

Connected k -Partition of k -Connected Graphs and c -Claw-Free Graphs

Ralf Borndörfer  

Zuse Institute Berlin, Germany

Katrin Casel  

Hasso Plattner Institute, University of Potsdam, Germany

Davis Issac  



Hasso Plattner Institute, University of Potsdam, Germany

Aikaterini Niklanovits  

Hasso Plattner Institute, University of Potsdam, Germany

Stephan Schwartz  

Zuse Institute Berlin, Germany

Ziena Zeif  

Hasso Plattner Institute, University of Potsdam, Germany

Abstract

A *connected partition* is a partition of the vertices of a graph into sets that induce connected subgraphs. Such partitions naturally occur in many application areas such as road networks, and image processing. In these settings, it is often desirable to partition into a fixed number of parts of roughly of the same size or weight. The resulting computational problem is called Balanced Connected Partition (BCP). The two classical objectives for BCP are to maximize the weight of the smallest, or minimize the weight of the largest component. We study BCP on c -claw-free graphs, the class of graphs that do not have $K_{1,c}$ as an induced subgraph, and present efficient $(c - 1)$ -approximation algorithms for both objectives. In particular, for 3-claw-free graphs, also simply known as claw-free graphs, we obtain a 2-approximation. Due to the claw-freeness of line graphs, this also implies a 2-approximation for the edge-partition version of BCP in general graphs.

A harder connected partition problem arises from demanding a connected partition into k parts that have (possibly) heterogeneous target weights w_1, \dots, w_k . In the 1970s Györi and Lovász showed that if G is k -connected and the target weights sum to the total size of G , such a partition exists. However, to this day no polynomial algorithm to compute such partitions exists for $k > 4$. Towards finding such a partition T_1, \dots, T_k in k -connected graphs for general k , we show how to efficiently compute connected partitions that at least approximately meet the target weights, subject to the mild assumption that each w_i is greater than the weight of the heaviest vertex. In particular, we give a 3-approximation for both the lower and the upper bounded version i.e. we guarantee that each T_i has weight at least $\frac{w_i}{3}$ or that each T_i has weight most $3w_i$, respectively. Also, we present a both-side bounded version that produces a connected partition where each T_i has size at least $\frac{w_i}{3}$ and at most $\max(\{r, 3\})w_i$, where $r \geq 1$ is the ratio between the largest and smallest value in w_1, \dots, w_k . In particular for the balanced version, i.e. $w_1 = w_2 = \dots = w_k$, this gives a partition with $\frac{1}{3}w_i \leq w(T_i) \leq 3w_i$.

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis

Keywords and phrases connected partition, Györi-Lovász, balanced partition, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2021.27

Category APPROX

Related Version *Full Version*: <https://arxiv.org/abs/2107.04837>

Funding This research was partially funded by the HPI Research School on Data Science and Engineering.

Aikaterini Niklanovits: HPI Research School

Ziena Zeif: HPI Research School



© Ralf Borndörfer, Katrin Casel, Davis Issac, Aikaterini Niklanovits, Stephan Schwartz, and Ziena Zeif; licensed under Creative Commons License CC-BY 4.0

Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2021).

Editors: Mary Wootters and Laura Sanità; Article No. 27; pp. 27:1–27:14



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Partitioning a graph into connected subgraphs is a problem that arises in many application areas such as parallel processing, road network decomposition, image processing, districting problems, and robotics [34, 35, 4, 1, 39]. Often in these applications, it is required to find a partition into a specified number k of connected subgraphs. For instance, in the parallel processing applications, the number of processors is restricted, and in robotics applications, the number of robots available is restricted. Formally, we call a partition T_1, T_2, \dots, T_k of the vertex set of graph, a *connected (k -)partition*, if the subgraph induced by the vertices in T_i is connected for each $1 \leq i \leq k$.

The typical modeling objective in such connected partition problems is to balance sizes among the k parts. Sometimes one needs to consider a vertex-weighted generalization e.g. weights representing the required amount of work at the entity corresponding to the vertex. The two classical balancing objectives for such k -partitions are to maximize the total weight of the lightest part, or to minimize the weight of the heaviest part. These objectives yield the following two versions of the *balanced connected partition problem* (BCP).

MAX-MIN BCP (MIN-MAX BCP)

Input: A vertex-weighted graph $G = (V, E, w)$ where $w : V \rightarrow \mathbb{N}$, and $k \in \mathbb{N}$.

Task: Find a connected k -partition T_1, \dots, T_k of G maximizing $\min_{i \in [k]} w(T_i)$ (minimizing $\max_{i \in [k]} w(T_i)$ resp.).

On general graphs, both variants of BCP are NP-hard [5], and hence the problems have been mostly studied from the viewpoint of approximation algorithms [6, 8, 9, 11, 12]. Most of the known results are for small values of k , and there are some results also for special classes like grid graphs or graphs of bounded treewidth (see related work section for further details). The currently best known polynomial-time approximation for general graphs for any k is a 3-approximation for both MAX-MIN and MIN-MAX BCP by Casel et. al. [6].

Intuitively, an obstacle for getting a balanced connected partition is a large induced star, i.e. a tree with one internal node and c leaves, denoted by $K_{1,c}$. We say a graph is *c -claw-free* or *$K_{1,c}$ -free* if it does not contain an induced $K_{1,c}$ as subgraph. For such graphs, we give a very efficient $(c - 1)$ -approximation algorithm for both the min-max and max-min objective. In particular by setting $c = 3$, we get a 2-approximation on $K_{1,3}$ -free graphs, better known as *claw-free graphs*.

Claw free graphs have been widely studied by Seymour and Chudnovsky in a series of seven papers under the name *Claw-free graphs I-VII* ([14]-[20]), who also provide a structure theorem for these graphs [21]. Some interesting examples of such graphs are line graphs, proper circular interval graphs and de-Bruijn graphs [22]. Apart from their structural properties, claw-free graphs have been studied in the context of obtaining efficient algorithms for several interesting problems, see e.g. [27, 24, 23].

Although, for $c > 3$ our algorithm gives a worse guarantee than the algorithm by Casel et. al. [6], we note that their algorithm runs in $\mathcal{O}(\log(X^*)k^2|V||E|)$ time for MAX-MIN BCP and in $\mathcal{O}(\log(X^*)|V||E|(\log \log X^* \log(|V|w_{\max}) + k^2))$ time for MIN-MAX BCP, where X^* denotes the optimum value and $w_{\max} := \max_{v \in V} w(v)$ the maximum weight of a vertex, whereas our algorithms give an $\mathcal{O}(\log(X^*)|E|)$ runtime for MAX-MIN BCP and an $\mathcal{O}(|E|)$ runtime for MIN-MAX BCP. Moreover, our algorithms are less technical and hence much easier to implement. We prove the following statements.

► **Theorem 1.** *Given a vertex-weighted $K_{1,c}$ -free graph $G = (V, E, w)$ and $k \in \mathbb{N}$, a $(c - 1)$ -approximation for MIN-MAX BCP can be computed in $\mathcal{O}(|E|)$ time.*

► **Theorem 2.** *Given a vertex-weighted $K_{1,c}$ -free graph $G = (V, E, w)$ and $k \in \mathbb{N}$, a $(c - 1)$ -approximation for MAX-MIN BCP can be computed in time $\mathcal{O}(\log(X^*)|E|)$, where X^* is the optimum value.*

Since line graphs are $K_{1,3}$ -free, these results directly imply efficient approximations for the following edge-partition versions of BCP. A k -partition of the edges of a graph, is called a *connected edge k -partition*, if the subgraph induced by the edges in each part is connected. In the problem Min-Max (Max-Min) balanced connected edge partition (BCEP), one searches for a connected edge k -partition of an edge-weighted graph minimizing the maximum (resp. maximizing the minimum) weight of the parts. This problem is equivalent to finding a connected k -partition of the vertices in the line graph of the input graph. The best known approximation for BCEP is for graphs with no edge weight larger than $w(G)/2k$. For such graphs, [13] give an algorithm that finds a connected edge k -partition, such that the weight of the heaviest subgraph is at most twice as large as the weight of the lightest subgraph, implying a 2-approximation for MIN-MAX and MAX-MIN BCEP. In comparison, our algorithms achieve the same approximation guarantee without restrictions on the edge weights.

► **Corollary 3.** *MIN-MAX BCEP and MAX-MIN BCEP have 2-approximations in polynomial time.*

An extension of BCP is demanding for fixed (possibly heterogeneous) size targets for each of the k parts. More precisely, given a graph G and w_1, \dots, w_k with $\sum_{i=1}^k w_i = n$, the task is to find a partition T_1, \dots, T_k where each T_i has size w_i and induces a connected subgraph. Such a connected k -partition with the fixed target weights exists for G only if G meets certain structural properties; a $K_{1,3}$ for example has no connected 2-partition T_1, T_2 with $|T_1| = |T_2| = 2$. A characterization of when such a connected partition always exists was independently proved by Györi [26] and Lovász [32]: They showed that in any k -connected graph a connected k -partition satisfying the target weights always exists. This result is the famous Györi-Lovász Theorem (GL theorem, for short):

► **Theorem 4** (Györi-Lovász Theorem [26, 32]). *Given a k -connected graph $G = (V, E, w)$, $n_1, \dots, n_k \in \mathbb{N}$ such that $\sum_{i=1}^k n_i = |V|$, and k terminal vertices $t_1, \dots, t_k \in V$, there exists a connected k -partition T_1, \dots, T_k of V such that for each $i \in [k]$, $|T_i| = n_i$ and $t_i \in T_i$.*

Recently, the theorem was generalized to vertex-weighted graphs as:

► **Theorem 5** (Weighted Györi-Lovász Theorem [7, 10, 28]). *Given a vertex-weighted k -connected graph $G = (V, E, w)$, $w_1, \dots, w_k \in \mathbb{N}$ such that $\sum_{i=1}^k w_i = w(V)$, and k terminal vertices t_1, \dots, t_k , there exists a connected k -partition T_1, \dots, T_k of V such that $w_i - w_{\max} < w(T_i) < w_i + w_{\max}$, and $t_i \in T_i$ for each $i \in [k]$, where w_{\max} is the largest vertex weight.*

We refer to the partition guaranteed by the (weighted) GL theorem as *GL partition*. We will however not consider the terminal vertices in the GL partitions in this work.

The GL theorem has found some applications in the field of algorithms. Chen et. al. [10] use it for proving the existence of low-congestion *confluent flows* in k -connected graphs. Further, Löwenstein et. al. [33] and Chandran et. al. [7] use it for finding spanning trees with low *spanning tree congestion*. Perhaps, the reason why such a strong combinatorial statement has not found further applications is that we do not know how to efficiently compute GL

partitions. About five decades after the discovery of the GL theorem, polynomial time algorithms for finding a GL partition (even in the unweighted case without terminals) are only known for $k \leq 4$ [37, 38, 28]. The fastest algorithm for general k takes $\Omega(2^n)$ time [7, 29]. Neither are there any impossibility results to exclude efficient computability of such partitions. Even when k is part of the input, a polynomial time algorithm is not ruled out.

The absence of efficient algorithms for finding exact GL partitions motivates finding GL-style partitions that approximately satisfy the weight targets. In this paper we present polynomial time algorithms for such approximations. First we give an algorithm for a “half-bounded” GL partition, in the sense that we can guarantee an approximate upper or lower bound on the weight of the parts.

► **Theorem 6.** *Let $G = (V, E, w)$ be a k -connected vertex-weighted graph and $w_1, \dots, w_k \in \mathbb{N}$ with $\sum_{i=1}^k w_i = w(G)$, and $\min_{i \in [k]} w_i \geq \max_{v \in V} w(v)$. A connected k -partition T_1, \dots, T_k of V such that either $w(T_i) \geq \frac{1}{3}w_i$ for every $i \in [k]$ (lower-bound version) or $w(T_i) \leq 3w_i$ for every $i \in [k]$ (upper-bound version) can be computed in time $\mathcal{O}(k|V|^2|E|)$.*

We then extend this result to a lower and upper bounded partition.

► **Theorem 7.** *Let $G = (V, E, w)$ be a k -connected vertex-weighted graph and $w_1, \dots, w_k \in \mathbb{N}$ with $\sum_{i=1}^k w_i = w(G)$, and $\min_{i \in [k]} w_i \geq \max_{v \in V} w(v)$, and $r := \frac{\max_{i \in [k]} w_i}{\min_{j \in [k]} w_j}$. Then, a connected k -partition T_1, \dots, T_k of V such that $\frac{1}{3}w_i \leq w(T_i) \leq \max\{r, 3\}w_i$ for every $i \in [k]$ can be found in time $\mathcal{O}(k|V|^2|E|)$.*

In particular, Theorem 7 implies the following approximately balanced partition of k -connected graphs.

► **Corollary 8.** *Let $G = (V, E, w)$ be a k -connected vertex-weighted graph such that $w(G) \geq k \max_{v \in V} w(v)$. Then, a connected k -partition T_1, \dots, T_k of V such that $\frac{1}{3} \left\lfloor \frac{w(G)}{k} \right\rfloor \leq w(T_i) \leq 3 \left\lceil \frac{w(G)}{k} \right\rceil$ for every $i \in [k]$ can be found in time $\mathcal{O}(k|V|^2|E|)$.*

To the best of our knowledge, these are the first polynomial time algorithms that *approximate* the GL theorem. We believe that such an efficient approximation will result in the theorem being used for developing algorithms in the future. Especially, we are hopeful that the both-side approximation for balanced connected partition of k -connected graphs will find applications. We remark, however that for the above mentioned applications of confluent flows and spanning tree congestion, the terminal vertices are essential and hence our algorithms cannot be used. An interesting future direction would be to extend our results to the setting with terminals.

Observe that Corollary 8 in some sense yields a 3-approximation simultaneously for MIN-MAX and MAX-MIN BCP in k -connected graphs. In this regard, it is interesting to note that the $+/-w_{\max}$ slack given in the weighted GL theorem is enough to retain hardness in the following sense: even for $k = 2$, MIN-MAX BCP and MAX-MIN BCP remain strongly NP-hard when restricted to 2-connected graphs; and the corresponding hardness-proof given in [8] also constructs an instance with $w(G) \geq k \max_{v \in V} w(v)$. This hardness can be extended to k -connected graphs for any fixed $k \geq 2$ (see [8][Theorem 3] for more details).

Lastly, we point out that this paper is only a short version and refer the reader for more technical details and complete proofs to the full version of it.

1.1 Related work

Both variants of BCP were first introduced for trees [36, 31]. Under this restriction, a linear time algorithm was provided for both variants in [25]. This is particularly important since different heuristics transform the original instance to a tree to efficiently solve the problem, see [13, 39]. For both variants of BCP, a 3-approximation is given in [6], which is the best known approximation in polynomial time. With respect to lower bounds, it is known that there exists no approximation for MAX-MIN BCP with a ratio below $6/5$, unless $P = NP$ [8]. For the unweighted case, a $\frac{k}{2}$ -approximation for MIN-MAX BCP with $k \geq 3$, is given in [11].

Balanced connected partitions for fixed small values of k , denoted BCP_k , have also been studied extensively. The restriction BCP_2 , i.e. balanced connected bipartition, is already NP-hard [5]. On the positive side, a $\frac{4}{3}$ -approximation for MAX-MIN BCP_2 is given in [12], and in [11] this result is used to derive a $\frac{5}{4}$ -approximation for MIN-MAX BCP_2 . Considering tripartitions, MAX-MIN BCP_3 and MIN-MAX BCP_3 can be approximated with ratios $\frac{5}{3}$ and $\frac{3}{2}$, respectively [9].

Regarding special graph classes, BCP has been investigated in grid graphs and series-parallel graphs. While it was shown that BCP is NP-hard for arbitrary grid graphs [2], the MAX-MIN BCP can be solved in polynomial time for ladders, i.e., grid graphs with two rows [3]. For the class of series-parallel graphs, Ito et. al. [30] observed that BCP remains weakly NP-hard (by a simple reduction from the Partition problem) and gave a pseudo-polynomial-time algorithm for both variants of BCP. They also showed that their algorithm can be extended to graphs with bounded tree-width.

The GL Theorem was independently proved by Györi [26] and Lovász [32]. Györi used an elementary graph theoretic approach while Lovász used ideas from topology. Lovász's proof also works for directed graphs. The Györi-Lovász Theorem is extended to weighted directed graphs by Chen et. al. [10] and Györi's original proof was generalized to weighted undirected graphs by Chandran et. al. [7]. Both papers only gave upper bounds of $w_i + w_{\max}$ on the weight of partition T_i and did not provide any lower bounds. Later Hoyer [28] showed that the method of Chandran et. al. [7] can be also extended to give the lower bound $w_i - w_{\max}$, even for directed graphs. Polynomial algorithms to also compute GL partitions are only known for the particular cases $k = 2, 3, 4$ [37, 38, 28] and all $k \geq 5$ are still open.

2 Preliminaries

By \mathbb{N} we denote the natural numbers without zero. We use $[k]$ to denote the set $\{1, \dots, k\}$.

All the graphs that we refer to in this paper are simple, finite and connected. Consider a graph $G = (V, E)$. We denote by $V(G)$ and $E(G)$ the set of vertices and edges of G respectively, and if the graph we refer to is clear, we may simply write V and E . For a set of vertex sets $\mathcal{S} \subseteq 2^V$ we use $V(\mathcal{S})$ to denote $\bigcup_{S \in \mathcal{S}} S$. We denote an edge $e = \{u, v\} \in E(G)$ by uv and the neighborhood of a vertex $v \in V$ in G by $N_G(v) = \{u \in V \mid uv \in E(G)\}$. Similarly we denote the neighborhood of a vertex set $V' \subseteq V$ in G by $N_G(V')$, that is $\bigcup_{v \in V'} N_G(v) \setminus V'$. We may omit the subscript G when the graph is clear from the context. We use $\Delta(G)$ to denote the maximum degree of G .

We denote a *vertex-weighted graph* by $G = (V, E, w)$ where w is a function assigning integer weights to vertices $w: V \rightarrow \mathbb{N}$, and V and E are vertex and edge sets. We denote by w_{\min} and by w_{\max} , $\min_{v \in V} w(v)$ and $\max_{v \in V} w(v)$, respectively. For any $V' \subseteq V$, we use $w(V')$ to denote the sum of weights of the vertices in V' . For a subgraph H of G we use $w(H)$ to denote $w(V(H))$, and refer to it as the *weight of the subgraph* H . For a rooted tree T and a vertex x in T , we use T_x to denote the rooted subtree of T rooted at x .

For $V' \subseteq V$ we denote by $G[V']$ the graph induced by V' , i.e. $G[V'] = (V', E')$ with $E' = E \cap (V' \times V')$. For vertex-weighted graphs, induced subgraphs inherit the vertex-weights given by w . For $V' \subseteq V$ we also use $G - V'$ to denote the subgraph $G[V \setminus V']$. Similarly, if V' is a singleton $\{v\}$ we also write $G - v$. For graphs G_1 and G_2 , we use $G_1 \cup G_2$ to denote the graph on vertices $V(G_1) \cup V(G_2)$ with edge set $E(G_1) \cup E(G_2)$.

Let $\mathcal{U} = \{U_1, \dots, U_r\}$ be such that each $U_i \subseteq V(G)$. We call \mathcal{U} a *connected packing* of $V(G)$ if each $G[U_i]$ is connected, and the sets in \mathcal{U} are pairwise disjoint. A connected packing \mathcal{U} is called *connected vertex partition* (CVP) of V , if also $\cup_{i=1}^r U_i = V(G)$. We denote a CVP that has k vertex sets as CVP_k . For any $\mathcal{U}' \subseteq \mathcal{U}$, we define $V(\mathcal{U}') := \cup_{U' \in \mathcal{U}'} U'$, and the weight $w(\mathcal{U}') := w(V(\mathcal{U}'))$. Let \mathcal{I} be an interval. If \mathcal{U} is a CVP and $w(U_i) \in \mathcal{I}$ for all $i \in [r]$, then we say that \mathcal{U} is an \mathcal{I} -connected vertex (r -)partition (\mathcal{I} -CVP $_k$ or just \mathcal{I} -CVP) of V . If \mathcal{U} is a connected-packing and $w(U_i) \in \mathcal{I}$ for all $i \in [r]$, then we say that \mathcal{U} is a \mathcal{I} -connected packing of V .

3 Approximation for BCP on c -claw-free graphs

In this section we give an idea of how to prove Theorems 1 and 2 by giving a $(c - 1)$ -approximation for MAX-MIN BCP and MIN-MAX BCP on $K_{1,c}$ -free graphs. We assume $c \geq 3$ as $c \leq 2$ gives trivial graph classes. We first show that a connected partition for $K_{1,c}$ -free graphs with parts of size in $[\lambda, (c - 1)\lambda]$ for some fixed λ can be found in linear time. For MAX-MIN BCP, this algorithm has to be called many times while doing a binary search for the optimum value. We point out that it is not difficult to adapt these algorithms to unconnected graphs achieving the same approximation results.

Exploiting that each vertex in any DFS-tree of a $K_{1,c}$ -free graph has at most $c - 1$ children, we can carefully extract connected components of a fixed size while also maintaining a DFS-tree for the remaining graph. Also, this can be done very efficiently, as stated in the following result.

► **Lemma 9.** *Given a $K_{1,c}$ -free graph G and a DFS-tree of G . For any $w(G) \geq \lambda \geq w_{\max}$, there is an algorithm that finds a connected vertex set S such that $\lambda \leq w(S) < (c - 1)\lambda$ and $G - S$ is connected, in $\mathcal{O}(|V|)$ time. Furthermore, the algorithm finds a DFS-tree of $G - S$.*

We use `BalancedPartition` to denote the algorithm that exhaustively applies Lemma 9. Observe that `BalancedPartition` produces a connected partition S_1, \dots, S_m where $w(S_i) \in [\lambda, (c - 1)\lambda]$ for every $i \in [m - 1]$ and $w(S_m) < (c - 1)\lambda$ in linear time, where the achieved runtime follows by saving already processed subtrees.

Theorem 1 now follows from running `BalancedPartition` with $\lambda = \max\{w_{\max}, \frac{w(G)}{k}\}$. Note that this choice of λ is a trivial lower bound for the optimum value.

As already mentioned, to prove Theorem 2, we first need to find an input parameter λ for Algorithm `BalancedPartition` that provides the desired $(c - 1)$ -approximation.

Let (G, k) be an instance of MAX-MIN BCP, where G is a $K_{1,c}$ -free graph. Let X^* be the optimal value for the instance (G, k) . For any given $X \leq w(G)/k$, we design an algorithm that either gives a $[\lfloor X/(c - 1) \rfloor, \infty)$ -CVP $_k$, or reports that $X > X^*$. Note that $X^* \leq w(G)/k$. Once we have this procedure in hand, a binary search for the largest X in the interval $(0, \lceil w(G)/k \rceil]$ for which we find a $[\lfloor X/(c - 1) \rfloor, \infty)$ -CVP $_k$ can be used to obtain an approximate solution for MAX-MIN BCP.

Algorithm MaxMinApx. First remove all vertices of weight more than $\lambda = \lfloor X/(c - 1) \rfloor$ and save them in H . Then save the connected components of weight less than λ in \mathcal{Q} .

Let $\mathcal{V} = \{V_1, \dots, V_\ell\}$ be the connected components of $G - (H \cup V(\mathcal{Q}))$. Apply algorithm `BalancedPartition` on each $G[V_i]$ with λ as input parameter to obtain $\mathcal{S}^i = \{S_1^i, \dots, S_{m_i}^i\}$ for every $i \in [\ell]$. If for some $i \in [\ell]$ the weight $w(S_{m_i}^i)$ is less than λ , then merge this vertex set with $S_{m_i-1}^i$ and accordingly update \mathcal{S}^i . Further, compute a (λ, ∞) -CVP $_{|H|}$ \mathcal{S}^H of $G[H \cup V(\mathcal{Q})]$ as follows: for each $h \in H$, we have a set $S^h \in \mathcal{S}^H$ with $h \in S^h$; we add each $Q \in \mathcal{Q}$ to some S^h such that $h \in N(Q)$. Let $\mathcal{S} = \mathcal{S}^H \cup \bigcup_{i=1}^{\ell} \mathcal{S}^i$. If $|\mathcal{S}| \geq k$, then merge connected sets arbitrarily in \mathcal{S} until $|\mathcal{S}| = k$ and return \mathcal{S} . If $|\mathcal{S}| < k$, report that $X > X^*$.

We point out that a $[\lambda, \infty)$ -CVP $_j$ with $j > k$, can easily be transformed to a $[\lambda, \infty)$ -CVP $_k$, since the input graph is connected. It is not hard to see that if algorithm `MaxMinApx` returns \mathcal{S} then this is a $[\lambda, \infty)$ -CVP $_k$ of V . The most complicated part of proving that `MaxMinApx` works correctly is showing that if it terminates with $|\mathcal{S}| < k$ and reports $X > X^*$ that this is indeed true.

► **Lemma 10.** *If Algorithm `MaxMinApx` terminates with $|\mathcal{S}| < k$, then $X > X^*$.*

Proof. Let $H, \mathcal{Q}, \mathcal{V} = \{V_1, \dots, V_\ell\}$ be the computed vertices and connected vertex sets in the algorithm for $\lambda = \lfloor X/(c-1) \rfloor$, respectively. Recall that $w(V_i) \geq \lambda$ for every $V_i \in \mathcal{V}$ and $w(Q) < \lambda$ for every $Q \in \mathcal{Q}$. Let $\mathcal{S}^* = \{S_1^*, \dots, S_k^*\}$ be an optimal solution of (G, k) , i.e., \mathcal{S}^* is an $[X^*, \infty)$ -CVP $_k$ of V . Consider the sets $\mathcal{V}^{H \cup \mathcal{Q}} := \{S_i^* \in \mathcal{S}^* \mid S_i^* \cap (H \cup V(\mathcal{Q})) \neq \emptyset\}$, $\mathcal{V}^1 := \{S_i^* \in \mathcal{S}^* \mid S_i^* \cap V_1 \neq \emptyset\} \setminus \mathcal{V}^{H \cup \mathcal{Q}}$, \dots , $\mathcal{V}^\ell := \{S_i^* \in \mathcal{S}^* \mid S_i^* \cap V_\ell \neq \emptyset\} \setminus \mathcal{V}^{H \cup \mathcal{Q}}$. We claim that these sets are a partition of \mathcal{S}^* . This follows directly from the fact that H separates all $V_i \in \mathcal{V}$ and all $Q \in \mathcal{Q}$ from each other. That is, for an $i \in [k]$ and $j \in [\ell]$ the connected vertex set S_i^* with $S_i^* \cap V_j \neq \emptyset$ and $S_i^* \cap V \setminus V_j \neq \emptyset$ contains at least one $h \in H$ and hence $S_i^* \in \mathcal{V}^{H \cup \mathcal{Q}}$. Otherwise, if $S_i^* \subseteq V_j$, then $S_i^* \in \mathcal{V}^j$.

Suppose `MaxMinApx` terminates with $|\mathcal{S}| < k$ although $X \leq X^*$. We show that $|\mathcal{V}^{H \cup \mathcal{Q}}| \leq |H|$ and $|\mathcal{V}^i| \leq |\mathcal{S}^i|$ for every $i \in [\ell]$, implying that $|\mathcal{S}^*| \leq |H| + \sum_{i=1}^{\ell} |\mathcal{S}^i| = |\mathcal{S}| < k$, which contradicts $|\mathcal{S}^*| = k$.

First, we show $|\mathcal{V}^{H \cup \mathcal{Q}}| \leq |H|$. For this, it is sufficient to prove that $S_i^* \cap H \neq \emptyset$ for each $S_i^* \in \mathcal{V}^{H \cup \mathcal{Q}}$ as \mathcal{S}^* is a partition of V . We prove this by contradiction. Suppose there is an $S_i^* \in \mathcal{V}^{H \cup \mathcal{Q}}$, such that $S_i^* \cap H = \emptyset$. This implies that $S_i^* \subseteq Q$ for some $Q \in \mathcal{Q}$, since H separates every $Q \in \mathcal{Q}$ from every other $Q' \in \mathcal{Q} \setminus \{Q\}$ and from the vertices $V \setminus (H \cup V(\mathcal{Q}))$. Thus, $w(S_i^*) \leq w(Q) < \lambda$ by the definition of \mathcal{Q} and therefore $w(S_i^*) < \lambda = \lfloor X/(c-1) \rfloor \leq \lfloor X^*/(c-1) \rfloor$, contradicting $\min_{i \in [k]} w(S_i^*) = X^*$.

It remains to show that $|\mathcal{V}^i| \leq |\mathcal{S}^i|$ for every $i \in [\ell]$. Fix an $i \in [\ell]$ and let $G[V_i]$ with λ be the input when calling algorithm `BalancedPartition`. Observe that the input is valid, since $G[V_i]$ is connected by definition and $w(G[V_i]) \geq \lambda \geq \max_{v \in V_i} w(v)$ as H contains all vertices that have weight more than λ . Algorithm `BalancedPartition` provides a CVP $\mathcal{S}^i = \{S_1^i, \dots, S_{m_i}^i\}$ of V_i with $w(S_j^i) \in [\lambda, (c-1)\lambda)$ for every $j \in [m_i-1]$ and $w(S_{m_i}^i) < (c-1)\lambda$. Consider \mathcal{S}^i before merging, i.e. we do not merge $S_{m_i}^i$ to $S_{m_i-1}^i$ in the algorithm `MaxMinApx` if $w(S_{m_i}^i) < \lambda$. That is, $w(S_{m_i}^i) < \lambda$ is possible, and we need to show $|\mathcal{V}^i| \leq m_i - 1 = |\mathcal{S}^i| - 1$. Observe for $\mathcal{S}^* \in \mathcal{V}^i$ that $\mathcal{S}^* \subseteq V_i$, i.e. $\sum_{j=1}^{m_i} w(S_j^i) \geq \sum_{\mathcal{S}^* \in \mathcal{V}^i} w(\mathcal{S}^*)$. As a result, we have $|\mathcal{S}^i|X \geq |\mathcal{S}^i|(c-1)\lambda > \sum_{j=1}^{m_i} w(S_j^i) \geq \sum_{\mathcal{S}^* \in \mathcal{V}^i} w(\mathcal{S}^*) \geq |\mathcal{V}^i|X^*$. Consequently, by $X \leq X^*$ we obtain $|\mathcal{V}^i| < |\mathcal{S}^i|$, which leads to $|\mathcal{V}^i| \leq |\mathcal{S}^i| - 1$. ◀

4 Approximation of the Györi-Lovász Theorem for k -connected Graphs

Our algorithms for the approximate GL theorems are based mainly on the following combinatorial lemma concerning certain vertex separators, that leads to useful structures in

k -connected graphs. Let $G = (V, E, w)$ be a connected vertex-weighted graph and let λ be an integer. We say $s \in V$ is a λ -separator if all connected components of $G - \{s\}$ weigh less than λ . We say G is λ -dividable if there is a $[\lambda, \infty)$ -CVP₂ of V .

► **Lemma 11** ([6]). *Let $G = (V, E, w)$ be a connected vertex-weighted graph and let $\lambda > w_{\max}$ be an integer. If $w(G) > 3(\lambda - 1)$, then either G is λ -dividable or there is a λ -separator. Furthermore, finding the connected vertex sets in case G is λ -dividable and finding the λ -separator in the other case can be done in $\mathcal{O}(|V||E|)$ time.*

4.1 Bounded Partition for k -connected Graphs

In this section, we give an algorithm for computing approximate GL partitions with one-side approximation bound (either lower bound or upper bound), thus proving Theorem 6. For this, we first use the following theorem, from which Theorem 6 follows as below.

► **Theorem 12.** *Let $G = (V, E, w)$ be a k -connected vertex-weighted graph and let $w_1, \dots, w_k \in \mathbb{N}$ with $\sum_{i=1}^k w_i = w(G)$, and $\min_{i \in [k]} w_i \geq \max_{v \in V} w(v)$. A set of connected vertex sets $\mathcal{T} = \{T_1, \dots, T_\ell\}$ with $\ell \leq k$ and $\alpha w_i \leq w(T_i) \leq 3\alpha w_i$ for every $i \in [\ell]$ can be computed in time $\mathcal{O}(k|V|^2|E|)$. Moreover, if $\ell < k$, then \mathcal{T} is also a CVP of V .*

By Theorem 12 we can derive Theorem 6 using $\alpha = 1/3$ and $\alpha = 1$ for the lower bound and upper bounded version, respectively.

In the following we always assume that w_1, \dots, w_k is sorted in descending order. To now give the algorithm proving Theorem 12, we make use of Lemma 11. For this, we first need to ensure that $w_{\max} < \alpha w_k$. Therefore, we perform a preprocessing step until we reach an instance that satisfies $w_{\max} < \alpha w_k$. Suppose $w_{\max} \geq \alpha w_\ell$, where ℓ is the smallest index in $[k]$ that satisfies this inequality. We remove a vertex v_{\max} with $w(v_{\max}) = w_{\max}$ from G and w_ℓ from \mathcal{W} . Further, we set $T_\ell = \{v_{\max}\}$ and consider the set for index ℓ to be finished, i.e., we now aim to find a set $\mathcal{T} = \{T_1, \dots, T_{\ell-1}, T_{\ell+1}, \dots, T_k\}$ according to $\mathcal{W} = \{w_1, \dots, w_{\ell-1}, w_{\ell+1}, \dots, w_k\}$. Observe that since we only deleted one vertex, G now is at least $(k - 1)$ -connected, and $|\mathcal{W}| = k - 1$. Note that since $w_{\max} \leq w_k \leq w_\ell \leq 3\alpha w_\ell$, we have $w(T_\ell) \in [\alpha w_\ell, 3\alpha w_\ell]$ as required. Also, we obtain $w(G) \geq \sum_{w_j \in \mathcal{W}} w_j$ after removing w_ℓ from \mathcal{W} and v_{\max} from G .

After this preprocessing, we can assume that we have a k -connected graph $G = (V, E, w)$ and natural numbers w_1, \dots, w_k sorted in descending order, where $\sum_{i=1}^k w_i \leq w(G)$ and $w_{\max} < \alpha w_k$.

On such a graph G we then gradually build a packing \mathcal{T} with the help of Lemma 11. During our algorithm to build \mathcal{T} we ensure that at each step $\mathcal{T} = \{T_1, T_2, \dots, T_{i-1}\}$ where each $T_j \in \mathcal{T}$ is a connected vertex set with weight in $[\alpha w_j, 3\alpha w_j]$ for each $j \in [i - 1]$. We then search in the remaining graph for the next set T_i and always use \overline{G} to denote the graph $G \setminus V(\mathcal{T})$. We say a connected subgraph is i -small if it has weight less than αw_i and i -big otherwise. In case we reach a situation, where \overline{G} has no connected component that is i -big, we have to alter the already built sets T_1, T_2, \dots, T_{i-1} to build T_i . For this, we use \mathcal{T}_a to denote the set of all $T_j \in \mathcal{T}$ that have no (αw_j) -separator, and \mathcal{T}_b to denote the set of all $T_j \in \mathcal{T}$ that have an (αw_j) -separator. For $T_j \in \mathcal{T}_b$ with an (αw_j) -separator s we use $C(T_j)$ to denote the connected components of $G[T_j \setminus \{s\}]$ (if there is more than one (αw_j) -separator, fix one of them arbitrarily). The following Algorithm BoundedGL formally explains our routine to build \mathcal{T} .

Algorithm BoundedGL

1. Initialize $\mathcal{T} := \emptyset$ as container for the desired connected-vertex-packing T_1, \dots, T_k of G and initialize $i := 1$ as an increment-variable.
2. While $\overline{G} = G \setminus V(\mathcal{T})$ is not the empty graph: //main loop
 - a. Find a connected vertex set T_i having weight in $[\alpha w_i, 3\alpha w_i]$, add T_i to \mathcal{T} , and increment i by one. If $i = k + 1$ then terminate the algorithm.
// See Lemma 14 for correctness of this step
 - b. While \overline{G} is not empty and has no i -big connected component: //inner loop
Pick an i -small connected component Q of \overline{G} . Pick a $T_j \in \mathcal{T}$ such that either $T_j \in \mathcal{T}_a$ and Q has an edge to T_j (Case 1), or $T_j \in \mathcal{T}_b$ and Q has an edge to some component $Q' \in C(T_j)$ (Case 2). // The occurrence of at least one of these cases is shown in Lemma 15.
If $w(T_j \cup Q) \leq 3\alpha w_j$ then update T_j to $T_j \cup Q$. Otherwise:
 - i. Case 1 ($T_j \in \mathcal{T}_a$): Apply the following **Divide-routine** on $T_j \cup Q$: Use Lemma 11 to compute a $[\alpha w_j, \infty)$ -CVP₂ V_1, V_2 of $T_j \cup Q$. Set $T_j = V_2$ (i.e. V_1 goes to \overline{G}).
 - ii. Case 2 ($T_j \in \mathcal{T}_b$): remove Q' from T_j (i.e. Q' goes back to \overline{G}) if $T_j \cup Q$ is not αw_j -dividable. Otherwise, apply divide routine on $T_j \cup Q$.

To prove the correctness of the algorithm, we will show that the following invariant is maintained.

► **Lemma 13.** *Algorithm BoundedGL maintains a packing $\mathcal{T} = \{T_1, T_2, \dots, T_{i-1}\}$ where each $T_j \in \mathcal{T}$ is a connected vertex set having weight in $[\alpha w_j, 3\alpha w_j]$.*

Towards proving this we use the following fact.

► **Lemma 14.** *Whenever the divide routine in algorithm BoundedGL is to be executed, there is at least one i -big component in \overline{G} .*

Proof. When $i = 1$ i.e. in the first main loop iteration, this holds because $\overline{G} = G$ is connected and $\alpha w_1 \leq w_1 \leq \sum_{i=1}^k w_i \leq w(G)$. For the subsequent iterations, the divide routine is only applied after the inner loop is terminated which only happens if either \overline{G} is empty or has an i -big component. In case \overline{G} is empty, then the algorithm terminates. So, if the algorithm applies the divide routine in the inner loop, then \overline{G} has an i -big component. ◀

Proof of Lemma 13. We increment i only in Step 2a. Before incrementing i , we add T_i to \mathcal{T} while ensuring that $w(T_i) \in [\alpha w_i, 3\alpha w_i]$ and $G[T_i]$ is connected. In Lemma 14, we prove that whenever the divide routine is about to be executed, there is an i -big component in \overline{G} , ensuring the existence of such a T_i . Once a T_j is added to \mathcal{T} , it is then modified only in Step 2b. So let us look into how it gets modified in Step 2b. If the condition $w(T_j \cup Q) \leq 3\alpha w_j$ is satisfied then it is clear that the new $T_j = T_j \cup Q$ also satisfies the weight constraints. Since Q has an edge to T_j and T_j and Q each were connected, it is also clear that the new T_j remains connected. So now consider the case when $w(T_j \cup Q) > 3\alpha w_j$. In Case 1 ($T_j \in \mathcal{T}_a$), we call the divide routine and the new T_j is the set V_2 returned by the routine. The set V_2 is connected due to the property of the divide routine. To see that it also satisfies the weight constraints, observe that $w(V_1 \cup V_2)$ is at most $4\alpha w_j$ as $w(T_j)$ was at most $3\alpha w_j$ and $w(Q) < \alpha w_j \leq \alpha w_j$. Since $w(V_1), w(V_2) \geq \alpha w_j$, we then have $w(V_2) \in [\alpha w_j, 3\alpha w_j]$. So it only remains to consider Case 2 ($T_j \in \mathcal{T}_b$). The case when $T_j \cup Q$ is αw_j -dividable is analog to Case 1. We know $w(Q') < \alpha w_j$ by definition. Also, since $w(T_j \cup Q)$ was more than $3\alpha w_j$ and $w(Q) < \alpha w_j$, we have that $w(T_j)$ was at least $2\alpha w_j$. Thus the new $T_j = T_j \setminus Q'$ has weight in $[\alpha w_j, 3\alpha w_j]$. Also, the new T_j is connected by the definition of Q' . ◀

It is clear from the algorithm that termination occurs only if \overline{G} is empty or $i = k + 1$. Then using Lemma 13, it is clear that \mathcal{T} contains the required packing as claimed in Theorem 12, provided that Step 2b runs correctly and the inner loop terminates, which we prove in the two lemmas below.

► **Lemma 15.** *In Step 2b at least Case 1 or Case 2 occurs.*

Proof. Suppose Case 1 does not occur i.e., Q does not have an edge to any $T_j \in \mathcal{T}_a$. For $T_j \in \mathcal{T}_b$, let s_j denote the fixed (αw_j) -separator vertex. Since G is k -connected and $|\mathcal{T}| < k$, there is an edge from Q to at least one vertex in $\bigcup_{T_j \in \mathcal{T}_b} T_j \setminus \{s_j\}$. Thus, Case 2 occurs. ◀

► **Lemma 16.** *The inner loop runs correctly and terminates after at most $|V|^2$ iterations.*

Proof. The occurrence of one of the two cases in Step 2b is shown in Lemma 15. For the correctness of Case 1, observe that we can use Lemma 11 to divide $T_j \cup Q$, as $T_j \cup Q$ cannot have an (αw_j) -separator (this would give an (αw_j) -separator for T_j implying that $T_j \in \mathcal{T}_b$). It remains to prove that the inner loop terminates as claimed. If Step 2(b)i is executed, then an i -big component is created in \overline{G} as \overline{G} now contains V_1 returned by the divide-routine, and hence the loop is terminated. The same yields if we apply the divide routine in Step 2(b)ii. So, suppose the inner loop never executes the divide routine. In the other cases, either a connected component is deleted from \overline{G} or new vertices are added to a connected component in \overline{G} . Also note that new connected components are not introduced to \overline{G} and vertices are not deleted from existing connected components (except when the whole connected component is removed; also, two or more connected components may merge due to the introduction of new vertices to \overline{G}). Thus, after $|V|^2$ iterations either there is an i -big component or \overline{G} is empty. ◀

It is tempting to think that one could use Theorem 12 to derive a CVP $\mathcal{T} = \{T_1, \dots, T_k\}$ such that $\alpha w_i \leq w(T_i) \leq 3\alpha w_i$ for each $i \in [k]$. If Algorithm **BoundedGL** terminates with $\ell < k$, then \mathcal{T} is a partition of the vertices in G , and we only have trouble with the lower bound on T_j for $\ell < j \leq k$. Otherwise, if it terminates with $\ell = k$, then \mathcal{T} satisfies all lower bounds, but might not be a partition. Assigning the remaining vertices in G to turn \mathcal{T} into a CVP in this case might yield violations of the upper bound. Since $\alpha = 1$ yields the first, and $\alpha = \frac{1}{3}$ the second case, one might think that choosing the correct α in between would result in a CVP with $\ell = k$. Unfortunately, Algorithm **BoundedGL** does not have a monotone behaviour w.r.t. $\alpha \in (\frac{1}{3}, 1)$ in the sense that for two values $\frac{1}{3} < \alpha_1 < \alpha_2 < 1$, the case $\ell = k$ for α_1 does not imply $\ell = k$ for α_2 . Thus, even if we could prove the existence of an optimal value for α , we have no way to search for it.

4.2 Both-side Bounded Partition for k -connected Graphs

In this section, we give the algorithms to derive Theorem 7 by providing a both-side bounded approximate GL partition. For achieving a simultaneous lower and upper bounded partition, as a starting point, we apply Theorem 12 with $\alpha = \frac{1}{3}$ obtaining a lower and upper bounded packing $\mathcal{T} = \{T_1, \dots, T_k\}$ with $\frac{1}{3}w_i \leq w(T_i) \leq w_i$ for every $i \in [k]$. As long as \mathcal{T} is not a CVP of V yet, we transfer a subset of the remaining vertices $V \setminus V(\mathcal{T})$ through a path in an auxiliary graph to elements in \mathcal{T} , while making sure that for each i , $\frac{1}{3}w_i \leq w(T_i) \leq \max\{r, 3\}w(T_i)$. We call one such transfer a *transferring-iteration*. We define $\mathcal{T}^* := \{T_1, T_2, \dots, T_j\}$ where j is the smallest number such that $w(T_i) \geq w_i$ for $i \in [j]$ and $w(T_{j+1}) < w_{j+1}$. In case of $w_a = w_b$ and $w(T_a) \geq w_a$, but $w(T_b) < w_b$ we assume that $a < b$. Note that this is easily realizable by a relabeling of indices. Observe that $\mathcal{T}^* = \emptyset$ if $w(T_1) < w_1$. As a measure of progress,

we guarantee in each transferring-iteration that either the cardinality of \mathcal{T}^* increases, or the number of vertices in $V(\mathcal{T})$ increases. Also, the cardinality of \mathcal{T}^* is non-decreasing throughout the algorithm. Note that if $T_i \in \mathcal{T}^*$ for all i , then it follows that $w(T_i) = w_i$ for all i and moreover, \mathcal{T} is a CVP.

Let $\mathcal{T} = \{T_1, \dots, T_k\}$ be a connected packing of V in G with $w(T_i) \leq \max\{r, 3\}w_i$ for every $i \in [k]$. We use \mathcal{Q} to denote the vertex sets forming the connected components of $G[V \setminus V(\mathcal{T})]$. We define $\mathcal{T}^+ := \{T_i \in \mathcal{T} \mid w(T_i) \geq w_i\}$ and $\mathcal{T}^- := \mathcal{T} \setminus \mathcal{T}^+$. Note that $\mathcal{T}^* \subseteq \mathcal{T}^+$. Analogous to section 4.1, we define \mathcal{T}_a^+ as the set of $T_i \in \mathcal{T}^+$ that do *not* have a w_i -separator vertex and \mathcal{T}_b^+ to be the ones in \mathcal{T}^+ having a w_i -separator. For $T_i \in \mathcal{T}_b^+$, we use $s(T_i)$ to denote its w_i -separator (if there are multiple we fix one arbitrarily) and $C(T_i)$ to denote the vertex sets forming the connected components of $G[T_i \setminus \{s(T_i)\}]$. We say a vertex $v \in V$ or a vertex set $V' \subseteq V$ is \mathcal{T} -assigned if $v \in V(\mathcal{T})$ or $V' \subseteq V(\mathcal{T})$, respectively. That is, the set of \mathcal{T} -assigned vertices is $V(\mathcal{T})$ and $V(\mathcal{Q})$ is the set of *not* \mathcal{T} -assigned vertices. We say \mathcal{T} is *pack-satisfied* if $|\mathcal{T}| = k$, each $T_j \in \mathcal{T}$ is connected, $w(T_j) \in [\frac{1}{3}w_j, \max\{r, 3\}w_j]$, and the vertex sets in \mathcal{T} are pairwise disjoint.

We define the *transfer-graph* $H = (\mathcal{V}_H, E_H)$ as $\mathcal{V}_H := (\bigcup_{T \in \mathcal{T}_b^+} C(T)) \cup \mathcal{T}_a^+ \cup \mathcal{T}^- \cup \mathcal{Q}$ and $E_H := \{(V_1, V_2) \in \binom{\mathcal{V}_H}{2} \mid N_G(V_1) \cap V_2 \neq \emptyset\}$.

Algorithm DoubleBoundedGL

1. Apply Theorem 12 with $\alpha = \frac{1}{3}$ on G to obtain a connected packing $\mathcal{T} = \{T_1, \dots, T_k\}$ with $w(T_i) \geq \frac{1}{3}w_i$ for every $i \in [k]$.
2. While $\mathcal{Q} \neq \emptyset$:
 - a. Find a minimal path in H from \mathcal{Q} to \mathcal{T}^- . Let this path be $P_Q^{T_i}$ where $Q \in \mathcal{Q}$ and $T_i \in \mathcal{T}^-$.
// Note that all vertices in $P_Q^{T_i}$ except the start and end vertex are in $\mathcal{T}_a^+ \cup \bigcup_{T \in \mathcal{T}_b^+} C(T)$ by minimality of the path.
 - b. Execute the **TransferVertices** routine given below, which augments vertices through the path $P_Q^{T_i}$ such that \mathcal{T} stays pack-satisfied, and either $|\mathcal{T}^*|$ increases, or $|\mathcal{T}^*|$ remains the same and the number of \mathcal{T} -assigned vertices increases.

We need some more notations for describing the **TransferVertices** routine. For $\mathcal{V}'_H \subseteq \mathcal{V}_H$ and $\mathcal{T}' \subseteq \mathcal{T}$ we define $\mathcal{T}'(\mathcal{V}'_H)$ as the set $\{T_i \in \mathcal{T}' \mid V(T_i) \cap V(\mathcal{V}'_H) \neq \emptyset\}$. For $H' \subseteq H$ we define $\mathcal{T}'(H') := \mathcal{T}'(\mathcal{V}_H(H'))$, and $V(H') := V(\mathcal{V}_H(H'))$. With $|P_Q^{T_i}|$ we denote the length of the path $P_Q^{T_i}$, i.e. the number of edges in $P_Q^{T_i}$. We define P_Q^ℓ as the vertex with distance ℓ to Q in $P_Q^{T_i}$, where $P_Q^0 = Q$, and define $T(P_Q^\ell)$ for $\ell \in [|P_Q^{T_i}|]$ as the function which returns T_j with $P_Q^\ell \subseteq T_j$. For $\mathcal{T}' \subseteq \mathcal{T}$ we define $I(\mathcal{T}') := \{i \mid T_i \in \mathcal{T}'\}$.

The **TransferVertices** routine transfers vertices through the path $P_Q^{T_i}$. Our input is a pack-satisfied \mathcal{T} and a $P_Q^{T_i}$ path according to Step 2b in algorithm **DoubleBoundedGL**. By the minimality of the path $P_Q^{T_i}$, it is clear that $V_H(P_Q^{T_i}) \setminus \{Q, T_i\} \subseteq \mathcal{T}_a^+ \cup \bigcup_{T \in \mathcal{T}_b^+} C(T)$. That is, except for the destination T_i we run only through vertex sets from \mathcal{T}^+ in $\mathcal{T}(P_Q^{T_i})$. Roughly, our goal is to transfer vertices of $V(P_Q^{T_i} - T_i)$ to T_i , thereby changing the division of the vertex sets $\mathcal{T}(P_Q^{T_i})$ and preserving the vertex sets in \mathcal{T}^* .

We often need to do a *truncate* operation on sets T_j with $w(T_j) > \max\{r, 3\}w_j$. We mean by *truncate* T_j that we remove vertices from T_j until $w_j \leq w(T_j) \leq \max\{r, 3\}w_j$ such that T_j remains connected. This can be done by removing a non-separator vertex from T_j until the weight drops below $\max\{r, 3\}w(T_j)$. Note that any connected graph has at least one non-separator vertex. Since $w_{\max} \leq w_j$ we know that the weight does not decrease below w_j during the last deletion.

Algorithm TransferVertices

1. Initialize $X := Q$ and let $u = \min(I(\mathcal{T}^-))$.
2. For $\ell = 1$ to $|P_Q^{\mathcal{T}^i}|$ do:
 - a. Let $T_j = T(P_Q^\ell)$.
 - b. If $w(X) \geq w_u$: set $T_u = X$. Truncate T_u if necessary and terminate the algorithm.
 - c. If $w(X \cup T_j) \leq \max\{r, 3\}w_j$: update T_j to $X \cup T_j$ and terminate the algorithm.
 - d. If $T_j \notin \mathcal{T}^*$: Set $T'_j = T_j \cup X$, $T_j = T_u$ and $T_u = T'_j$. Truncate T_j and T_u if necessary and terminate the algorithm.
 - e. If $T_j \in \mathcal{T}_a^+$: divide $T_j \cup X$ into connected vertex sets V_1, V_2 with $w(V_1), w(V_2) \geq w_j$ using the construction given by Lemma 11. Set $T_j = V_1$ and $T_u = V_2$. Truncate T_j and T_u if necessary and terminate the algorithm.
 - f. We know $T_j \in \mathcal{T}_b^+ \cap \mathcal{T}^*$. Set $X = X \cup P_Q^\ell$ and remove P_Q^ℓ from T_j .

References

- 1 Curtis A Barefoot, Roger Entringer, and Henda Swart. Vulnerability in graphs a comparative survey. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 1:13–22, 1998.
- 2 Ronald Becker, Isabella Lari, Mario Lucertini, and Bruno Simeone. Max-min partitioning of grid graphs into connected components. *Networks: An International Journal*, 32(2):115–125, 1998.
- 3 Ronald Becker, Isabella Lari, Mario Lucertini, and Bruno Simeone. A polynomial-time algorithm for max-min partitioning of ladders. *Theory of Computing Systems*, 34(4):353–374, 2001.
- 4 Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. In *Algorithm Engineering*, pages 117–158. Springer, 2016.
- 5 Paolo M Camerini, Giulia Galbiati, and Francesco Maffioli. On the complexity of finding multi-constrained spanning trees. *Discrete Applied Mathematics*, 5(1):39–50, 1983.
- 6 Katrin Casel, Tobias Friedrich, Davis Issac, Aikaterini Niklanovits, and Ziena Zeif. Balanced crown decomposition for connectivity constraints. *arXiv preprint arXiv:2011.04528*, 2020.
- 7 L. Sunil Chandran, Yun Kuen Cheung, and Davis Issac. Spanning tree congestion and computation of generalized györi-lovász partition. In *45th International Colloquium on Automata, Languages, and Programming*, volume 107 of *LIPICs*, pages 32:1–32:14, 2018.
- 8 Frédéric Chataigner, Liliane Benning Salgado, and Yoshiko Wakabayashi. Approximation and inapproximability results on balanced connected partitions of graphs. *Discrete Mathematics and Theoretical Computer Science*, 9(1), 2007. URL: <http://dmcs.episciences.org/384>.
- 9 Guangting Chen, Yong Chen, Zhi-Zhong Chen, Guohui Lin, Tian Liu, and An Zhang. Approximation algorithms for the maximally balanced connected graph tripartition problem. *Journal of Combinatorial Optimization*, pages 1–21, 2020.
- 10 Jiangzhuo Chen, Robert D Kleinberg, László Lovász, Rajmohan Rajaraman, Ravi Sundaram, and Adrian Vetta. (almost) tight bounds and existence theorems for single-commodity confluent flows. *Journal of the ACM (JACM)*, 54(4):16, 2007.
- 11 Yong Chen, Zhi-Zhong Chen, Guohui Lin, Yao Xu, and An Zhang. Approximation algorithms for maximally balanced connected graph partition. In *International Conference on Combinatorial Optimization and Applications*, pages 130–141. Springer, 2019.
- 12 Janka Chlebíková. Approximating the maximally balanced connected partition problem in graphs. *Information Processing Letters*, 60(5):225–230, 1996.
- 13 An-Chiang Chu, Bang Ye Wu, and Kun-Mao Chao. A linear-time algorithm for finding an edge-partition with max-min ratio at most two. *Discrete Applied Mathematics*, 161(7-8):932–943, 2013.
- 14 Maria Chudnovsky and Paul Seymour. Claw-free graphs. I. orientable prismatic graphs. *Journal of Combinatorial Theory, Series B*, 97(6):867–903, 2007.

- 15 Maria Chudnovsky and Paul Seymour. Claw-free graphs. II. non-orientable prismatic graphs. *Journal of Combinatorial Theory, Series B*, 98(2):249–290, 2008.
- 16 Maria Chudnovsky and Paul Seymour. Claw-free graphs. III. circular interval graphs. *Journal of Combinatorial Theory, Series B*, 98(4):812–834, 2008.
- 17 Maria Chudnovsky and Paul Seymour. Claw-free graphs. IV. decomposition theorem. *Journal of Combinatorial Theory, Series B*, 98(5):839–938, 2008.
- 18 Maria Chudnovsky and Paul Seymour. Claw-free graphs. V. global structure. *Journal of Combinatorial Theory, Series B*, 98(6):1373–1410, 2008.
- 19 Maria Chudnovsky and Paul Seymour. Claw-free graphs VI. colouring. *Journal of Combinatorial Theory, Series B*, 100(6):560–572, 2010.
- 20 Maria Chudnovsky and Paul Seymour. Claw-free graphs. VII. quasi-line graphs. *Journal of Combinatorial Theory, Series B*, 102(6):1267–1294, 2012.
- 21 Maria Chudnovsky and Paul D Seymour. The structure of claw-free graphs. *Surveys in combinatorics*, 327:153–171, 2005.
- 22 Phillip EC Compeau, Pavel A Pevzner, and Glenn Tesler. How to apply de bruijn graphs to genome assembly. *Nature biotechnology*, 29(11):987–991, 2011.
- 23 Marek Cygan, Geevarghese Philip, Marcin Pilipczuk, Michał Pilipczuk, and Jakub Onufry Wojtaszczyk. Dominating set is fixed parameter tractable in claw-free graphs. *Theoretical Computer Science*, 412(50):6982–7000, 2011.
- 24 Yuri Faenza, Gianpaolo Oriolo, and Gautier Stauffer. Solving the weighted stable set problem in claw-free graphs via decomposition. *Journal of the ACM (JACM)*, 61(4):1–41, 2014.
- 25 Greg N. Frederickson. Optimal algorithms for tree partitioning. In *Proceedings of the Second Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms*, pages 168–177. ACM/SIAM, 1991. URL: <http://dl.acm.org/citation.cfm?id=127787.127822>.
- 26 E Gyori. On division of graphs to connected subgraphs, combinatorics. In *Colloquia Mathematica Societatis Janos Bolyai, 1976*, 1976.
- 27 Danny Hermelin, Matthias Mnich, and Erik Jan van Leeuwen. Parameterized complexity of induced graph matching on claw-free graphs. *Algorithmica*, 70(3):513–560, 2014.
- 28 Alexander Hoyer. *On the Independent Spanning Tree Conjectures and Related Problems*. PhD thesis, Georgia Institute of Technology, 2019.
- 29 Davis Issac. *On some covering, partition and connectivity problems in graphs*. PhD thesis, Saarländische Universitäts-und Landesbibliothek, 2019.
- 30 Takehiro Ito, Xiao Zhou, and Takao Nishizeki. Partitioning a graph of bounded tree-width to connected subgraphs of almost uniform size. *Journal of discrete algorithms*, 4(1):142–154, 2006.
- 31 Sukhamay Kundu and Jayadev Misra. A linear tree partitioning algorithm. *SIAM Journal on Computing*, 6(1):151–154, 1977.
- 32 László Lovász. A homology theory for spanning trees of a graph. *Acta Mathematica Academiae Scientiarum Hungarica*, 30(3-4):241–251, 1977.
- 33 Christian Löwenstein, Dieter Rautenbach, and Friedrich Regen. On spanning tree congestion. *Discrete mathematics*, 309(13):4653–4655, 2009.
- 34 Mario Lucertini, Yehoshua Perl, and Bruno Simeone. Most uniform path partitioning and its use in image processing. *Discrete Applied Mathematics*, 42(2-3):227–256, 1993.
- 35 Rolf H Möhring, Heiko Schilling, Birk Schütz, Dorothea Wagner, and Thomas Willhalm. Partitioning graphs to speedup dijkstra’s algorithm. *Journal of Experimental Algorithmics (JEA)*, 11:2–8, 2007.
- 36 Yehoshua Perl and Stephen R Schach. Max-min tree partitioning. *Journal of the ACM (JACM)*, 28(1):5–15, 1981.
- 37 Hitoshi Suzuki, Naomi Takahashi, and Takao Nishizeki. A linear algorithm for bipartition of biconnected graphs. *Information Processing Letters*, 33(5):227–231, 1990.

27:14 Connected k -Part. of k -Conn. & c -Claw-Free Graphs

- 38 Koichi Wada and Kimio Kawaguchi. Efficient algorithms for tripartitioning triconnected graphs and 3-edge-connected graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 132–143. Springer, 1993.
- 39 Xing Zhou, Huaimin Wang, Bo Ding, Tianjiang Hu, and Suning Shang. Balanced connected task allocations for multi-robot systems: An exact flow-based integer program and an approximate tree-based genetic algorithm. *Expert Systems with Applications*, 116:10–20, 2019.