge and vertex connectivity, which describe the size of the minimum edge and vertex cuts, respectively. Since edge and vertex cuts limit the information flow, we intuitively expect that the larger the connectivity of a graph, the more information can be transfered in one step.

This is illustrated by Menger's Theorem [BM08, Theorem 9.1]: Given a graph with edge connectivity λ or vertex connectivity k and two nodes v and w, there are λ edge-disjoint or k internally vertex-disjoint paths from v to w, respectively. Hence, using these paths, it is possible to send λ or k messages from v to w at the same time, respectively. Each message is transmitted along one of these paths. But what happens if we deal with more than one pair of nodes? Applying Menger's Theorem to each pair gives paths from the sender to the receiver. However, it is unclear how the paths of different pairs of nodes work together. It is possible that the paths of different pairs are not disjoint. Hence, the messages from different nodes might interfere, and thus, the throughput is decreased.

Therefore, a different approach is needed. We focus on the vertex connectivity and give a rather natural method, which divides the nodes of a graph in (fractionally) vertex-disjoint connected dominating sets (CDS). Our algorithm is based on the fractional dominating tree packing algorithm of Censor-Hillel, Ghaffari, and Kuhn [CHGK14a]. They call this approach *connectivity decomposition* as the graph is decomposed in smaller (fractionally) vertex-disjoint subgraphs, which almost preserve the connectivity. These subgraphs can be used to achieve an information flow that almost matches the connectivity of the graph.

For example, [CHGK14a, Appendix A] shows how these fractionally vertex-disjoint connected dominating sets can be used for the *gossiping* problem, which is also known as *all-to-all broadcast*. The basic idea is to send each message along one connected dominating set such that the messages are almost evenly distributed among these connected dominating sets. As the connected dominating sets are fractionally vertex-disjoint, they can be used simultaneously to send messages, which leads to a high throughput.

Applied to a network with n nodes, vertex-connectivity k and maximum degree Δ our algorithm finds a fractional CDS packing of size $\Omega(k/\log n)$. It takes $O(\log^2 n(D + \sqrt{n \log n} \log^* n + k\Delta))$ rounds in the V-CONGEST model and $O(\log^2 n(D + \sqrt{n \log n} \log^* n + k))$ in the less restricted E-CONGEST model.

Our algorithm works on a virtual graph that consists of $\Theta(\log n)$ copies of the original graph. These copies are arranged in layers. The *lower* layers contain one copy and the *upper* layers contain two copies each. The two copies in each upper layer have types 1 and 2. A formal definition of the virtual graph is given in Section 3.1.

The algorithm consists of two parts. In the first one, we assign each node of the lower layers to a randomly chosen class. According to Theorem 4.5, which was first presented in [CHGK14b, Lemma 3.2], this gives domination for each class with high probability (w.h.p.), which means that the probability that this happens is at least $1 - 1/n^c$ for some constant $c \geq 1$.

The classes may be unconnected after this step. Thus, the second part seeks to establish connectivity by gradually adding nodes to the classes. We consider the upper layers one after another. Assume that we have already assigned class numbers to the nodes of levels 1 to ℓ. We call the virtual nodes of layer $\ell + 1$ *new* and the nodes of layers 1 to ℓ *old* nodes. We then assign classes to the new nodes using the following steps:

1. We determine the connected components of the old nodes.

2. All new nodes of type 1 choose a class randomly. Components that are now connected to other components of the same class become inactive.

3. For each active component $\mathcal{C}$, we determine a maximal set of paths that connect $\mathcal{C}$ with another component of the same class. In compliance with [CHGK14b], we call these paths *long connector paths* (cf. Section 3.2 for a definition). Note that there must exist such paths of length at most 3, due to the domination of the classes (cf. Lemma 3.2). To find these paths we apply a distributed maximal matching algorithm to bipartite virtual graphs, which are simulated by the real graph G. This step is the key part of the algorithm. A detailed description is given in Section 3.4.

4. Now each type-2 new node v knows all connector paths going through it and the classes of the components they belong to. The node v then discards a path if the type-1 internal node on this path has not chosen the class the path belongs to. All remaining paths have the property that if v chooses their class they establish a connection between two components of the same class. Finally, v chooses one remaining path randomly and assigns its class to itself.

After we have run these steps for all layers each class represents a connected dominating set w.h.p. This is shown in Theorem 4.4.

1.1 Related Work

Fractional CDS packings can be used to achieve a high throughput when broadcasting multiple messages [CHGK14a, Appendix A]. There are other approaches to this problem, which is called the *multi-message broadcast* problem. Many of these focus on the *radio network* model, which was introduced in [CK85]. In this model each node decides whether it sends or receives in the current round. Moreover, receiving a message is only successful if only one neighbor sends a message at that time.

Bar-Yehuda et al. [BYII89] presented a routing based algorithm that asymptotically needs $O(\log \Delta \log n)$ rounds per package in expectation, where n and Δ denote the number of nodes and the maximum degree, respectively. Using network coding Khabbazian and Kowalski improved this number of rounds per package to $O(\log \Delta)$ [KK11]. Another approach was introduced by Ghaffari and Haeupler [GH13], which asymptotically needs $O(\log n)$ rounds per package. This is worse than the $O(\log \Delta)$ rounds by Khabbazian and Kowalski. However, it takes less time for the initialization as the network is structured differently in what they call *collision-free layers*.

Other works establish bounds for the runtime that is needed. For example, it has been shown by Ghaffari et al. [GHK13] that there are networks in which broadcasting p packages takes $\Omega(p \log n)$ rounds in the radio network model.

The following two results are the ones our approach is based on. Therefore, we give a more detailed description.

Censor-Hillel, Ghaffari, and Kuhn showed that each graph G with n nodes and vertex-connectivity k has a fractional CDS packing of size $\Omega(k/\log n)$ and that this bound is optimal [CHGK14b]. To show the existence of a large fractional CDS packing, they present a centralized algorithm that outputs a fractional CDS packing of size $\Theta(k/\log n)$ w.h.p. This algorithm introduces the structure of the algorithm we use and the layered virtual graph and the notion of connector paths. The optimality of the bound is shown by building a graph whose largest fractional CDS packing is of size $O(k/\log n)$.

The same authors also presented a distributed algorithm that finds a fractional CDS packing of size $\Omega(k/\log n)$ in $O(\log^3 n \cdot \min\{(n \log n)/k, D + \sqrt{n \log n} \log^* n\})$ rounds in the V-CONGEST model [CHGK14a]. Our approach is directly based on this algorithm.

There are $t = \Theta(k)$ classes that shall form a CDS after the algorithm is executed. They work on a virtual graph $\mathcal{G}$ that is similar to the one we present in Section 3.1. Their virtual graph is organized in L layers each consisting of 3 copies of each real node. The copies are of types 1, 2 and 3, respectively.

The nodes in the first half of the layers join random classes, which gives domination w.h.p. Each of the remaining layers is considered on its own. For each layer ℓ the following steps are taken. They find the connected components of nodes of the same class in layers 1 to $\ell - 1$. Then, the type-1 and type-3 nodes in layer ℓ randomly select a class. If they connect two components of the same class, these components become inactive.

In the next step, the bipartite *bridging graph* is built as follows. The active components are the nodes of one side of the bipartite graph. Hence, we act as if all nodes of the component are contracted to one node. The other side of the graph is formed by the type-2 nodes of layer ℓ. These are the nodes that still have to choose a class. There is an edge between an active component $\mathcal{C}$ of class i and a type-2 node v iff v has a neighbor in $\mathcal{C}$ and a neighbor w of type-3 in layer ℓ that joined class i and has a neighbor in a component $\mathcal{C}' \neq \mathcal{C}$ of class i. This rule corresponds to the intuition that v and $\mathcal{C}$ are neighbors in the bridging graph if and only if the component $\mathcal{C}$ can be connected to another component $\mathcal{C}'$ via nodes v and w. In other words: If v joins class i, the component $\mathcal{C}$ is joined with the component $\mathcal{C}'$.

As the final step for each layer, they apply a maximal matching algorithm to the bridging graph. Each type-2 node that gets matched to a component $\mathcal{C}$ joins the class of the component. All unmatched nodes choose a random class.

The following section contains a comparison of our algorithm and the one we just presented.

1.2 Comparison with [CHGK14a]

In this section, we compare our approach to the one of Censor-Hillel, Ghaffari, and Kuhn [CHGK14a], which is the base for our algorithm. We use the V-CONGEST model to compare these two algorithms.

If we have a look at the running times of the two algorithms, we see that our algorithm has round-complexity $O(\log^2 n(D + \sqrt{n \log n} \log^* n + k\Delta))$ compared to $O(\log^3 n \cdot \min\{D + \sqrt{n \log n} \log^* n, (n \log n)/k\})$ of their algorithm.

Both algorithms depend on the vertex-connectivity k. However, our algorithm becomes slower when k increases while theirs becomes faster. The increase in the running time is due to the fact that we calculate one matching per class and the number of classes is proportional to k. But if the vertex-connectivity k and the maximum degree Δ are not "too large", our algorithm is faster.

If we assume $D+\sqrt{n \log n} \log^* n = O(n \log n/k)$, the algorithm of Censor-Hillel, Ghaffari, and Kuhn takes $O(\log^3 n(D+\sqrt{n \log n} \log^* n))$ rounds. In this case our algorithm is faster if $k\Delta = o(\log n(D+\sqrt{n \log n} \log^* n))$. The improvement we achieve in this case is up to a factor of $\Theta(\log n)$.

Where does this improvement come from? To answer this question we need a more detailed comparison of the steps of the algorithms. The general structure of both algorithms is the same. They create a layered virtual graph. Note that the exact definition of this virtual graph varies, e.g. there are two or three node types, but these are technical details. In fact, both algorithms could work on the other graph with only minor modifications. For example, the algorithm of Censor-Hillel, Ghaffari, and Kuhn could remove the nodes of type 3 and use those of type 1 instead.

Both assign random classes to some part of the virtual nodes and then try to merge the classes. They go over the remaining nodes layer by layer and do the following steps: First, they find the connected components of nodes that belong to the same class. Then, some fraction of the nodes in the layer (type 1 vs. type 1 and 3) randomly join a class.

They differ in how the remaining type-2 nodes select their classes. Both algorithms use connector paths (cf. Section 3.2 for a formal definition) that connect components of the same class. We determine them explicitly while the algorithm of Censor-Hillel, Ghaffari, and Kuhn uses them only implicitly. They are only mentioned in the proof that the algorithm is correct.

They calculate a matching on a graph, which they call the bridging graph. Some nodes of this bridging graph are simulated by connected components of nodes. Hence, all nodes in the same connected component must agree on a common strategy. Therefore, the matching algorithm needs $O(\log^2 n \cdot \min\{D+\sqrt{n \log n} \log^* n, (n \log n)/k\})$ rounds. The last factor describes the time needed for the dissemination of information and the common strategy across the component.

In our algorithm all nodes in the matching graph are simulated by one real node. Thus, one application of the matching algorithm takes $O(\Delta \log n)$ rounds. Note that the matching graphs can be simulated by the real graph with only a constant overhead. As we calculate one matching per class, the total running time of the algorithm is $O(k\Delta \log n)$. If we compare these two running times, we can see that they form exactly the gap in the total running time.

The final step in each layer is that the type-2 nodes choose a class. As the previous step is already different, the strategies vary. In the algorithm of Censor-Hillel, Ghaffari, and Kuhn the type-2 nodes select the class of the component they are matched to. In our algorithm they determine the useful paths, i.e. paths that can still connect components of the same class (cf. Section 3.2), pick one randomly and join the class of this path.

1.3 Outline

In the following chapter, we define the basic terms and notations that are used throughout this thesis. We then present the fractional CDS packing algorithm (Chapter 3) and prove its correctness (Chapter 4). Finally, we summarize the contribution of this thesis in Chapter 5 and give a brief outlook on possible future research.

2. Preliminaries

This chapter contains basic definitions and statements that are used in this thesis.

2.1 Graphs

An *undirected graph* $G = (V, E)$ consists of a set of nodes V and a set of edges $E \subseteq \{\{v, w\} \mid v, w \in V\}$. A *subgraph* of G is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E \cap \{\{v, w\} \mid v, w \in V'\}$, i.e. all edges in E' belong to E and their endpoints lie in V'. A graph is *bipartite* if there are two disjoint sets of nodes V_1 and V_2 such that $V = V_1 \cup V_2$ and all edges have one endpoint in V_1 and one in V_2.

Let $V' \subseteq V$ be a set of nodes. We denote the subgraph that is induced by V' with $G[V']$. This means that $G[V'] = (V', E')$, where $E' = E \cap \{\{v, w\} \mid v, w \in V'\}$ contains exactly those edges that have both endpoints in V'.

A *path* from v_0 to v_ℓ is a sequence of nodes $(v_0, \ldots, v_\ell)$ such that there is an edge between v_i and v_{i+1} for $i = 0, \ldots, \ell - 1$. The nodes v_0 and v_ℓ are called *endpoints* and the other ones are *internal* nodes. A path is *simple* if the nodes $v_0, \ldots, v_{\ell-1}$ are pairwise distinct. In this thesis, we assume that all paths we work with are simple. The *length* of the path $(v_0, \ldots, v_\ell)$ is ℓ. Intuitively, it describes the number of steps that are needed to go from one endpoint to the other. A set of paths is *internally vertex-disjoint* iff each node is an internal node of at most one path in this set.

The *distance* $\operatorname{dist}(v, w)$ of two nodes $v, w \in G$ is the length of the shortest path from v to w. The largest distance of two nodes of graph G is the *diameter* D of this graph, i.e. $D = \max\{\operatorname{dist}(v, w) \mid v, w \in V\}$.

The graph $G = (V, E)$ is *connected* if for each pair of nodes $v, w \in V$ there exists a path from v to w in G. A *connected component* is a maximal subset of nodes $C \subseteq V$ such that $G[C]$ is connected. This implies that if we add any node $v \in V \setminus C$ the graph induced by $C \cup \{v\}$ is not connected. The graph G is *k'-vertex-connected* if the graph is still connected if we remove an arbitrary set of k' nodes. In other words, the graph is k'-vertex-connected iff for each set $S \subseteq V$ with $|S| = k'$ the graph $G[V \setminus S]$ is still connected. The *vertex connectivity* of G is the largest number k such that G is k-vertex-connected.

A subset $S \subseteq V$ of the nodes is a *dominating set* if each node $v \in V \setminus S$ has a neighbor in S. In this case, we also say that the set S *dominates* the graph G. A dominating set S is called a *connected dominating set* (CDS) if the subgraph $G[S]$ is connected.

A *fractional connected dominating set packing* is a set $\mathcal{S}$ of connected dominating sets, where each set $S \in \mathcal{S}$ has a weight $x_S \in [0, 1]$ associated with it. For each node $v \in V$ let $\mathcal{S}_v \subseteq \mathcal{S}$ be the connected dominating sets that contain v. The sum of the weights of these node has to be at most 1, i.e. $\sum_{S \in \mathcal{S}_v} x_S \leq 1$. The total weight of all sets $\sum_{S \in \mathcal{S}} x_S$ is the *size* of the fractional CDS packing.

An *independent set* is a set $S \subseteq V$ of nodes such that $v, w \in S$ implies that $\{v, w\} \notin E$. In other words, for each node $v \in S$ the set S does not contain a neighbor of v.

A set $M \subseteq E$ of edges is called a *matching* iff no two edges in M have a common endpoint, i.e. if $e, e' \in M$ and $e \neq e'$ then the two sets are disjoint, i.e. $e \cap e' = \emptyset$. The *size* of a matching is the cardinality of the set M. A *maximum matching* is a matching that has maximum size of all possible matchings in the graph G. A matching M is said to be *maximal* if there is no matching $M' \subseteq E$ such that $M \subsetneq M'$. Hence, if we add an edge to a maximal matching, the result is not a matching anymore. Note that each maximum matching is maximal but the converse does not hold. It is a well-known fact that the size of a maximal matching is at least half of the size of a maximum matching.

Matchings and independent sets are quite similar. In fact, a matching can be seen as an independent set of edges. For a precise statement, we need the notion of a *line graph*, which basically switches the roles of nodes and edges. Let $G = (V, E)$ be a graph. The line graph $L(G) = (V', E')$ of G is defined as follows: For each edge e in the graph G, the line graph $L(G)$ contains a node v_e. There exists an edge between two different nodes v_e and $v_{e'}$ if and only if the edges e and e' have a common endpoint in G.

An example of this construction can be found in Figure 2.1. It also highlights the correspondence of matchings in G and independent sets in the line graph $L(G)$, which we show in the following lemma.

Lemma 2.1. *Let $G = (V, E)$ be a graph and $L(G)$ its line graph.*

(a) If $M \subseteq E$ is a matching in G, the set $\{v_e \mid e \in M\}$ forms an independent set in $L(G)$.

(b) If $S \subset V'$ is an independent set in $L(G)$, the set $\{e \mid v_e \in S\}$ forms a matching in G.

Proof. (a) Let $M \subseteq E$ be a matching in G. We show that $S := \{v_e \mid e \in M\}$ is an independent set of $L(G)$. We need to show that the nodes $v_e, v_{e'} \in S$ are not adjacent. Since e and e' are both included in the matching M, they do not share an endpoint. By definition of $L(G)$, this implies that there is no edge between v_e and $v_{e'}$. As this is true for all pairs of nodes in S, the set S forms an independent set of $L(G)$.

(b) Suppose that $S \subseteq V'$ is an independent set. This induces the set $M := \{e \mid v_e \in S\}$, which is a subset of the edges of G. We show that no two elements of M have a common endpoint. Fix two edges $e, e' \in M$. Due to the definition of M the independent set S contains both nodes v_e and $v_{e'}$. Therefore, there is no edge between these two nodes. According to the construction of $L(G)$, this is equivalent to the fact that the edges e and e' do not have a common endpoint. As the choice of e and e' was arbitrary, the set M is a matching. □

A consequence of this correspondence is that a maximal independent set in the line graph $L(G)$ translates to a maximal matching in G.

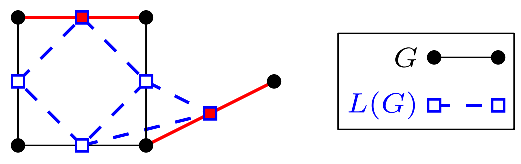

Figure 2.1: Example of relationship between matchings in G and independent sets in $L(G)$. The marked nodes and edges show an independent set in $L(G)$ and the corresponding matching in G.

2.2 Distributed Computing

When talking about distributed computing there are several different models that are used (see e.g. [Pel00, Section 2.3]). In this thesis we focus on one that is called the V-CONGEST model. Given a communication network (i.e. an undirected graph) with n nodes, we assume that initially each node knows its unique ID and the IDs of its neighbors. Hence, it only has a local view of the network.

The nodes of a graph communicate in synchronous rounds. Each round consists of the following parts: (1) local computation, (2) sending one message to all neighbors, and (3) receiving the messages from its neighbors. The size of each message is bounded by $O(\log n)$. Hence, in each round each node can send one message of size $O(\log n)$ to all of its neighbors. Note that in this model a node sends the same message to all of its neighbors.

This restriction is removed in the E-CONGEST model, which is also called the $\mathcal{CONGEST}$ model. Here, each node can send different messages to its neighbors. But still each message must have $O(\log n)$ bits.

We present our algorithm in the V-CONGEST model and explain how the running time can be improved if we use the E-CONGEST model instead.

2.3 Mathematical Background

Given a graph with n nodes we use the term *with high probability* (w.h.p.) to indicate that the probability is at least $1 - 1/n^c$ for a constant $c \geq 1$.

The following three lemmas contain inequalities, which we need to show the correctness of our algorithm.

Lemma 2.2 (Markov's Inequality). *Let X be a nonnegative random variable with expected value $\mathbb{E}[X]$. For all $a > 0$ the following inequality holds:*

$$\Pr[X \geq a] \leq \frac{\mathbb{E}[X]}{a}$$

Proof. A proof can be found in [Kle14, Theorem 5.11]. □

The second inequality describes an upper bound for the values of binomial coefficients.

Lemma 2.3. *For natural numbers n and $k \leq n$ we have*

$$\binom{n}{k} \leq \left(\frac{en}{k}\right)^k$$

Proof. A proof is shown in [CLRS09, Equation C.5]. □

The third lemma contains an inequality that deals with exponential functions.

Lemma 2.4. *The following inequality holds for $t \geq 1$ and $0 \leq \alpha \leq t$:*

$$\left(1 - \frac{\alpha}{t}\right)^t \leq e^{-\alpha}$$

Proof. This inequality is presented in [MR95, Proposition B.3] □

We presented the basic terms and statements used in this thesis. The following chapter contains a description of our fractional CDS packing algorithm.

3. Fractional Connected Dominating Set Packing Algorithm

In this chapter we present our fractional CDS packing algorithm. We start with the definitions of the virtual graph (Section 3.1) and connector paths (Section 3.2). The remaining sections contain the description of the algorithm.

3.1 The Virtual Graph

We use the general idea of the layered virtual graph, which was introduced in [CHGK14b] but build it in a slightly different way.

Given a graph $G = (V, E)$, we construct a virtual graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, which can easily be simulated by G. Each node $v \in G$ simulates $L_1 + 2L_2 = \Theta(\log n)$ copies. These nodes are ordered in $L = L_1 + L_2 = \Theta(\log n)$ layers, which are numbered from 1 to L. We call the first L_1 layers the *lower* layers and the remaining ones *upper* layers. The lower layers contain one copy of each node whereas the upper ones contain two copies each, one of type 1 and one of type 2.

To distinguish the nodes of the two graphs, we refer to the nodes of G and $\mathcal{G}$ as *real* and *virtual* nodes, respectively. The *projection* $\Psi(v)$ of a virtual node v is the real node that simulates v. Likewise, the projection of a set $\mathcal{S}$ of virtual nodes is the union of the projection of the elements, i.e. $\Psi(\mathcal{S}) = \{\Psi(v) \mid v \in \mathcal{S}\}$.

There is an edge between two virtual nodes v and w if they are projected to the same real node, i.e. $\Psi(v) = \Psi(w)$, or there exists an edge in G between their projections $\Psi(v)$ and $\Psi(w)$. An example of an virtual graph is shown in Figure 3.1.

3.2 Connector Paths

An important tool for connecting components in the virtual graph are *connector paths*. The notion of these paths was first developed in [CHGK14b].

Suppose that all nodes of levels 1 to ℓ have already selected their classes. We call these nodes *old nodes*. Consider a class i and the set nodes of this class in layers 1 to ℓ, which we denote with $\mathcal{V}_\ell^i$. Let $\mathcal{C}$ be a connected component of the graph $\mathcal{G}[\mathcal{V}_\ell^i]$, which is induced by the old nodes of class i. Assume that $\mathcal{C}$ is not single in its class, i.e. we still need to connect $\mathcal{C}$ to another component of class i. Let $C = \Psi(\mathcal{C})$ be the projection of $\mathcal{C}$ to the real graph G. We call a path P in the real graph G a *potential connector* if the following conditions hold.:

(a) One endpoint of P is in C and the other is in $\Psi(\mathcal{V}_\ell^i \setminus \mathcal{C})$, i.e. P connects C and another component of class i.

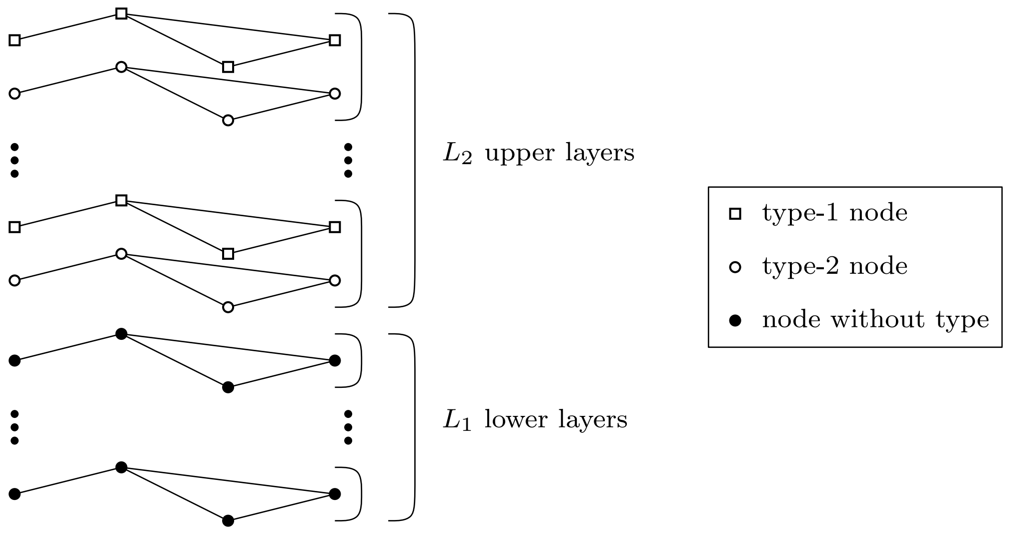

Figure 3.1: Example of virtual graph $\mathcal{G}$. The edges between nodes of different layers or types are not shown.

(b) P has at most two internal nodes.

(c) If P has exactly two internal nodes, i.e. it is of form (s, v, w, t) where $s \in C$, v has no neighbor in $\Psi(\mathcal{V}_\ell^i \setminus \mathcal{C})$ and w has no neighbor in C.

The last condition ensures that the potential connector paths cannot be made shorter by removing one of the internal nodes and possibly selecting a new endpoint.

From the potential connector P we construct a *connector path* $\mathcal{P}$ such that the projection of $\mathcal{P}$ is P. Assume that the endpoints of P are s and t. We then select virtual nodes $s', t' \in \mathcal{V}_\ell^i$ of class i, such that their projections are s and t, respectively. For the internal nodes of $\mathcal{P}$ we choose nodes from layer $\ell + 1$. If P has exactly one internal node v, we select the type-1 node that is projected to v. If otherwise P has two internal nodes, let v be the node that is adjacent to a node in C and w be the other internal node. We choose the copy of v with type 2 and the copy of w with type 1. We say that the connector path $\mathcal{P}$ *belongs to* component $\mathcal{C}$ and class i.

Figure 3.2 shows an example of connector paths. Note that the path from the projection of component $\mathcal{C}$ via nodes v_2 and w_2 to the projection of $\mathcal{C}_2$ is not a valid potential connector path. It can be shortened as node v_2 is adjacent to a node in $\Psi(\mathcal{C}_1)$. Hence, it violates condition (c), which requires the paths to be minimal. In Section 3.5, we analyze properties of connector paths. This includes an intuitive explanation of why there are "many" connector paths per component (cf. Lemma 3.2).

A connector path with one internal node is called a *short connector path*, whereas a connector path with two internal nodes is called a *long connector path*. Moreover, we call a long connector path belonging to a component of class i *useful* if its type-1 internal node has assigned itself to class i. Otherwise, the path is *useless*. This corresponds to the intuition that useless connector paths cannot connect the two components of class i anymore since one internal node has chosen the "wrong" class. We also say that a connector path *selects class i* if all internal nodes join class i.

3.3 Description of the Algorithm

Given a graph $G = (V, E)$ with n nodes, diameter D and vertex-connectivity k, our goal is to find $\Theta(k)$ connected dominating sets such that each node of G is contained in $O(\log n)$ sets. To get the desired packing, we assign a weight of $\Omega(1/\log n)$ to each set. This gives a fractional CDS packing of size $\Omega(k/\log n)$.

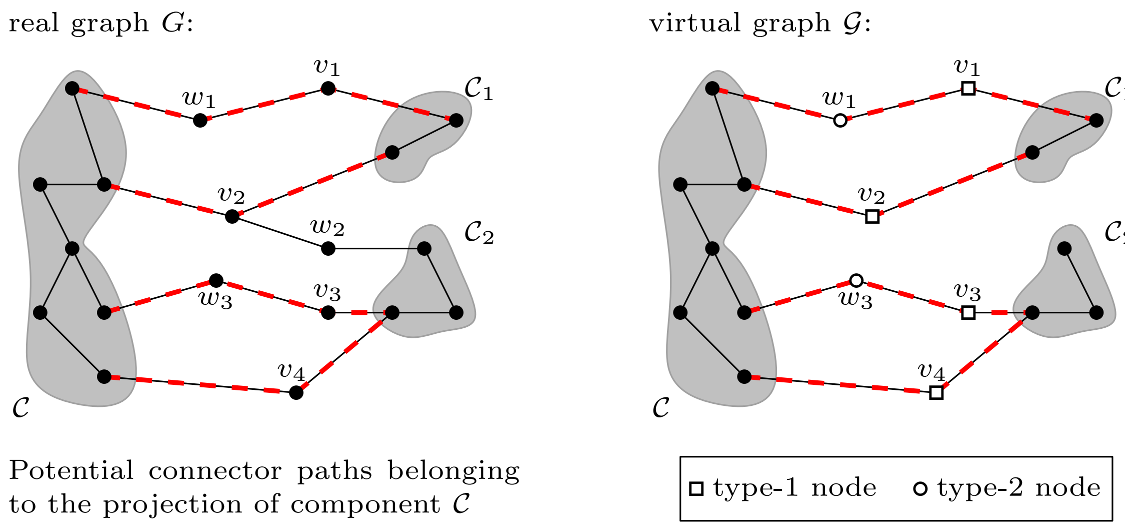

Figure 3.2: Potential connector paths and the corresponding connector paths belonging to component $\mathcal{C}$

We create $t = \alpha k$ classes for a constant $\alpha > 0$. During the execution of the algorithm all virtual nodes join exactly one class. They select a class such that in the end all classes are connected dominating sets w.h.p., and each real node is contained in $O(\log n)$ classes. Giving all classes a weight of $L_1 + 2L_2$ and projecting them to the real graph leads to the desired fractional CDS packing, in which each node is contained in $O(\log n)$ classes.

Recall that for a layer ℓ and a class i we denote the virtual nodes of layers 1 to ℓ that belong to class i with $\mathcal{V}_\ell^i$. Moreover, let the set $\mathcal{V}_\ell$ contain all nodes in these layers, i.e. it is the union of $\mathcal{V}_\ell^i$ where i ranges over all classes.

Throughout the algorithm we use the following protocol, which was presented by Censor-Hillel, Ghaffari, and Kuhn. We refer the reader to their article for a more detailed description. The protocol is based on the connected component identification algorithm of Thurimella [Thu97, Algorithm 5] combined with the MST-algorithm of Kutten and Peleg [KP95].

Theorem 3.1 (Theorem B.2 of [CHGK14a]). *Let $G = (V, E)$ be a graph with n nodes and diameter D and $G' = (V, E')$ be a subgraph of G where $E' \subseteq E$. Suppose each node has a value with $O(\log n)$ bits. There is a distributed algorithm taking $O(\min\{D', D + \sqrt{n}\log^* n\})$ rounds in the V-CONGEST model that lets each node know the smallest value of all nodes in the same connected component of G'. Here, D' denotes the largest diameter among all connected components of G'.*

The algorithm consists of two parts: After the first one all classes dominate the graph G. In the second one we make all classes connected by adding nodes to the classes.

Part A – Lower layers

In the first part of the algorithm each virtual node in the lower layers randomly joins a class obeying a uniform distribution.

Part B – Upper layers

We then consider the upper layers one after another. Suppose the virtual nodes of layers 1 to ℓ have already selected a class. We call the nodes of these layers *old* nodes and nodes of layer $\ell + 1$ *new* nodes. We perform the following steps to determine the classes of all new nodes.

Step B.1 – Identify connected components of old nodes

We determine the connected components of the old nodes of the virtual graph by applying the protocol of Theorem 3.1, where we use the node IDs as the values. This step is identical to the step presented in Section B.1 of [CHGK14a].

After that each node knows the ID of the component it belongs to. It then sends this information to all neighbors.

Step B.2 – Type-1 new nodes select class

After the components have been determined, all type-1 new nodes randomly choose a class and send their choices to their neighbors. Additionally, each node v includes the CONNECTOR-symbol if it connects two components. This means that if v has chosen class i it searches for two old neighbors that both belong to class i but not to the same component. This is easy to determine since v already knows the classes and component IDs of its old neighbors.

If a virtual node w receives a message containing the CONNECTOR-symbol and the class it belongs to, it knows that its component has been connected and it has to become inactive. This information has to be shared by all nodes in the component, which is done by an other application of the algorithm of Theorem 3.1.

The deactivation of components is identical to the first part of Section B.2 in [CHGK14a]. As the details of this step can be found there, we omit them here.

Step B.3 – Determine internally vertex-disjoint connector paths

For each active component we find internally vertex-disjoint long connector paths by applying a maximal matching algorithm. We run this algorithm once for each class i on a bipartite graph $\mathcal{H}_i$, which represents the possible long connector paths of all active components of this class.

As this is the key part of the algorithm, we give a detailed description of this step in the next section.

Step B.4 – Type-2 new nodes select class

In the last step each type-2 new node v determines which connector paths going through it are useful, i.e. the internal type-1 node of this path has chosen the right class. It then chooses one of the remaining paths randomly and joins the class the connector path belongs to. Both internal nodes of this class have joined the correct class, and hence, the path connects two components of the same class. If no path is useful, v randomly selects any class.

Final Part – Assign Weights

After all virtual nodes have selected their class, we assign each class a weight of $1/(L_1 + 2L_2)$ as each real node simulates exactly $L_1 + 2L_2$ virtual copies of itself.

After we completed all the steps, the classes are connected dominating sets w.h.p. Together with the weights they form a fractional CDS packing. It still remains to describe how to find the vertex-disjoint, long connector paths, which we do next.

3.4 Finding Vertex-Disjoint Connector Paths

In this section we give a detailed description of how we find internally vertex-disjoint long connector paths for each active connected component. Note that we might not determine a maximum set of these paths. But we make sure that the number of paths we find is at least half of the maximum number. In Lemma 3.2 we show that each connected component has at least k connector paths.

Finding many long connector paths is especially interesting if an active component has at least $k/2$ internally vertex-disjoint long connector paths, which is the case we need to show the correctness of the algorithm (cf. Lemma 4.7). In this case we find at least $k/4$ vertex-disjoint connector paths.

In order to find these paths, we build one graph per class, which itself is the disjunct union of bipartite graphs, one for each active component of this class. Each edge represents a long connector path. Basically, this graph is determined by taking all long connector paths of this component and removing their endpoints, i.e. only the internal nodes and the edges between them remain. A formal definition is given in Section 3.4.1. In Section 3.4.2 we describe how the graph can be found algorithmically. A matching in this graph induces a set of internally vertex-disjoint connector paths of this component. Therefore, we use a maximal matching algorithm to find these paths, which is presented in Section 3.4.3.

3.4.1 The Virtual Graph $\mathcal{H}_i$

In this section we formally define the virtual graph $\mathcal{H}_i$ for class i, which we use to determine the sets of internally vertex-disjoint connector paths of this class.

Fix a class i. We obtain the graph $\mathcal{H}_i$ as the disjunct union of bipartite graphs, one for each active component of class i. Hence, we only need to describe how the graph is built for an active component $\mathcal{C}$ of class i.

Let $C = \Psi(\mathcal{C})$ be its projection to the real graph G. For each type-2 new node v we add a node $v_\mathcal{C}$ to $\mathcal{H}_i$ if and only if the following three conditions hold:

(a) The projection of v does not belong to C.

(b) v has a neighbor in $\mathcal{C}$.

(c) No old neighbor of v belongs to a component of class i other than $\mathcal{C}$.

These conditions ensure that v might be the type-2 internal node of a long connector path of component $\mathcal{C}$.

For each type-1 node w that is a neighbor of any type-2 node we have chosen above, we add a virtual node $w_\mathcal{C}$ if w has a neighbor in a component $\mathcal{C}' \neq \mathcal{C}$ of class i but no neighbor in $\mathcal{C}$. We then include an edge between two nodes $v_\mathcal{C}$ and $w_\mathcal{C}$ in $\mathcal{H}_i$ if there is an edge between the nodes v and w in $\mathcal{G}$.

This construction ensures that there exists a long connector path of component $\mathcal{C}$ with internal nodes v and w if and only if there is an edge between $v_\mathcal{C}$ and $w_\mathcal{C}$ in $\mathcal{H}_i$. Hence, each matching in $\mathcal{H}_i$ represents a set of internally vertex-disjoint long connector paths for components of class i.

Figure 3.3 shows an example of a graph G and the corresponding graph $\mathcal{H}_i$. Note that there is no copy of node z in the virtual graph $\mathcal{H}_i$. To be included in $\mathcal{H}_i$ there are two possibilities: either as a copy of a type-1 or of a type-2 node. There exists no type-2 copy of z because z has neighbors in two different connected components of class i. To

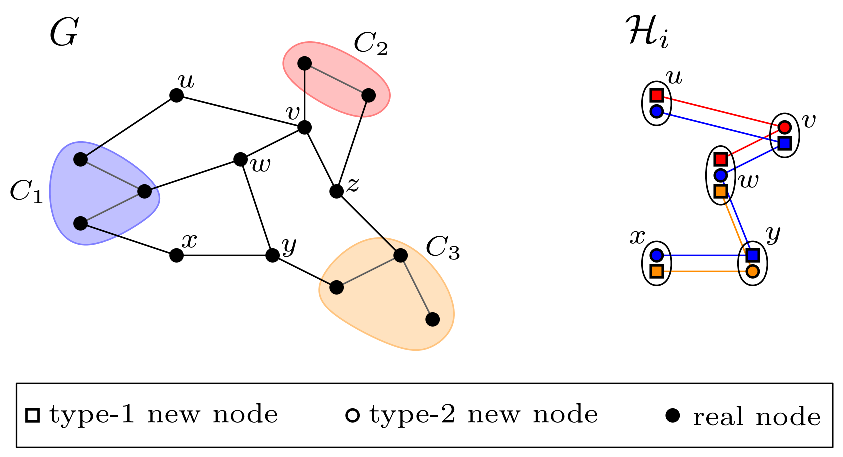

Figure 3.3: Example graph G and the corresponding graph $\mathcal{H}_i$

be included as the copy of a type-1 node, there has to be a neighbor of type-2 included in $\mathcal{H}_i$. This is indeed the case: The node v has a type-2 copy in $\mathcal{H}_i$, which belongs to component $\mathcal{C}_2$. But z also has a neighbor in the same component. Therefore, we do not add a copy of z to $\mathcal{H}_i$.

The arguments given refer to the definition of the graph $\mathcal{H}_i$. But to see why $\mathcal{H}_i$ is defined that way, we have to think about long connector paths. Each edge in the graph $\mathcal{H}_i$ shall represent a long connector path in $\mathcal{G}$. We include a node in $\mathcal{H}_i$ if and only if there exists a long connector path of class i that has this node as internal node. But there exists no long connector path that contains z.

3.4.2 Algorithm for Building $\mathcal{H}_i$

The previous section contains a definition of the graph $\mathcal{H}_i$. But to use this graph in our fractional CDS packing algorithm, we need to give an algorithm for building it. This is exactly what we do in the following.

The type-2 new nodes initiate the creation of the graph $\mathcal{H}_i$. All type-2 nodes already know the classes and components their old neighbors belong to. Hence, each type-2 node can determine whether it shall simulate a node in $\mathcal{H}_i$. Note that for each class i, a type-2 new node lies on connector paths of at most one component of $\mathcal{G}[\mathcal{V}_\ell^i]$ (cf. Lemma 3.3). Therefore, it only simulates at most one node in $\mathcal{H}_i$. We will need this fact later when we analyze the runtime of the maximal matching algorithm. If a node v adds a copy of itself to $\mathcal{H}_i$ it sends the ID of its neighboring component to all type-1 new neighbors. These check whether they satisfy the conditions to join the graph $\mathcal{H}_i$. This can be done locally as they already have the information which classes and components their neighbors belong to. If the result is positive they answer with a special PATH-FOUND symbol. This completes the creation of $\mathcal{H}_i$.

3.4.3 Finding Matchings in $\mathcal{H}_i$

Our goal is to find a set of internally vertex-disjoint connector paths for each active component. Since each edge in the graph $\mathcal{H}_i$ represents a long connector path, we can run a matching algorithm to find internally vertex-disjoint connector paths. In this section we describe how we do this.

We run the maximal matching algorithm for bipartite graph that was proposed in [CHGK14a], which adapts Luby's algorithm for finding maximal independent sets [Lub86]. In our case the implementation is even simpler because each virtual node in $\mathcal{H}_i$ is simulated by one real node, whereas in the algorithm of [CHGK14a] there are virtual nodes that represent connected components. We do not need additional communication to ensure that all nodes in one component agree on their behavior in the next stage of the algorithm.

The variant we use has $O(\log n)$ stages, which work as follows: Each type-2 node $v_\mathcal{C} \in \mathcal{H}_i$ that is still active randomly picks a number between 1 and n^{16} for each active edge that has $v_\mathcal{C}$ as its endpoint. Therefore, we can say each edge has a random number assigned to it, which we will use as its priority. The node $v_\mathcal{C}$ then picks the edge with the highest priority and proposes to the other endpoint of this edge. This proposal includes the priority.

Some nodes of $\mathcal{H}_i$ that are simulated by a type-1 node of $\mathcal{G}$ have received proposals. Suppose $w_\mathcal{C}$ is one of them. It finds the proposal with the highest priority and accepts it. This means that $w_\mathcal{C}$ sends a message to the sender $v_\mathcal{C}$ of this proposal containing the PROPOSAL-ACCEPTED-symbol.

If a node $v_\mathcal{C}$ receives a PROPOSAL-ACCEPTED-symbol from $w_\mathcal{C}$, it includes the edge in the matching. The nodes $v_\mathcal{C}$ and $w_\mathcal{C}$ know that an adjacent edge is included in the matching. Thus, they become inactive and send the INACTIVE-symbol to all neighbors in $\mathcal{H}_i$. Upon receiving a message containing the INACTIVE-symbol, the node marks the edge to the sender as inactive and ignores this edge in the following stages.

After $O(\log n)$ stages the resulting matching is maximal w.h.p. (cf. Lemma 4.12). As all edges in the graph $\mathcal{H}_i$ stand for long connector paths of components of class i, the edges in the matching represent a subset of these paths. We call these long connector paths *active*. Due to the matching condition all active connector paths that belong to the same component are internally vertex-disjoint. Note that this does not hold for active connector paths of different components as shown in Figure 3.4.

In this section we made use of the fact that each new type-2 node lies on at most one connector path of each class. In the following section we study the properties of connector paths, which includes a proof of this fact.

3.5 Properties of Connector Paths

In this section we examine general properties of connector paths. This includes statements concerning the number of connector paths per connected component (cf. Lemma 3.2) and the number of long connector paths per type-2 new node (cf. Lemma 3.3).

Lemma 3.2. *Let i be a class and $\ell \geq L_1$ a layer. Consider any connected component $\mathcal{C}$ of $\mathcal{G}[\mathcal{V}_\ell^i]$. If $\mathcal{V}_\ell^i$ dominates $\mathcal{G}$ and $\mathcal{C}$ is not the only component of $\mathcal{G}[\mathcal{V}_\ell^i]$, the component $\mathcal{C}$ has at least k internally vertex-disjoint connector paths.*

This lemma is stated as Lemma 4.3 in [CHGK14a], and a formal proof can be found there. We want to give an intuition about how this proof works.

Let $\mathcal{C}$ be a connected component of class i that satisfies the conditions of the lemma. Additionally, let $\mathcal{C}' \neq \mathcal{C}$ be another component of the same class. Fix real nodes s and t in $C = \Psi(\mathcal{C})$ and $C' = \Psi(\mathcal{C}')$, respectively. According to Menger's Theorem there are at least k internally vertex-disjoint paths from s to t in the real graph G [BM08, Theorem 9.1]. These paths are usually not connector paths themselves but we can derive connector paths by shortening them.

In Figure 3.5 an example of such a path is shown. We can see that at some point the path P leaves the component C and enters another component $C'' \neq C$. Due to the domination of the class, we can pick one or two nodes between the components such that they form the internal nodes of a potential connector path. Setting the types of the internal nodes properly, we get a connector path of component $\mathcal{C}$ in $\mathcal{G}$.

Note that the path P may leave and enter the component C before it enters a different component of the same class, as it is shown in the top left of Figure 3.5. Moreover, the component C'' might be different from the component C' the path ends in. Also note that the connector path derived from path P may contain nodes and edges that do not belong to P itself. In our example the connector path starts at node u, which is not on P. All internal nodes of the connector path—v and w in our example—also lie on the path P

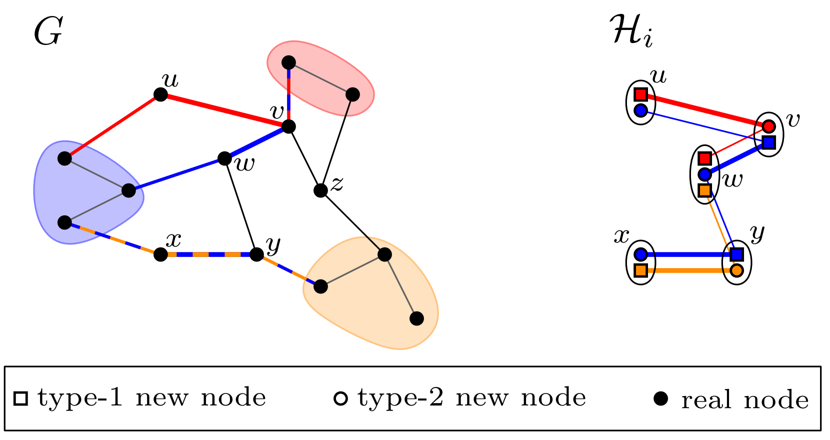

Figure 3.4: A possible result of the matching algorithm on the graph $\mathcal{H}_i$ and the projection of the active connector paths to the graph G

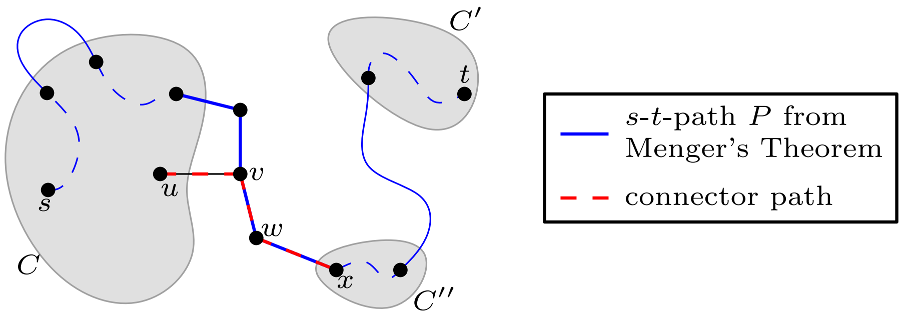

Figure 3.5: Shortening of a path to a connector path

though. Hence, no other connector path of component $\mathcal{C}$ contains them. This implies that the resulting connector paths are internally vertex-disjoint.

The following lemma examines the long connector paths belonging to the same class. It is presented in [CHGK14b] as Proposition 3.3 and in [CHGK14a] as Proposition 4.2. However, neither of these two articles contains a full proof of this statement. Therefore, we give one here.

Lemma 3.3. *Suppose we are at an upper layer and v is a type-2 new node. For each class i, there is at most one component of class i that has long connector paths with v as internal node.*

Proof. We show that if there were two such components, we could shorten the long connector paths. This would violate the minimality of the connector paths.

Let $\ell + 1$ be an upper layer and i a class. Assume there is a type-2 new node v and two components $\mathcal{C}_1$ and $\mathcal{C}_2$ of $\mathcal{G}[\mathcal{V}_\ell^i]$ such that both components have connector paths that contain the node v. Let P_1 and P_2 be such paths that belong to components $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. Moreover, we denote the endpoint of path P_j that belongs to component $\mathcal{C}_j$ with u_j. This situation is shown in Figure 3.6.

However, this implies that there is a short connector path from u_1 to u_2 via the type-1 new node that is simulated by the same real node as v. This violates the requirement that long connector paths cannot be shortened. Hence, the paths P_1 and P_2 are not long connector paths, which contradicts the assumption. Thus, all long connector paths of the same class that have the same type-2 internal node belong to the same component. $\square$

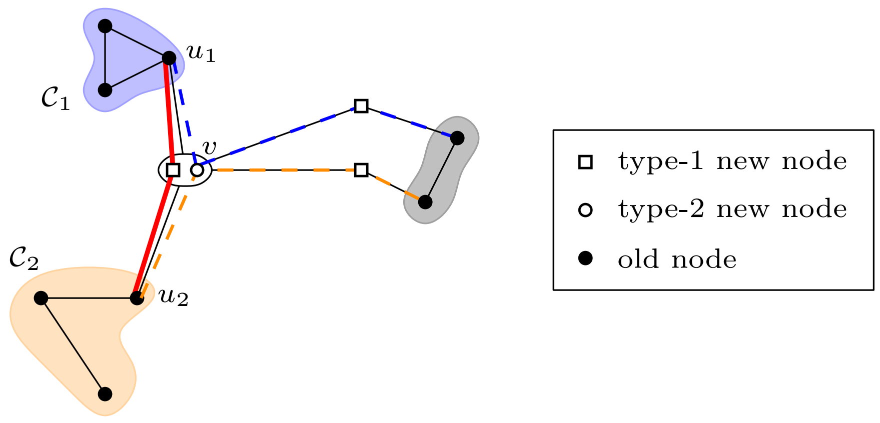

Figure 3.6: Two long connector paths of the same class that have the same internal type-2 node v. The solid red path is the short connector path that makes the two long connector paths invalid.

4. Analysis

We analyze the algorithm in this chapter. First, we determine its runtime in both the V-CONGEST and the E-CONGEST model (cf. Section 4.1). In the second part of this chapter, we show that the algorithm works correctly, i.e. it outputs a fractional CDS packing w.h.p.

4.1 Runtime

In this section we show the total runtime of the algorithm. We focus on the V-CONGEST model. Additionally, we show how the runtime can be improved if we use the less restricted E-CONGEST model.

Theorem 4.1. *Let G be a graph with n nodes, vertex-connectivity k and maximum degree Δ. The algorithm takes $O(\log^2 n(D + \sqrt{n \log n} \log^* n + k\Delta))$ rounds of communication in the V-CONGEST model.*

Proof. Clearly, we do not need any communication for the first L_1 layers. We analyze how many rounds of communication we need for each upper layer in the following.

Identify connected components of old nodes: Suppose the nodes of layers 1 to ℓ have already selected their classes. According to Theorem 3.1, the algorithm to determine the connected components of old nodes needs $O(D + \sqrt{|\mathcal{V}|} \log^* |\mathcal{V}|)$ rounds on the virtual graph $\mathcal{G}$. Since $\mathcal{G}$ has $|\mathcal{V}| = O(n \log n)$ nodes and the same diameter as G (unless G consists of only one node), the running time can be simplified to $O(D + \sqrt{n \log n} \log^* n)$ rounds on the virtual graph $\mathcal{G}$. Each real node simulates $O(\log n)$ virtual nodes so each round on $\mathcal{G}$ can be simulated in $O(\log n)$ rounds on G. Hence, we need $O(\log n(D + \sqrt{n \log n} \log^* n))$ rounds for this step.

Type-1 new nodes select class: All old nodes send their component IDs to their neighbors and all type-1 new nodes send the ID of the class they joined. Both can be done in one step on the virtual graph $\mathcal{G}$, and thus, it takes $O(\log n)$ rounds on G. Broadcasting that a component is inactive is done by another application of the algorithm of Theorem 3.1. Hence, we again need $O(\log n(D + \sqrt{n \log n} \log^* n))$ rounds.

Determine active connector paths: Each real node simulates one new node of type 1 and one of type 2. Therefore, establishing the graph $\mathcal{H}_i$ can be done in two rounds. Although a real node may simulate one type-2 node and multiple type-1 nodes in $\mathcal{H}_i$, each edge is used at most twice, which is shown in Lemma 4.3. This implies that in one round on $\mathcal{H}_i$ each real node sends at most two messages per edge. Hence, each round on $\mathcal{H}_i$ can be simulated in $O(\Delta)$ rounds on the real graph G, where Δ denotes the maximum degree.

In Lemma 4.12 it is shown that the maximal matching algorithm takes $O(\log n)$ rounds on the graph $\mathcal{H}_i$. Our argumentation above implies that this can be simulated in $O(\Delta \log n)$ rounds on the real graph G. We apply the maximal matching algorithm once for each class. Therefore, we need $O(t\Delta \log n) = O(k\Delta \log n)$ rounds to find internally vertex-disjoint connector paths for all classes.

Wrap Up

All in all, we need $O(\log n(D + \sqrt{n \log n} \log^* n + k\Delta))$ rounds per layer and thus $O(\log^2 n(D + \sqrt{n \log n} \log^* n + k\Delta))$ rounds in total, which completes the proof. □

The proof above assumes that we work in the V-CONGEST model. If we use the less restricted E-CONGEST model instead, it is possible to improve the runtime of the algorithm.

Corollary 4.2. *Let G be a graph with n nodes and vertex-connectivity k. The algorithm takes $O(\log^2 n(D + \sqrt{n \log n} \log^* n + k))$ rounds of communication in the E-CONGEST model.*

Proof. Note that the runtime in the E-CONGEST model does not depend on the maximum degree Δ of the graph. To get this result, we show that we can simulate the matching algorithm on the graph $\mathcal{H}_i$ more efficiently. Recall that each real edge simulates at most two edges in $\mathcal{H}_i$ (cf. Lemma 4.3). In the E-CONGEST model a node can send different messages along different edges at the same time, which is impossible in the V-CONGEST model. Hence, we can simulate each round on $\mathcal{H}_i$ in two rounds on G in the E-CONGEST model as opposed to $O(\Delta)$ rounds in the V-CONGEST model.

Leaving all other parts of the algorithm unchanged, we get a total runtime of $O(\log^2 n \cdot (D + \sqrt{n \log n} \log^* n + k))$ rounds in the E-CONGEST model. □

In the proofs above, we make use of the following structure of the graph $\mathcal{H}_i$, which allows to simulate this graph more efficiently.

Lemma 4.3. *Let i be a class. Consider the graph $\mathcal{H}_i$ that is built at an upper layer. Each real edge in the graph G simulates at most two edges in $\mathcal{H}_i$.*

Proof. Fix an edge e of $\mathcal{H}_i$. Since the graph $\mathcal{H}_i$ has been constructed separately for each component, the edge e belongs to a component $\mathcal{C}$ of $\mathcal{G}[\mathcal{V}_\ell^i]$. All virtual edges connect type-1 and type-2 nodes. Thus, one of the endpoints of e has to be of type 2. Let this endpoint be $v_{\mathcal{C}}$ and the endpoint of type 1 be $w_{\mathcal{C}}$. These nodes are simulated by the virtual type-2 node $v \in \mathcal{G}$ and type-1 node $w \in \mathcal{G}$, respectively. We show that there is no other edge in $\mathcal{H}_i$ whose endpoints are simulated by the same nodes in $\mathcal{G}$.

Assume that there is another edge e' in $\mathcal{H}$ which is simulated by the virtual nodes v and w. Due to the construction of $\mathcal{H}_i$, this edge e' cannot belong to the same component as e, which is $\mathcal{C}$. According to Lemma 3.3 there is at most one component of class i such that this component has long connector paths that include the type-2 new node v. We already know that $\mathcal{C}$ is this component because the existence of edge e implies a long connector path from $\mathcal{C}$ via v and w to another component of class i. Therefore, no other component of class i has a long connector path containing v, and thus, e is the only edge in $\mathcal{H}_i$ between copies of v and w. □

This completes our analysis of the runtime of the algorithm. In the remainder of the chapter, we show that the algorithm works correctly.

4.2 Correctness

In this section we deal with the correctness of the algorithm, which is formulated in the following theorem.

Theorem 4.4. *The classes form a fractional connected dominating set packing w.h.p.*

Proof. We need to show that the following conditions hold w.h.p.:

1. Each class dominates all nodes (cf. Theorem 4.5).
2. Each class is connected (cf. Theorem 4.6).
3. The total weight of all classes a node belongs to is at most 1.

Condition 3 is always satisfied. Each real node v simulates $L_1 + 2L_2$ virtual nodes. Hence, it is included in at most $L_1 + 2L_2$ classes. As each class has weight $1/(L_1 + 2L_2)$, the total weight of all classes that include v is at most 1.

According to Theorem 4.5, the probability that all classes are dominating sets is at least $1 - 1/n^{c_1}$ for a constant $c_1 \geq 2$. Similarly, Theorem 4.6 says that if the classes are dominating, the probability that all classes are connected is at least $1 - 1/n^{c_2}$ for a constant $c_2 \geq 2$. Setting $c' = \min\{c_1, c_2\}$, we get the following lower bound for the probability that all conditions 1–3 are satisfied.

$$\begin{aligned}
\Pr[\text{1–3 satisfied}] &= \Pr[\text{classes dominate}] \cdot \Pr[\text{classes are connected} \mid \text{classes dominate}] \\
&\geq (1 - \frac{1}{n^{c'}}) \cdot (1 - \frac{1}{n^{c'}}) \\
&\geq 1 - \frac{2}{n^{c'}} \\
&\geq 1 - \frac{1}{n^{c'-1}}
\end{aligned}$$

The last inequality holds if $n \geq 2$. For $n = 1$, i.e. the graph consists of only one node, the result of the algorithm is always a fractional CDS packing of size 1.

The calculation shows that the classes form a fractional CDS packing w.h.p. □

The remainder of the chapter contains the proofs that the result of our algorithm also satisfies the first two properties. We begin with the proof of the dominance in the next section. The other sections starting with Section 4.2.2 deal with the connectedness of the classes.

4.2.1 Dominance

In this section we show that all classes form dominating sets w.h.p., which is formulated in the following theorem. The algorithm in [CHGK14a] uses the same technique to achieve dominance and already contains a proof that this technique works. However, many details are left out. Therefore, we present a more detailed proof of this theorem. Moreover, we simplified the proof by removing the use of a Chernoff bound.

Theorem 4.5. *The following three statements hold w.h.p.*

(a) The virtual nodes of each class that belong to the lower layers form a dominating set.

(b) All nodes of each class form a dominating set in $\mathcal{G}$.

(c) The projection of all classes to G form dominating sets.

Before we begin with the proof, let us consider where we need each part of the theorem. Part (c) states the desired result of the algorithm. Parts (a) and (b) are used to show that classes are also connected.

Proof. (a) Fix a class i. We show that the lower virtual nodes of this class form a dominating set. Due to the construction of the virtual graph, all virtual nodes that are projected to the same real node have the same neighbors. Therefore, it suffices to show that the virtual nodes of layer 1 are dominated.

Let v be a virtual node in layer 1. Since G has vertex connectivity k, each real node has at least k neighbors. Therefore, v has at least $kL_1 = k\lambda \log n$ neighbors in the lower layers. All neighbors independently join class i with probability $1/t = 1/\alpha k = \beta/k$ for $\beta = 1/\alpha$.

$$\begin{aligned}
\Pr[\text{one neighbor selects class } i] &\geq 1 - \left(1 - \frac{\beta}{k}\right)^{k\lambda \log n} \\
&= 1 - \left(\left(1 - \frac{\beta}{k}\right)^{k}\right)^{\lambda \log n} \\
&\overset{(*)}{\geq} 1 - e^{-\beta\cdot\lambda \log n} \\
&= 1 - \frac{1}{n^{\beta\lambda}}
\end{aligned}$$

The step marked with $(*)$ uses the inequality from Lemma 2.4. Using a union bound over all choices of nodes in layer 1 and all classes we obtain a lower bound for the probability that statement (a) holds.

$$\begin{aligned}
\Pr[\text{(a) holds}] &\geq 1 - \sum_{i=1}^{t} \sum_{v \in \mathcal{V}_1} \Pr[v \text{ has no lower-layer neighbor in class } i] \\
&\geq 1 - \sum_{i=1}^{t} \sum_{v \in \mathcal{V}_1} \frac{1}{n^{\beta\lambda}} \\
&= 1 - \frac{nt}{n^{\beta\lambda}} \\
&\geq 1 - \frac{1}{n^{\beta\lambda - 2}}
\end{aligned}$$

Hence, the nodes of class i in the lower layers dominate the virtual graph w.h.p., which completes the proof of part (a).

(b) Clearly, if the nodes of a class in the lower levels already dominate $\mathcal{G}$, this is also true if we additionally consider the nodes of this class in the upper layers. Thus, all virtual nodes of a class dominate $\mathcal{G}$ w.h.p.

(c) Due to the construction of the virtual graph $\mathcal{G}$, the projection of a dominating set in $\mathcal{G}$ is a dominating set in G. Hence, the resulting sets of real nodes dominate G w.h.p. □

4.2.2 Connectedness

In the previous section, we have seen that the results of the algorithm are indeed dominating sets. We now turn to the second part of the algorithm and show that these sets are also connected.

Theorem 4.6. *W.h.p., the nodes of all classes are connected after L layers, i.e. $\mathcal{G}[\mathcal{V}_L^i]$ is connected for $i \in \{1, \ldots, t\}$.*

Proof. Let X_ℓ^i be the number of connected components of $\mathcal{G}[\mathcal{V}_\ell^i]$ and $M_\ell^i = X_\ell^i - 1$ the number of "missing connections" between these components. We denote the sum of missing connections over all classes with $M_\ell = \sum_{i=1}^t M_\ell^i$. Hence, we have that $M_\ell = \sum_{i=1}^t X_\ell^i - t$. Our goal is to show that after L layers the nodes of all classes are connected, which means $M_L = 0$.

In Lemma 4.7 we show that the number missing components decreases in each layer by a constant factor with probability at least π. Using this lemma, we are able to prove that $M_L = 0$ w.h.p. We do this in three steps: First, we determine an upper bound for the value of M_{L_1}. In a second step we calculate the expected value of M_L. Finally, we derive the desired bound for the probability by an application of Markov's inequality.

To obtain an upper bound for the number of missing connections we note that each class has less that n components. Therefore, we have that tn and thus n^2 are upper bounds for M_{L_1}.

Since the values of M_ℓ are nonnegative integers, $M_L = 0$ is equivalent to $M_L < 1$. We claim that setting $L_2 > -(2+c)\log n / \log(1-\delta\pi)$ suffices to get a probability of at least $1 - 1/n^c$ for the event that all classes are connected. Note that $\log(1-\delta\pi) < 0$. Therefore, L_2 is always positive.

In order to calculate $\mathbb{E}[M_L]$, we analyze the relation between the expected values of two succeeding layers $\mathbb{E}[M_\ell]$ and $\mathbb{E}[M_{\ell+1}]$. According to Lemma 4.7 we have that $M_{\ell+1} \leq (1-\delta) \cdot M_\ell$ with probability at least π. If the value of $M_{\ell+1}$ does not decrease by this constant factor $1-\delta$, we still know that $M_{\ell+1} \leq M_\ell$. Hence, we obtain the following inequality for the expected value of $M_{\ell+1}$:

$$\mathbb{E}[M_{\ell+1}] \leq (\pi(1-\delta) + (1-\pi)\cdot 1) \cdot \mathbb{E}[M_\ell] = (1-\delta\pi)\cdot \mathbb{E}[M_\ell]$$

Using this inequality L_2 times, once for each layer $\ell \in \{L_1, \dots, L-1\}$, we get an upper bound for the expected value of $\mathbb{E}[M_L]$.

$$\begin{aligned}
\mathbb{E}[M_L] &\leq (1-\delta\pi)^{L_2} \cdot \mathbb{E}[M_{L_1}] \\
&< (1-\delta\pi)^{\frac{-(2+c)\log n}{\log(1-\delta\pi)}} \cdot n^2 \\
&= \frac{1}{n^{2+c}} \cdot n^2 \\
&= \frac{1}{n^c}
\end{aligned}$$

An application of Markov's inequality gives an upper bound for the probability that not all classes are connected, i.e. $M_L \geq 1$.

$$\Pr[M_L \geq 1] \leq \frac{\mathbb{E}[M_L]}{1} \leq \frac{1}{n^c}$$

Hence, we have $\Pr[M_L = 0] = \Pr[M_L < 1] \geq 1 - 1/n^c$, which means that all classes are connected w.h.p. □

The following lemma is our variant of what Censor-Hillel, Ghaffari and Kuhn call the "Fast Merger Lemma" [CHGK14a, Lemma 4.4]. It is the key to the proof of the connectedness. Its proof is based on the proof of the Fast Merger Lemma of Censor-Hillel, Ghaffari, and Kuhn. That proof however contains some minor technical flaws. Therefore, we present a corrected version of this proof as part (b), which is also slightly adapted to fit to our algorithm.

The main flaw in the proof of the Fast Merger Lemma in [CHGK14a] is that they seem to use the variable M_ℓ for two different things: Sometimes, it denotes the number of missing connections, which is the correct usage according to the definition. But in some equations it is used to express the total number of connected components, which is actually $M_\ell + t$. Despite the flaws in the proof, the statement of the Fast Merger Lemma itself is correct.

In our corrected version we always refer to the total number of connected components as $M_\ell + t$ and additionally account for the fact that there are components that are single in its class.

Although the proof of part (a) can be found in the proof of the Fast Merger Lemma in [CHGK14a], we include it here to be self-contained.

Lemma 4.7 (Fast Merger Lemma)**.** *Provided that all classes form dominating sets, the following holds for each layer $\ell \in \{L_1, \dots, L-1\}$:*

(a) $M_{\ell+1} \leq M_\ell$*,*

(b) For sufficiently small constants δ and π we have $\Pr[M_{\ell+1} \leq (1-\delta)M_\ell] \geq \pi$ *with independence between the layers.*

Proof (based on the proof of Lemma 4.4 in [CHGK14a])*.* Let $\ell \in \{L_1, \dots, L-1\}$ be a layer. Recall that the virtual nodes in layer 1 to ℓ are called old nodes and the nodes in layer $\ell+1$ are new nodes. Suppose a new node v joins class i. Since the nodes of class i form a dominating set, v has at least one neighbor in class i. Hence, adding v to class i does not create a new connected component. Therefore, the number of components cannot increase, which implies part (a).

For part (b), let i be a class such that $\mathcal{G}[\mathcal{V}_\ell^i]$ has at least two components. Suppose $\mathcal{C}$ is one of these. We call the component $\mathcal{C}$ *good* if at least one of the following conditions holds:

(A) There is a type-1 new node u such that it is adjacent to a node in $\mathcal{C}$ and another node in $\mathcal{V}_\ell^i \setminus \mathcal{C}$, and u joins class i.

(B) There are two adjacent nodes w and v with types 1 and 2, respectively, such that v has a neighbor in $\mathcal{C}$ and w has a neighbor in $\mathcal{V}_\ell^i \setminus \mathcal{C}$, and both nodes join class i.

Otherwise, we call the component $\mathcal{C}$ *bad*. We can see that a good component of old nodes is merged with another component of the same class at the next layer. Note that the converse does not hold, i.e. there might be components that are merged although they are bad. This might occur because condition (b) is not symmetric. Nevertheless, we can use the number of good components as a lower bound for the number of components that are merged at the next layer. If a component is the only component of its class we say that this component is *single*.

We first show that there are at least $3\delta \cdot M_\ell$ good components in expectation for a constant $\delta > 0$. An application of Markov's inequality then shows that with constant probability there are at least $2\delta \cdot M_\ell$ good components, which implies that the number of missing connections decreases by a constant factor, i.e. $M_{\ell+1} \leq (1-\delta)M_\ell$ with constant probability.

We denote the number of single components with Z_ℓ. Let Y_ℓ be the total number of bad components that are not single in its class and X_ℓ be the number of good components.

Since each component $\mathcal{C}$ is either good, bad, or single, $\mathcal{C}$ is counted exactly once. Therefore, the sum $X_\ell + Y_\ell + Z_\ell$ is the total number of components of old nodes, which can also be expressed as $M_\ell + t$. Hence, we have that $Y_\ell = M_\ell - X_\ell - Z_\ell + t$.

Each good component is merged with at least one other component. Hence, a group of at least two components is merged to one component in the next layer. Therefore, the total number of components decreases by at least $X_\ell/2$.

Therefore, we have for the total number of components at level $\ell+1$ that $M_{\ell+1} + t \leq X_\ell/2 + Y_\ell + Z_\ell$. Using the fact that $Y_\ell = M_\ell - X_\ell - Z_\ell + t$ we can simplify this to $M_{\ell+1} \leq M_\ell - X_\ell/2$. Intuitively, this means that the good components decrease the number of missing connections.

Suppose that $M_\ell = m$ and $Z_\ell = z$. Fixing these two values follows the structure of our algorithm since they do not depend on what happens to the nodes in the layer $\ell+1$. Intuitively, we consider the situation when all nodes in the old layers have selected a class, and hence, the values of M_ℓ and Z_ℓ are fixed.

Assume we know that $\mathbb{E}[X_\ell \mid M_\ell = m, Z_\ell = z] \geq 3\delta \cdot (M_\ell + t - Z_\ell)$ for some constant $\delta > 0$, which is independent of the values of m and z. Since $X_\ell + Y_\ell = M_\ell - Z_\ell + t$, we have that in this case the expected value of the number of bad components is $\mathbb{E}[Y_\ell \mid M_\ell = m, Z_\ell = z] \leq (1-3\delta)(M_\ell + t - Z_\ell)$. Applying Markov's inequality gives an upper bound for the probability that there are "many" bad components.

$$\Pr[Y_\ell \geq (1-2\delta)(M_\ell + t - Z_\ell) \mid M_\ell = m, Z_\ell = z] \leq \frac{\mathbb{E}[Y_\ell \mid M_\ell = m, Z_\ell = z]}{(1-2\delta)(M_\ell + t - Z_\ell)} \leq \frac{1-3\delta}{1-2\delta}$$

Having $Y_\ell \geq (1-2\delta)(M_\ell + t - Z_\ell)$ is equivalent to the event $X_\ell \leq 2\delta(M_\ell + t - Z_\ell)$. Hence, we can derive the following lower bound for the probability that there are enough good components:

$$\Pr[X_\ell \geq 2\delta(M_\ell + t - Z_\ell) \mid M_\ell = m, Z_\ell = z] \geq 1 - \frac{1-3\delta}{1-2\delta} =: \pi$$

As this probability holds for all possible values of m and z, it follows that $\Pr[X_\ell \geq 2\delta(M_\ell + t - Z_\ell)] \geq \pi$. Since there is at most one single component per class, the total number of single components Z_ℓ is at most t. Hence, we have $t - Z_\ell \geq 0$ and thus

$$\Pr[X_\ell \geq 2\delta \cdot M_\ell] \geq \Pr[X_\ell \geq 2\delta(M_\ell + t - Z_\ell)] \geq \pi.$$

Together with $M_{\ell+1} \leq M_\ell - X_\ell/2$, this implies that $\Pr[M_{\ell+1} \leq (1-\delta)M_\ell] \geq \pi$, which would complete the proof.

It remains to show that $\mathbb{E}[X_\ell \mid M_\ell = m, Z_\ell = z] \geq 3\delta \cdot (M_\ell + t - Z_\ell)$ for all possible values of m and z. Fix $M_\ell = m$ and $Z_\ell = z$. We divide the components that are not single in two groups. The first group contains those components that have at least $k/2$ short connector paths. We call these components *fast*. The remaining components go into the group of *slow* components. Since we know from Lemma 3.2 that all components that are not single have at least k vertex-disjoint connector paths, all slow components have at least $k/2$ vertex-disjoint long connector paths.

We show that both slow and fast components become good with at least constant probability. Each component has two possibilities to become good: Either the internal node of a short connector path or the two internal nodes of a long connector path join the right class. In the algorithm we first check whether the first condition holds before we try to find long connector paths. The analysis below follows this structure.

Let $\mathcal{C}$ be a fast component of class i. Since $\mathcal{C}$ has $k' \geq k/2$ short connector paths and $t = \alpha k$, we have $k' \geq \gamma t$ for $\gamma = 1/2\alpha$. Each internal node of a short connector path has probability $1/t$ to join class i. An application of Lemma 2.4 shows that the probability that no internal node of these paths joins class i is constant:

$$\Pr[\text{no path selects class } i] = \left(1 - \frac{1}{t}\right)^{k'} \leq \left(1 - \frac{1}{t}\right)^{\gamma t} \leq e^{-\gamma}$$

Thus, the probability that at least one internal node of these paths selects class i is at least $\delta_F = 1 - e^{-\gamma}$. This already shows that the probability that $\mathcal{C}$ becomes good is at least δ_F. Therefore, we do not need to consider the long connector paths.

Now suppose $\mathcal{C}$ is a slow component of class i. There is a positive probability ρ that $\mathcal{C}$ becomes good because a short connector path selects class i. Suppose this does not happen. In Lemma 4.11 we show that in this case there is a constant probability $\delta_S > 0$ that at least one long connector path selects class i, and thus, the component $\mathcal{C}$ becomes good. Hence, the probability that $\mathcal{C}$ becomes good is at least $\rho + (1-\rho)\delta_S \geq \delta_S$.

To combine these two results we select a constant $\delta > 0$ such that $3\delta \leq \min\{\delta_F, \delta_S\}$. Hence, for each component $\mathcal{C}$ that is not single the probability that $\mathcal{C}$ becomes good is at least 3δ. Note that the value of δ is independent of the values of $M_\ell = m$ and $Z_\ell = z$. There are $M_\ell + t - Z_\ell$ components that are not single. Thus, using the linearity of expectation, we obtain that $\mathbb{E}[X_\ell \mid M_\ell = m, Z_\ell = z] \geq 3\delta \cdot (M_\ell + t - Z_\ell)$, which completes the proof. □

We assumed that the active, slow components become good with constant probability. In the following section, we show that this assumption is true.

4.2.3 Connecting Slow Components

In order to make the proof cleaner we replace the strategy of how a type-2 node chooses a class by a different and slightly more complicated one. To distinguish the two strategies we call the one used in the algorithm *strategy A* and the modified one *strategy B*.

Definition 4.8 (Selection Strategy B)**.** *First, each type-2 node v discards all paths whose type-1 internal node has not chosen the right class. Then, all paths are independently discarded with probability* $1/2$. *We call this step the* random discard step. *The node v randomly selects one of the remaining paths and assigns the class of the path to itself. If all paths are discarded, v joins a random class.*

The random discard step is the only difference between the two strategies, i.e. if we remove this step we get strategy A, which we use in the algorithm. In order to show a constant lower bound for the probability that active, slow components become good, we take the following steps:

1. We first show that if we use strategy B, the probability that an active long connector path selects class i is at least $1/4t$ (cf. Lemma 4.9).
2. In Lemma 4.10 we compare the two strategies and show that the probability that a path selects class i is higher if we use strategy A. This implies that the lower bound of $1/4t$ holds for strategy A as well. Additionally, we prove an upper bound of $1/t$.
3. Finally, we combine these bounds to obtain the desired result. This is shown in Lemma 4.11.

Lemma 4.9. *Let P be a long connector path of an active component of class i. The probability that both internal nodes of P join class i is at least* $1/4t$ *if we use strategy B.*

Proof. This proof follows the structure of the first part of the proof of Lemma 4.5 in [CHGK14a]. We also use a result that is shown there, which we present in Lemma A.1.

Let $\mathcal{C}$ be a component of class i that is active after the type-1 new nodes have chosen their classes. Fix a long connector path P of $\mathcal{C}$ and let v and w be the type-2 and type-1 new internal nodes of P. We show that the probability that P selects class i, and hence, connects $\mathcal{C}$ to another component of class i, is at least $1/4t$.

In order to simplify the proof of this lower bound we only focus on the case in which P, the long connector path of component $\mathcal{C}$, is the only useful path that goes through v and thus v joins class i. This is the case if and only if the following two conditions hold:

(A) The type-1 node w selects class i and is not discarded.

(B) All other active connector paths through v that do not lead through w are discarded.

These two conditions are independent. Note that in condition (B) we exclude all paths with v and w on them as (A) already implies that they are discarded. If we included them, the two conditions above would not be independent anymore.

For (A) to hold, the node w must join class i and the path P is not discarded. Since w is of type 1, it randomly selects one class. Hence, the probability that w joins class i is $1/t$. The node v discards the path P with probability $1/2$. As these two events are independent, the probability that condition (A) is met is $1/2t$.

In order to calculate a lower bound for the probability that condition (B) holds, we need to look at the active connector paths that contain v.

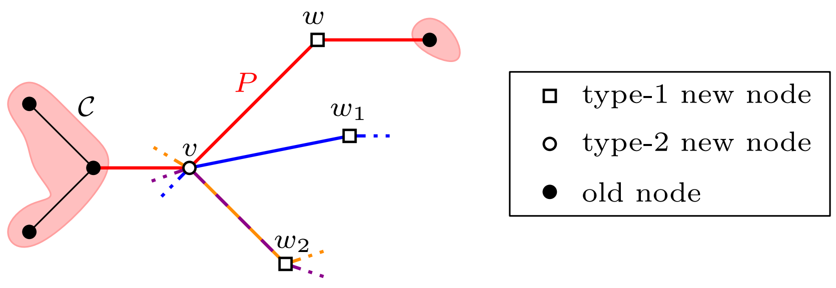

Figure 4.1: Active connector paths of different classes that have the same internal type-2 node.

Let $i_1, \dots, i_z$ be the classes that have active long connector paths going through v but not through w. Clearly, we have $z < t$. Moreover, let $w_1, \dots, w_{z'}$ be the type-1 new nodes on these connector paths. Since multiple connector paths might share the same type-1 node we have $z' \leq z$. For each node w_j, we denote the number of connector paths that go through v and w_j with x_j. This situation is shown in Figure 4.1. In this example, there is one path that contains the node w_1 and thus we have $x_1 = 1$. Similarly, we get $x_2 = 2$ because there are two connector paths going though w_2.

According to Lemma 3.3, there is at most one component of each class with a connector path that contains the type-2 node v. Since the active connector paths of each component are internally vertex-disjoint, there is at most one active connector path containing v per class and thus at most t in total. This implies $\sum_{j=1}^{z'} x_j < t$. Note that this sum is strictly less than the number of classes because the path of class i is not counted.

Fix one type-1 node w_j. The probability that all paths through w_j are discarded can be determined by looking at two distinct cases: (a) The node chooses a class different from the classes of the paths that go through it or (b) the node chooses a class of a path but this path is discarded by v in the random discard step. There are paths of x_j classes that have a long connector path containing v and w_j. Thus, the probability that the node w_j chooses none of these classes is $\Pr[\text{(a) holds}] = 1 - x_j/t$. Otherwise, if it selects one of these classes, the probability that v discards the path of this class is $1/2$. Hence, (b) holds with probability $x_j/t \cdot 1/2$. We have

$$\begin{aligned}\Pr[\text{all paths through } w_j \text{ are discarded}] &= \Pr[\text{(a) holds}] + \Pr[\text{(b) holds}] \\ &= \left(1 - \frac{x_j}{t}\right) + \frac{x_j}{t} \cdot \frac{1}{2} \\ &= 1 - \frac{x_j}{2t}\end{aligned}$$

The events for all type-1 nodes $w_1, \dots, w_{z'}$ are independent. Thus, Lemma A.1 gives a lower bound of $1/2$ for the probability of condition (B). As the two conditions (A) and (B) are independent, the probability that both are satisfied is at least $1/2 \cdot 1/2t = 1/4t$.

We focused on the case that the long connector path P of class i is the only one that is not discarded. Of course, there are other cases in which the type-2 node v might also join class i. But these cases only increase the probability that both internal nodes v and w of path P join class i. Therefore, $1/4t$ is a lower bound of the probability that this happens, which completes the proof. □

We have seen that the probability that both internal nodes of an active long connector path join the right class, i.e. the class the path belongs to, is at least $1/4t$ if we use strategy B. But in the algorithm we use a different strategy, which we call strategy A. We need to show that the same lower bound of the probability also holds if we use this strategy.

Lemma 4.10. *Let P be an active long connector path of an active component $\mathcal{C}$ of class i. If we use strategy A, the probability that both internal nodes of P join class i is in the interval $[1/4t, 1/t]$ and the upper bound of $1/t$ holds independently of what happens to the other active paths of component $\mathcal{C}$.*

Proof. We compare the two strategies and show that the probability is higher if we use strategy A instead of strategy B. Since we already know that the lower bound of $1/4t$ holds for strategy B (cf. Lemma 4.9), the claim follows.

Let $\mathcal{C}$ be a component of class i. Fix a long connector path P that was determined with the maximal matching algorithm. Let w and v be the internal nodes of the path P with types 1 and 2, respectively.

If w does not choose class i, the path P is useless and both strategies discard it. The probability that both internal nodes of P join class i is clearly 0 in both cases.

We now assume that the path P is useful and there are exactly $0 \leq s < t$ other useful paths. We denote the events that v selects class i in this case with V_A^s and V_B^s for the two strategies, respectively.

Strategy A randomly selects one of the $s+1$ useful paths. Hence, the probability that the path P is chosen is $\Pr[V_A^s] = 1/(s+1)$.

We now turn to strategy B and show that the probability for V_B^s is slightly less than $1/(s+1)$. Since the situation is the same for all remaining useful paths, all of them are equally likely to be chosen. Thus, the probability that P is chosen can be at most $1/(s+1)$. It is possible however that all paths are discarded and therefore no path is selected. Hence, the probability is even less than $1/(s+1)$.

To formally obtain the probability V_B^s, we first consider the case that exactly j out of the other s paths (without P) are not discarded. In this case $j+1$ paths remain, and hence, the probability that path P is chosen is $1/(j+1)$. As all paths are discarded with probability $1/2$ independently of each other, the number of remaining paths follows a binomial distribution. Thus, we have

$$\begin{aligned}
\Pr[P \text{ chosen and } j \text{ other paths}] &= \Pr[P \text{ chosen} \mid j \text{ other paths}] \cdot \Pr[j \text{ other paths}] \\
&= \frac{1}{j+1} \cdot \binom{s}{j} \left(\frac{1}{2}\right)^j \left(\frac{1}{2}\right)^{s-j} \\
&= \frac{1}{j+1} \cdot \binom{s}{j} \left(\frac{1}{2}\right)^s
\end{aligned}$$

Summing over all possibilities for the value of j, we get the probability that path P is selected if it is not discarded. Hence, we get the following probability for the event that P is chosen, which is V_B^s.

$$\begin{aligned}
\Pr[V_B^s] &= \Pr[P \text{ not discarded}] \cdot \Pr[V_B^s \mid P \text{ not discarded}] \\
&= \frac{1}{2} \cdot \sum_{j=0}^{s} \frac{1}{j+1} \binom{s}{j} \left(\frac{1}{2}\right)^s \\
&= \frac{1}{2^{s+1}} \sum_{j=0}^{s} \frac{1}{j+1} \cdot \frac{s!}{(s-j)!j!} \\
&= \frac{1}{2^{s+1}} \sum_{j=0}^{s} \frac{1}{s+1} \cdot \frac{(s+1)!}{(s+1-j-1)!(j+1)!} \\
&= \frac{1}{(s+1)2^{s+1}} \sum_{j=0}^{s} \binom{s+1}{j+1}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{(s+1)2^{s+1}}\left(2^{s+1}-1\right) = \frac{1}{s+1}\left(1-\frac{1}{2^{s+1}}\right) \\
&< \frac{1}{s+1} = \Pr\left[V_A^s\right]
\end{aligned}
$$

We compared the two strategies in all possibles cases. Combining these results leads to the desired lower bound of $1/4t$ for the probability that both internal nodes of path P join class i when we use strategy A. We denote this event with X_A and the corresponding event for strategy B with X_B. Furthermore, let W be a random variable that describes the class the type-1 node w has selected. Note that we do not have to distinguish between the two strategies here because in both strategies w picks its class in the same way. Let S be another random variable that describes the number of useful paths other than P that contain the type-2 node v. We show that $\Pr[X_A] \geq \Pr[X_B]$.

$$
\begin{aligned}
\Pr[X_A] &= \Pr[W=i] \cdot \left(\sum_{s=0}^{t-1} \Pr[S=s] \cdot \Pr[V_A^s]\right) \\
&\geq \Pr[W=i] \cdot \left(\sum_{s=0}^{t-1} \Pr[S=s] \cdot \Pr[V_B^s]\right) \\
&= \Pr[X_B] \\
&\geq \frac{1}{4t}
\end{aligned}
$$

Thus, we have shown the lower bound of $1/4t$ for the probability of the event X_A.

For the upper bound of $1/t$ for X_A we note that the type-1 internal node of the long connector path P joins class i with probability $1/t$. Hence, the probability for the event that both internal nodes of P join class i is at most $1/t$. Since all connector paths of component $\mathcal{C}$ are internally vertex-disjoint, this upper bound holds independently of what happens with the other connector paths of component $\mathcal{C}$. □

The previous lemma considers one fixed active connector path of a connected component $\mathcal{C}$ of class i. The component $\mathcal{C}$ becomes good if any of its active connector paths selects class i. To get a lower bound for the probability that this happens, we must consider all connector paths of the connected component $\mathcal{C}$.

Lemma 4.11. *Suppose that the nodes of class i dominate. Let $\ell+1$ be an upper layer and $\mathcal{C}$ a component of class i that is still active after the type-1 new nodes have selected a class. If $\mathcal{C}$ has $k/2$ internally vertex-disjoint long connector paths, the probability that for any of these paths both internal nodes join class i is at least $\delta = 1/40 - 1/2^{11}$.*

Proof. This proof is based on the last part of the proof of Lemma 4.5 in [CHGK14a].

According to Lemma 4.12 the matching we have calculated for $\mathcal{H}_i$ is maximal with probability at least $1 - 1/|\mathcal{H}_i|^c$ for a constant $c \geq 1$. Since the component $\mathcal{C}$ is not single and we assume that nodes in $\mathcal{V}_\ell^i$ dominate the graph $\mathcal{G}$, there exists a connector path belonging to component i. Hence, the graph $\mathcal{H}_i$ contains at least one edge and thus at least two nodes, i.e. $|\mathcal{H}| \geq 2$. This implies that the matching in $\mathcal{H}_i$ is maximal with probability at least $1 - 1/2^1 = 1/2$. For now, we assume that the matching is maximal.

We constructed $\mathcal{H}_i$ as the disjunct union of bipartite graphs, one for each connected component of $\mathcal{G}[\mathcal{V}_\ell^i]$. Let $\mathcal{H}_\mathcal{C}$ be the component of $\mathcal{H}_i$ that belongs to the component $\mathcal{C}$. The maximal matching in $\mathcal{H}_i$ induces a maximal matching in the subgraph $\mathcal{H}_\mathcal{C}$.

There are at least $k/2$ internally vertex-disjoint, long connector paths belonging to component $\mathcal{C}$. Note that we do not know them. But we can use this fact to derive a lower bound of the number of active connector paths of component $\mathcal{C}$. The existence of these connector paths implies that there is a matching of size at least $k/2$ in the graph $\mathcal{H}_\mathcal{C}$. Since the size of a maximal matching is at least half of the size of a maximum matching, we know that the number of active connector paths belonging to $\mathcal{C}$ is at least $k/4$.

Let Z be a random variable which counts the number of active connector paths that belong to component $\mathcal{C}$ and select class i. Note that the events that two different paths select class i are not independent. Hence, we cannot use a Chernoff bound, and Markov's inequality is not strong enough. Censor-Hillel, Ghaffari, and Kuhn faced almost the same problem and developed a sufficiently strong lower bound for $\Pr[Z \geq 1]$, which is presented in the last part of the proof of Lemma 4.5 in [CHGK14a]. We present it including the small modifications in Lemma A.2. Applying this lemma to our situation gives $\Pr[Z \geq 1 \mid \text{matching is maximal}] \geq 2\delta$ for $\delta = 1/40 - 1/2^{11}$.

Remember that the computation above is only valid if the matching is maximal. Otherwise, there might be not enough active connector paths. We denote the event that the matching is maximal with X. We have already determined that this happens with probability $\Pr[X] \geq 1/2$. Therefore, we have

$$\begin{aligned}
\Pr[Z \geq 1] &= \Pr[X] \cdot \Pr[Z \geq 1 \mid X] + \Pr[\overline{X}] \cdot \Pr[Z \geq 1 \mid \overline{X}] \\
&\geq \Pr[X] \cdot \Pr[Z \geq 1 \mid X] \\
&\geq \frac{1}{2} \cdot 2\delta = \delta
\end{aligned}$$

Hence, the probability that both internal nodes of any active long connector path of component $\mathcal{C}$ join class i is at least $\delta = 1/40 - 1/2^{11}$, which completes the proof.

□

We showed that the slow components, i.e. those with less than $k/2$ long connector paths, become good with constant probability, which is used in the proof of Lemma 4.7. We still need to prove that the maximal matching algorithm works correctly, which is done in the next section.

4.2.4 Correctness of the Matching Algorithm

In this section we show that the matching algorithm presented in Section 3.4.3 produces a maximal matching w.h.p.

Our maximal matching algorithm can be seen as an application of Luby's maximal independent set algorithm [Lub86, Monte Carlo Algorithm A] to the line graph of the graph $\mathcal{H}_i$. To prove the correctness of our maximal matching algorithm, we compare it to Luby's maximal independent set algorithm. In order to do this, we give a short description of the latter.

Suppose we work on a graph $G = (V, E)$ with n nodes. Luby's algorithm runs for $O(\log n)$ stages, each consisting of the following steps: Each active node picks a random number chosen from $\{1, \dots, n^4\}$. Then, the values of all adjacent nodes are compared and the nodes with the lower numbers are discarded. The remaining nodes are those that have chosen a higher number than all of their neighbors. These nodes are added to the independent set. Additionally, they and all their neighbors become inactive and do not participate in the remaining stages.

Clearly, this algorithm produces an independent set. Moreover, Luby has proven that w.h.p. the independent set is maximal after $O(\log n)$ stages. We use this result to show the correctness of the maximal matching algorithm.

Lemma 4.12. *Let i be a class and $\mathcal{H}_i$ be the bipartite graph that is built at an upper layer. Running the matching algorithm on $\mathcal{H}_i$ for $O(\log n)$ stages results in a maximal matching w.h.p.*

Proof. To prove this lemma, we use the correspondence of independent sets and matchings, which is shown in Lemma 2.1. Luby has shown that w.h.p. the independent set algorithm has produced a maximal independent set after $O(\log n)$ stages. We use this result to show that the matching algorithm outputs a maximal matching w.h.p.

Our matching algorithm consists of $O(\log n)$ stages, which simulate the stages of Luby's independent set algorithm. Each real node simulates up to one node per connected component of $\mathcal{G}[\mathcal{V}_\ell^i]$. The number of connected components of one class is clearly bounded by the total number of nodes, which is n. Hence the graph $\mathcal{H}_i$ has less than n^2 nodes and therefore at most n^4 edges. This implies that it is sufficient if each edge picks a priority between 1 and $(n^4)^4 = n^{16}$. However, all computation is done by nodes and thus, the selection of the values must be done by nodes. Therefore, the type-2 new nodes perform this task and choose a priority for all adjacent edges in $\mathcal{H}_i$.

In the next step, the priorities of adjacent edges are compared. First, all type-2 nodes pick the largest priority of their adjacent edges and propose this edge to the type-1 endpoint of this edge. The type-1 nodes compare the priorities of all received proposals and accept the one with the highest priority. This is almost equivalent to the comparison step in Luby's algorithm on the line graph of $\mathcal{H}_i$. Note that an edge e might be included in the matching although it does not have the highest priority of all adjacent edges. This might happen if the edge with the higher priority is not proposed to the type-2 endpoint of e. An example is shown in Figure 4.2. The edge e is included in the matching although the edge e' has a higher priority. But e' is not proposed because the edge on the bottom has a higher priority.

However, this is not a problem. Clearly, the result of our matching algorithm is still a matching. If we additionally include an edge in the matching without violating the matching condition there are less remaining edges after this stage. Hence, the matching is maximal even faster. Therefore, the running time of $O(\log |L(\mathcal{H}_i)|) = O(\log n)$ stages of Luby's algorithm implies that after $O(\log n)$ stages the matching is maximal w.h.p. □

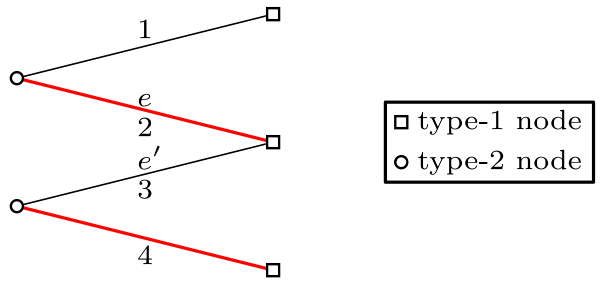

Figure 4.2: Example of graph $\mathcal{H}_i$, where an edge is selected although a neighboring edge has higher priority. The numbers at the edges denote the priorities and the marked edges are proposed and included in the matching.

5. Conclusion

In this thesis we present a distributed algorithm for finding a fractional connected dominating set packing. Given a graph with n nodes, vertex-connectivity k and maximum degree Δ, our algorithm finds a fractional CDS packing of size $\Omega(k/\log n)$. It takes $O(\log^2 n(D + \sqrt{n \log n}\log^* n + k\Delta))$ rounds of communication in the V-CONGEST model. Moreover, we describe how the number of rounds can be decreased to $O(\log^2 n(D + \sqrt{n \log n}\log^* n + k))$ rounds if we use the less restricted E-CONGEST model.

The algorithm is based on the fractional dominating tree algorithm of Censor-Hillel et al. [CHGK14a]. It improves the runtime by up to a factor of $O(\log n)$ if the vertex connectivity of the graph is not too large while the size of the resulting fractional CDS packing stays the same.

The bound for the runtime of our algorithm could be improved if one would show that each class contains $O(n \log n/k)$ nodes w.h.p. We could use this as a bound for the number of nodes per component, and thus have a bound for the diameter D' of the component. This would improve the running time of the protocol presented in Theorem 3.1 as we have a non-trivial bound for D'.

Moreover, the matching algorithm we use guarantees that its result is a maximal matching w.h.p. But we only need that this happens with constant probability. Hence, we could replace the matching algorithm with a weaker version, which might be faster. This could reduce the total runtime of the algorithm.

Bibliography

[BM08] John A. Bondy and Uppaluri S. R. Murty. *Graph theory.* Graduate texts in mathematics. Springer London, 2008.

[BYII89] Reuven Bar-Yehuda, Amos Israeli, and Alon Itai. Multiple communication in multi-hop radio networks. In *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing*, PODC '89, pages 329–338. ACM, 1989.

[CHGK14a] Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. Distributed connectivity decomposition. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, pages 156–165. ACM, 2014.

[CHGK14b] Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. A new perspective on vertex connectivity. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 546–561. SIAM, 2014.

[CK85] Imrich Chlamtac and Shay Kutten. On broadcasting in radio networks – problem analysis and protocol design. *IEEE Transactions on Communications*, 33(12):1240–1246, 1985.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT Press, 4th edition, 2009.

[GH13] Mohsen Ghaffari and Bernhard Haeupler. Fast structuring of radio networks large for multi-message communications. In *Distributed Computing*, volume 8205 of *Lecture Notes in Computer Science*, pages 492–506. Springer Berlin Heidelberg, 2013.

[GHK13] Mohsen Ghaffari, Bernhard Haeupler, and Majid Khabbazian. A bound on the throughput of radio networks. *arXiv preprint arXiv:1302.0264*, 2013.

[KK11] Majid Khabbazian and Dariusz R. Kowalski. Time-efficient randomized multiple-message broadcast in radio networks. In *Proceedings of the Thirtieth Annual ACM Symposium on Principles of Distributed Computing*, PODC '11, pages 373–380. ACM, 2011.

[Kle14] Achim Klenke. *Probability theory: a comprehensive course.* Universitext. Springer London, 2nd edition, 2014.

[KP95] Shay Kutten and David Peleg. Fast distributed construction of k-dominating sets and applications. In *Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '95, pages 238–251. ACM, 1995.

[Lub86] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[Pel00] David Peleg. *Distributed computing: a locality sensitive approach.* Society for Industrial and Applied Mathematics, 2000.

[Thu97] Ramakrishna Thurimella. Sub-linear distributed algorithms for sparse certificates and biconnected components. *Journal of Algorithms*, 23(1):160 – 179, 1997.

Appendix

The following two lemmas are taken from [CHGK14a]. As these statements are not presented explicitly but only as a part of a proof, we include them here. We use the first lemma it in the proof of Lemma 4.9.

Lemma A.1 (Part of Lemma 4.5 of [CHGK14a]). *Let $X_1, \dots, X_{z'}$ be independent events and $x_1, \dots, x_z$ natural numbers s.t. $\sum_{j=1}^{z'} x_{z'} < t$ and* $\Pr[X_j] = 1 - x_j/2t$. *Then, the probability that all events occur simultaneously is at least* $1/2$.

Proof. Clearly, we have $x_j < t$ and thus $x_j/2t \leq 1/2$. This implies

$$1 - \frac{x_j}{2t} \geq 4^{-\frac{x_j}{2t}}$$

Using this result, we can calculate a lower bound for the probability that all events occur simultaneously.

$$\Pr[X_1 \cap \dots \cap X_{z'}] = \prod_{j=1}^{z'} \left(1 - \frac{x_j}{2t}\right) \geq 4^{-\sum_{j=1}^{z'} \frac{x_j}{2t}} \geq 4^{-\frac{1}{2}} = \frac{1}{2}$$

The last inequality holds because $\sum_{j=1}^{z'} x_j < t$. □

The second lemma is used in the proof of Lemma 4.11.

Lemma A.2 (Part of Lemma 4.5 of [CHGK14a]). *Let $k' = \Omega(k)$ and $X_1, \dots, X_{k'}$ be random variables with values 0 and 1 s.t. $Pr[X_j = 1] \in [1/4t, 1/t]$ for $j = 1, \dots, k'$. Suppose that the upper bound of* $1/t$ *for* $\Pr[X_j = 1]$ *holds independently of the values of the other random variables. Then, we have*

$$\Pr\left[\sum_{j=1}^{k'} X_j \geq 1\right] \geq \frac{1}{20} - \frac{1}{2^{10}}$$

Proof. Let $Z = \sum_{j=1}^{k'} X_j$. We want to show that $\Pr[Z \geq 1] \geq 1/20 - 1/2^{10}$. As most of the proof can be found in [CHGK14a], we only give a short overview and mention where we modify it.

Originally, the probability that $X_j = 1$ lies in the interval $[1/4t, 1/2t]$ for each random variable X_j, and the upper bound holds independently of what happens with the other paths. Our situation is almost the same. The only difference is that the upper bound is $1/t$.

Using the fact that $k' = \Omega(k)$ and the linearity of expectation, they determine that $\mathbb{E}[Z] \geq z_0$ for a constant $z_0 > 1$ if the constant α in the definition of the number of classes t is sufficiently small. Then, they calculate an upper bound for $\Pr[Z = \zeta]$ for a variable ζ. In this step we have to account for the different upper bounds for the probabilities that one specific path selects class i. But applying a stronger upper bound for the binomial coefficients (cf. Lemma 2.3), we are able to derive exactly the same upper bound for $\Pr[Z = \zeta]$.

$$\Pr[Z = \zeta] \leq \binom{k'}{\zeta} \left(\frac{1}{t}\right)^{\zeta} \overset{(*)}{\leq} \left(\frac{ek'}{\zeta}\right)^{\zeta} \cdot \left(\frac{1}{t}\right)^{\zeta}$$

The $(*)$ indicates where the stronger bound for the binomial coefficient is used. Having calculated this upper bound for $\Pr[Z = \zeta]$, they derive an upper bound for $\mathbb{E}[Z]$, which depends on $\Pr[Z \geq 1]$. A quick computation leads to $\Pr[Z \geq 1] \geq 1/20 - 1/2^{10}$. $\square$