

Parameterized Algorithms for Min-Max Multiway Cut and List Digraph Homomorphism^{*†‡}

Eunjung Kim[§] Christophe Paul[¶] Ignasi Sau[¶]

Dimitrios M. Thilikos^{¶||**}

Abstract

In this paper we design FPT-algorithms for two parameterized problems. The first is LIST DIGRAPH HOMOMORPHISM: given two digraphs G and H and a list of allowed vertices of H for every vertex of G , the question is whether there exists a homomorphism from G to H respecting the list constraints. The second problem is a variant of MULTIWAY CUT, namely MIN-MAX MULTIWAY CUT: given a graph G , a non-negative integer ℓ , and a set T of r terminals, the question is whether we can partition the vertices of G into r parts such that (a) each part contains one terminal and (b) there are at most ℓ edges with only one endpoint in this part. We parameterize LIST DIGRAPH HOMOMORPHISM by the number w of edges of G that are mapped to non-loop edges of H and we give a time $2^{O(\ell \cdot \log h + \ell^2 \cdot \log \ell)} \cdot n^4 \cdot \log n$ algorithm, where h is the order of the host graph H . We also prove that MIN-MAX MULTIWAY CUT can be solved in time $2^{O((\ell r)^2 \log \ell r)} \cdot n^4 \cdot \log n$. Our approach introduces a general problem, called LIST ALLOCATION, whose expressive power permits the design of parameterized reductions of both aforementioned problems to it. Then our results are based on an FPT-algorithm for the LIST ALLOCATION problem that is designed using a suitable adaptation of the *randomized contractions* technique (introduced by [Chitnis, Cygan, Hajiaghayi, Pilipczuk, and Pilipczuk, FOCS 2012]).

Keywords: Parameterized complexity; Fixed-Parameter Tractable algorithm; Multiway Cut, Digraph homomorphism.

*The third author was co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program “Education and Lifelong Learning” of the National Strategic Reference Framework (NSRF), Research Funding Program: ARISTEIA II.

†Emails: eunjungkim78@gmail.com, {Christophe.Paul, Ignasi.Sau}@lirmm.fr, and sedthilk@thilikos.info

‡An extended abstract of this work will appear in the *Proceedings of the 10th International Symposium on Parameterized and Exact Computation (IPEC), Patras, Greece, September 2015*.

§CNRS, LAMSADE, Paris, France.

¶AIGCo project-team, CNRS, LIRMM, Montpellier, France

||Department of Mathematics, University of Athens, Athens, Greece

**Computer Technology Institute & Press “Diophantus”, Patras, Greece

1 Introduction

The MULTIWAY CUT problem asks, given a graph G , a set of r terminals T , and a non-negative integer ℓ , whether it is possible to partition $V(G)$ into r parts such that each part contains exactly one of the terminals of T and there are at most ℓ edges between different parts (i.e., at most ℓ crossing edges). In the special case where $|T| = 2$, this gives the celebrated MINIMUM CUT problem, which is polynomially solvable [31]. In general, when there is no restriction on the number of terminals, the MULTIWAY CUT problem is NP-complete [8] and a lot of research has been devoted to the study of this problem and its generalizations, including several classic results on its polynomial approximability [3, 14, 17, 18, 24, 30].

More recently, special attention to the MULTIWAY CUT problem was given from the parameterized complexity point of view. The existence of an FPT-algorithm for MULTIWAY CUT (when parameterized by ℓ), i.e., an $f(\ell) \cdot n^{O(1)}$ -step algorithm, had been a long-standing open problem. This question was answered positively by Marx in [26] with the use of the *important separators technique* which was also used for the design of FPT-algorithms for several other problems such as DIRECTED MULTIWAY CUT [4], VERTEX MULTICUT, and EDGE MULTICUT [28]. This technique has been extended to the powerful framework of *randomized contractions technique*, introduced in [5]. This made it possible to design FPT-algorithms for several other problems such as UNIQUE LABEL COVER, STEINER CUT, EDGE/VERTEX MULTIWAY CUT-UNCUT. We stress that this technique is quite versatile.

In this paper we use it in order to design FPT-algorithms for parameterizations of two problems that do not seem to be directly related to each other: the MIN-MAX-MULTIWAY CUT problem [32] and the LIST DIGRAPH HOMOMORPHISM problem.

1.1 Min-Max-Multiway Cut

In the MULTIWAY CUT problem the parameter ℓ bounds the total number of crossing edges (i.e., edges with endpoints in different parts). Svitkina and Tardos [32] considered a “min-max” variant of this problem, namely the MIN-MAX-MULTIWAY CUT, where ℓ bounds the maximum number of outgoing edges of the parts¹. In [32], it was proved that MIN-MAX-MULTIWAY CUT is NP-complete even when the number of terminals is $r = 4$. As a consequence of the results in [32] and [29], MIN-MAX-MULTIWAY CUT admits an $O(\log^2 n)$ -approximation algorithm. This was improved recently in [1] to a $O((\log n \cdot \log r)^{1/2})$ -approximation algorithm.

To our knowledge, nothing is known about the parameterized complexity of this problem. We prove the following.

¹Notice that under this viewpoint MULTIWAY CUT can be seen as MIN-SUM-MULTIWAY CUT.

Theorem 1. *There exists an algorithm that solves the MIN-MAX-MULTIWAY CUT problem in $2^{O((r\ell)^2 \log r\ell)} \cdot n^4 \cdot \log n$ steps, i.e., MIN-MAX-MULTIWAY CUT belongs to FPT when parameterized by both r and ℓ .*

(Throughout the paper, we use $n = |V(G)|$ when we refer to the number of vertices of the graph G in the instance of the considered problem.)

1.2 List Digraph Homomorphism

Given two directed graphs G and H , an H -homomorphism of G is a mapping $\chi : V(G) \rightarrow V(H)$ such that if (x, y) is an arc of G , then $(\chi(x), \chi(y))$ is also an arc in H . In the LIST DIGRAPH HOMOMORPHISM problem, we are given two graphs G and H and a list function $\lambda : V(G) \rightarrow 2^{V(H)}$ and we ask whether G has a H -homomorphism such that for every vertex v of G , $\chi(v) \in \lambda(v)$. Graph and digraph homomorphisms have been extensively studied both from the combinatorial and the algorithmic point of view (see e.g., [2, 13, 15, 16, 21]).

Especially for the LIST DIGRAPH HOMOMORPHISM problem, a dichotomy characterizing the instantiations of H for which the problem is hard was given in [22] (see also [12]). Notice that the standard parameterization of LIST DIGRAPH HOMOMORPHISM by the size of the graph H is para-NP-complete as it yields the 3-COLORING problem when G is restricted to be a simple graph and $H = K_3$. A more promising parameterization of LIST HOMOMORPHISM (for undirected graphs) has been introduced in [10], where the parameter is a bound on the number of pre-images of some prescribed set of vertices of H (see also [9, 11, 27]). Another parameterization, again for the undirected case, was introduced in [6], where the parameter is the number of vertices to be removed from the graph G so that the remaining graph has a list H -homomorphism.

We introduce a new parameterization of LIST DIGRAPH HOMOMORPHISM where the parameter is, apart from $h = |V(H)|$, the number of “crossing edges”, i.e., the edges of G whose endpoints are mapped to different vertices of H . For this, we enhance the input with an integer ℓ and ask for a list digraph homomorphism with at most ℓ crossing edges. Clearly, when $\ell = |E(G)|$, this yields the original problem. We call the new problem BOUNDED LIST DIGRAPH HOMOMORPHISM (in short, BLDH). Notice that the fact that LIST DIGRAPH HOMOMORPHISM is NP-complete even when $h = 3$, implies that BLDH is para-NP-complete when parameterized only by h . The input of BLDH is a quadruple (G, H, λ, ℓ) where G is the guest graph, H is the host graph, $\lambda : V(G) \rightarrow 2^{V(H)}$ is the list function and ℓ is a non-negative integer. Our next step is to observe that BLDH is W[1]-hard, when parameterized only by ℓ . To see this consider an input (G, k) of the CLIQUE problem and construct the input $(K, \tilde{G}, \lambda, \ell)$ where K is a the complete digraph on k vertices, \tilde{G} is the digraph obtained by G by replacing each edge by two

opposite direction arcs between the same endpoints, $\lambda = \{(v, V(G)) \mid v \in V(K)\}$, and $\ell = k(k-1)$. Notice that (G, k) is a YES-instance of CLIQUE iff $(K, \bar{G}, \lambda, \ell)$ is a YES-instance of BLDH.

We conclude that when BLDH is parameterized by ℓ or h only, then one may not expect it to be fixed parameter tractable. This means that the parameterization of BLDH by h and ℓ is meaningful to consider. Our result is the following.

Theorem 2. *There exists an algorithm that solves the BOUNDED LIST DIGRAPH HOMOMORPHISM problem in $2^{O(\ell \cdot \log h + \ell^2 \cdot \log \ell)} \cdot n^4 \cdot \log n$ steps, i.e., BOUNDED LIST DIGRAPH HOMOMORPHISM belongs to FPT when parameterized by the number ℓ of crossing edges and the number h of vertices of H .*

1.3 List Allocation

In order to prove Theorems 1 and 2, we prove that both problems are Turing FPT-reducible² to a single new problem that we call LIST ALLOCATION (in short, LA).

The LIST ALLOCATION problem is defined as follows: We are given a graph G and a set of r “boxes” indexed by numbers from $\{1, \dots, r\}$. Each vertex v of G is accompanied with a list $\lambda(v)$ of indices corresponding to the boxes where it is allowed to be allocated. Moreover, there is a weight function α assigning to every pair of different boxes a non-negative integer. The question is whether there is a way to place each of the vertices of G into some box of its list such that, for any two different boxes i and j , the number of crossing edges between them is exactly $\alpha(i, j)$.

As we easily see in Subsection 2.3, LIST ALLOCATION is NP-complete, even when $r = 2$. Throughout this paper, we parameterize the LIST ALLOCATION problem by the total number w of “crossing edges” between different boxes, i.e., $w = \sum_{1 \leq i < j \leq r} \alpha(i, j)$.

Our main result is that this parameterization of LA is in FPT.

Theorem 3. *There exists an algorithm that, given as input an instance $I = (G, r, \lambda, \alpha)$ of LIST ALLOCATION, returns an answer to this problem in $2^{O(w^2 \cdot \log w)} \cdot n^4 \cdot \log n$ steps, where $w = \sum_{1 \leq i < j \leq r} \alpha(i, j)$.*

To witness the expressive power of LIST ALLOCATION, let us first exemplify why MULTIWAY CUT, parameterized by w , is T-FPT-reducible to LIST ALLOCATION.

²Let **A** and **B** be two parameterized problems. We say that a parameterized problem **A** is Turing FPT-reducible to **B** when the existence of an FPT-algorithm for **B** implies the existence of an FPT-algorithm for **A**. (For brevity, in this paper, we write “T-FPT” instead of “Turing FPT”.)

Given an instance of MULTIWAY CUT, we first discard from its graph all the connected components that have at most 1 terminal. Clearly, this gives an equivalent instance $(G, T = \{t_1, \dots, t_r\}, w)$ where $r \leq w + 1$.

Next, we consider the set \mathcal{A} containing every weight function α such that $\sum_{1 \leq i < j \leq r} \alpha(i, j) \leq w$. Let also $\lambda : V(G) \rightarrow 2^{[r]}$ be the list function such that if $v = t_i \in T$, then $\lambda(v) = \{i\}$, otherwise $\lambda(v) = \{1, \dots, r\}$. It is easy to verify that (G, T, w) is a YES-instance of MULTIWAY CUT if and only if there exists some $\alpha \in \mathcal{A}$ such that (G, r, λ, α) is a YES-instance of LIST ALLOCATION. This yields the claimed reduction, as $|\mathcal{A}|$ is clearly bounded by some function of w . This reduction to the LIST ALLOCATION problem turns out to be quite flexible and, as we will see in Subsection 3.1 (Theorem 4), it can easily be adapted to a T-FPT-reduction of MIN-MAX-MULTIWAY CUT to LIST ALLOCATION. The reduction of BOUNDED LIST DIGRAPH HOMOMORPHISM to LIST ALLOCATION is more complicated and is described in Subsection 3.2 (Theorem 6). Theorem 3, together with the aforementioned reductions, yields Theorems 1 and 2.

2 Preliminaries and the definition of LIST ALLOCATION

2.1 Functions and allocations

We use the notation $\log(n)$ to denote $\lceil \log_2(n) \rceil$ for $n \in \mathbb{Z}_{\geq 1}$ and we agree that $\log(0) = 1$. Given a non-negative integer n , we denote by $[n]$ the set of all positive integers no bigger than n . Given a finite set A and an integer $s \in \mathbb{Z}_{\geq 0}$, we denote by $\binom{A}{s}$ (resp. $\binom{A}{\leq s}$) the set of all subsets of A with exactly (resp. at most) s elements. Given a function $f : A \rightarrow \mathbb{Z}_{\geq 0}$ we define $\sum f = \sum_{x \in A} f(x)$. An r -allocation of a set S is an r -tuple $\mathcal{V} = (V_1, \dots, V_r)$ of, possibly empty, sets that are pairwise disjoint and whose union is the set S . We refer to the elements of \mathcal{V} as the *parts* of \mathcal{V} and we denote by $\mathcal{V}^{(i)}$ the i -th part of \mathcal{V} , i.e., $\mathcal{V}^{(i)} = V_i$.

2.2 Definitions about graphs

In this paper, when giving the running time of an algorithm of some problem whose instance involves a graph G , we agree that $n = |V(G)|$ and $m = |E(G)|$.

All graphs in this paper are loopless and they may have multiple edges. The only exception to this agreement is in Subsection 3.2 where we also allow loops. If G is a graph and X, Y are two disjoint vertex subsets of $V(G)$, we define $\delta_G(X, Y)$ as the set of edges with one endpoint in X and the other in Y . Given a graph G , denote by $\mathcal{C}(G)$ the collection of all connected components of G .

2.3 The list allocation problem

We define the problem LA as follows.

<p>LIST ALLOCATION (LA)</p> <p><i>Input:</i> A tuple $I = (G, r, \lambda, \alpha)$ where G is a graph, $r \in \mathbb{Z}_{\geq 1}$, $\lambda : V(G) \rightarrow 2^{[r]}$, and $\alpha : \binom{[r]}{2} \rightarrow \mathbb{Z}_{\geq 0}$.</p> <p><i>Output:</i> An r-allocation \mathcal{V} of $V(G)$ such that</p> <ol style="list-style-type: none"> 1. $\forall \{i, j\} \in \binom{[r]}{2}, \delta_G(\mathcal{V}^{(i)}, \mathcal{V}^{(j)}) = \alpha(i, j)$ and 2. $\forall v \in V(G), \forall i \in [r],$ if $v \in \mathcal{V}^{(i)}$ then $i \in \lambda(v)$, <p>or a correct report that no such r-allocation exists.</p>

For simplicity, in the above definition we write $\alpha(\{i, j\})$ as $\alpha(i, j)$ and we agree that $\alpha(i, j) = \alpha(j, i)$. Also, given an instance I of LA, we denote³ $w(I) = \sum \alpha$. We will also use w instead of $w(I)$ when it is clear what is the instance we are working with. We assume that the multiplicity of each edge in G does not exceed w as, if this happens, then reducing it to w creates an equivalent instance of the problem.

In the definition of LA each vertex v of G carries a *list* $\lambda(v)$ indicating the parts where v can be possibly allocated. Moreover, α is a function assigning weights to pairs of parts in \mathcal{V} . The weights defined by α prescribe the precise number of crossing edges between distinct parts of \mathcal{V} .

Notice that LA is an NP-hard problem by a simple reduction from the MAX CUT problem, asking whether, for an input graph G and some $w \in \mathbb{Z}_{\geq 0}$, whether there is a partition V_1, V_2 of $V(G)$ such that there are *exactly*⁴ w edges each with endpoints in both V_1 and V_2 . Indeed, given an instance $I = (G, w)$ of MAX CUT, construct the instance $I' = (G, 2, \lambda, \alpha)$ where $\lambda(v) = \{1, 2\}$ for every $v \in V(G)$ and $\alpha(1, 2) = w$. Note also that when $r = 2$, LA is polynomially solvable on planar graphs as it directly reduces to PLANAR MAX CUT that is polynomially solvable [20].

3 Main reductions

In this section we formally define MIN-MAX-MULTIWAY CUT and LIST DIGRAPH HOMOMORPHISM and we reduce them to LIST ALLOCATION.

³Given a function $\tau : A \rightarrow \mathbb{Z}_{\geq 0}$, we denote $\sum \tau = \sum_{x \in A} \tau(x)$.

⁴It is straightforward to see that the standard reduction from NAE-3-SAT also works when the question of MAX CUT asks for exactly w crossing edges instead of at least w crossing edges.

3.1 Min-Max-Multiway Cut

The MIN-MAX-MULTIWAY CUT problem is formally defined as follows:

<p>MIN-MAX-MULTIWAY CUT</p> <p><i>Input:</i> A tuple $I = (G, \ell, r, T)$ where G is an undirected graph, $\ell, r \in \mathbb{Z}_{\geq 0}$, and $T \subseteq V(G)$ with $T = r$.</p> <p><i>Output:</i> A partition $\{\mathcal{P}_1, \dots, \mathcal{P}_r\}$ of $V(G)$ such that for every $i \in [r]$, it holds that $\mathcal{P}_i \cap T = 1$ and $\delta_G(\mathcal{P}_i, V(G) \setminus \mathcal{P}_i) \leq \ell$, or a correct report that no such partition exists.</p>
--

Similarly to the case of LA, we assume that the multiplicity of each edge in G does not exceed ℓ .

Theorem 4. *If there is an algorithm that solves LA in $T(n, w(I))$ steps, then there exists an algorithm that solves MIN-MAX-MULTIWAY CUT in $2^{O(r \cdot \min\{\ell \cdot \log r, r \cdot \log \ell\})}$ steps.*

Proof. Given an input $I = (G, \ell, r, T)$ of MIN-MAX-MULTIWAY CUT, we fix (arbitrarily) a bijection $\mu : V(T) \rightarrow [r]$ and we define $\lambda : V(G) \rightarrow 2^{[r]}$ such that

$$\lambda(x) = \begin{cases} [r] & \text{if } x \in V(G) \setminus T \\ \{\mu(x)\} & \text{if } x \in T. \end{cases}$$

We now consider the family $\mathcal{U}(I)$ of instances of LA containing one element $I' = (G, r, \lambda, \alpha)$ for each choice of function $\alpha : \binom{[r]}{2} \rightarrow \mathbb{Z}_{\geq 0}$ satisfying

$$\forall i \in [r], \sum_{j \in [r] \setminus i} \alpha(i, j) \leq \ell.$$

Notice that I is a YES-instance of MIN-MAX-MULTIWAY CUT if and only if there exists some $I' \in \mathcal{U}(I)$ that is a YES-instance of LA. As $|\mathcal{U}(I)| = 2^{O(r \cdot \min\{\ell \cdot \log r, r \cdot \log \ell\})}$ and for each $I' \in \mathcal{U}(I)$ it holds that $w(I') = O(r\ell)$, the result follows. \square

3.2 List Digraph Homomorphism

Let G and H be directed graphs where G is simple and H may have loops but not multiple directed edges. A (directed) edge in the digraph G from the vertex x to the vertex y is denoted by (x, y) . Let also $\lambda : V(G) \rightarrow 2^{V(H)}$. We denote by $E_1(H)$ the loops of H and by $E_2(H)$ the edges of H between distinct vertices. An λ -list H -homomorphism of G is a function $\chi : V(G) \rightarrow V(H)$ such that

- $\chi(v) \in \lambda(v)$ for every $v \in V(G)$, and

- $(\chi(u), \chi(v)) \in E(H)$ for every $(u, v) \in E(G)$.

Given a list H -homomorphism χ of G and an edge $e = (a, b) \in E_2(H)$ we define

$$C(e) = \{(u, v) \in E(G) \mid \chi(u) = a \text{ and } \chi(v) = b\}.$$

BOUNDED LIST DIGRAPH HOMOMORPHISM is formally defined as follows.

BOUNDED LIST DIGRAPH HOMOMORPHISM (BLDH)

Input: A tuple $I = (G, H, \lambda, \ell)$ where G and H are digraphs, $\lambda : V(G) \rightarrow 2^{V(H)}$, and $\ell \in \mathbb{N}_{\geq 0}$.

Output: A λ -list H -homomorphism of G where $\sum_{e \in E_2(H)} |C(e)| \leq \ell$ or a correct report that no such homomorphism exists.

We now define the following more general problem.

ARC-SPECIFIED LIST DIGRAPH HOMOMORPHISM (ASLDH)

Input: A tuple $I = (G, H, \lambda, \alpha)$ where G and H are digraphs, $\lambda : V(G) \rightarrow 2^{V(H)}$, and $\alpha : E_2(H) \rightarrow \mathbb{Z}_{\geq 0}$.

Output: A λ -list H -homomorphism χ of G such that $\forall_{e \in E_2(H)} |C(e)| = \alpha(e)$ or a correct report that no such λ -list H -homomorphism exists.

Given an instance $I = (G, H, \lambda, \alpha)$ of ASLDH we define $d(I) = \sum \alpha$. As we already did for the cases of LA and MIN-MAX-MULTIWAY CUT, we assume that the multiplicity of the edges of the instance of BLDH (resp. ASLDH) does not exceed ℓ (resp. $d(I)$).

In the next sections we will prove that there exists an FPT-algorithm for ASLDH, when parameterized by *both* $h = |V(H)|$ and $d = d(I)$. This fact together with the following result yields Theorem 2.

Theorem 5. *If there is an algorithm that solves ASLDH in $T(n, d(I))$ steps, then there exists an algorithm that solves BLDH in $2^{O(\ell \log h)} \cdot T(n, \ell)$ steps where $h = |V(H)|$.*

Proof. Given an instance $I = (G, H, \lambda, \ell)$ of BLDH we set $\mathcal{U}(I) = \{(G, H, \lambda, \alpha) \mid \sum \alpha \leq \ell\}$ and we observe that I is a YES-instance of BLDH if and only if some $I' \in \mathcal{U}(I)$ is a YES-instance of ASLDH. The lemma follows as $|\mathcal{U}(I)| = 2^{O(\ell \log h)}$ and $d(I') \leq \ell$. \square

3.3 A sparsifier for ASLDH

In order to prove that ASLDH admits an FPT-algorithm when parameterized by both $h = |V(H)|$ and $d = d(I)$, we will give a Turing-FPT reduction of ASLDH to LA in Subsection 3.4. The latter problem can be solved by an FPT-algorithm due to the result of Section 4. The reduction of Subsection 3.4 receives an instance (G, H, λ, α) of ASLDH and returns an equivalent instance $(G', r, \lambda', \alpha')$ of LA where $|V(G')| = O(|E(G)|)$ which is $O(w \cdot |V(G)|^2)$, in general. In order to avoid this blow-up in the polynomial running time of our final FPT-algorithm, we give a way to transform the instances of ASLDH to equivalent instances of the same problem whose graphs are sparse. This “sparsification” procedure is described below.

A graph is *d-edge connected* if it has at least two vertices and for every two vertices there are d edge disjoint paths between them. We use the following result from [25].

Proposition 1. *For every $d \in \mathbb{Z}_{\geq 1}$, every graph G where $|E(G)| \geq d \cdot (|V(G)| - 1)$ contains a d -edge connected subgraph.*

We need first the following known result. For completeness, we provide the proof.

Lemma 1. *Let G be a d -edge connected graph and let $S = (s_1, \dots, s_d)$ and $T = (t_1, \dots, t_d)$ be two orderings of vertices of $V(G)$, possibly with repetitions. Then, there exists a bijection $\sigma : [d] \rightarrow [d]$ and a collection \mathcal{C} of d pairwise edge-disjoint paths such that for each $i \in [d]$, s_i and $t_{\sigma(i)}$ are the endpoints of some path in \mathcal{C} .*

Proof. We add in G two new vertices s and t and connect s with each vertex in S and t with each vertex in T such that the multiplicity of each edge $\{s, x\}$ is equal to the number of times x appears in S and multiplicity of each edge $\{t, x\}$ is equal to the number of times x appears in T . We observe that there are d -edge-disjoint paths from s to t . To see this, suppose that a set of fewer than d edges in $E(G')$ disconnects s and t . This means that removal of fewer than d edges of $E(G)$ disconnects s_i and t_j for some $i, j \in [d]$, contradicting d -edge-connectivity of G . Hence, we can find d edge-disjoint paths between s and t by Menger’s Theorem. Removing s and t from these paths yields d edge-disjoint paths between S and T having the desired property. \square

The following Lemma is based on Lemma 1.

Lemma 2. *Let G be a graph and let $\mathcal{C} = \{C_1, \dots, C_r\}$ be a collection of vertex disjoint connected subgraphs of G . Let also G' be the graph obtained if we contract in G all edges in the graphs in \mathcal{C} . If G' is d -edge connected and each graph in \mathcal{C} is d -edge connected or a single vertex, then G contains a subgraph that is d -edge connected.*

Proof. Let H be the subgraph of G induced by the vertices in the graphs in \mathcal{C} . Given a vertex $v \in V(H)$, we denote by C_v the graph in \mathcal{C} that is either v itself or is contracted in G to create v in H . We prove that for every two vertices s and t in $V(H)$ there are d -edge-disjoint paths between them. This follows easily in the case where both s and t belong in the same $C_v \in \mathcal{C}$ because of the d -edge connectivity of C_v . Assume now that $x \in C^{(s)}$ and $y \in C^{(t)}$ where $C^{(s)}$ and $C^{(t)}$ are different graphs in \mathcal{C} . Let also v_s and v_t be vertices of G' such that $C_{v_s} = C^{(s)}$ and $C_{v_t} = C^{(t)}$. As G' is d -edge connected, there is a collection $\mathcal{P} = \{P_1, \dots, P_d\}$ of d edge-disjoint paths in G' from v_s to v_t . We direct all these paths from v_s to v_t and we set $W = \bigcup_{i \in [d]} P_i$. Let $v \in W$ and let E_v be the set of edges in G' incident to v . Notice that E_v has a partition $\{E_v^1, \dots, E_v^d\}$ such that E_v^i are the edges of P_i that are incident to v . Clearly, each $E_{v_s}^i$ has only one edge and the same holds for each $E_{v_t}^i$. Moreover, each E_v^i with $v \notin \{v_s, v_t\}$ has cardinality two. We enhance the notation of the sets E_v^i as follows: if $E_v^i = \{e\}$ and $v = v_s$ then we write $E_v^i = (s, e)$. If $E_v^i = \{e\}$ and $v = v_t$ then we write $E_v^i = (e, t)$. If $v \notin \{v_s, v_t\}$ and $E_v^i = \{e, e'\}$ such that e is ingoing to v in P_i and e' is outgoing to v in P_i then we write $E_v^i = (e, e')$. We now define the pair \mathbf{p}_i^v as follows: if $E_v^i = (s, e)$ and y is the endpoint the edge e in G' that belongs in C_s then we set $\mathbf{p}_i^v = (s, y)$, if $E_v^i = (e, t)$ and y is the endpoint the edge e in G' that belongs in C_t , then we set $\mathbf{p}_i^v = (y, t)$, and if $E_v^i = (e, e')$, then y and y' are the endpoint the edges e and e' respectively that belong in C_v , then $\mathbf{p}_i^v = (y, y')$. For each $v \in W$ we create two orderings $S_v = (s_1^v, \dots, s_d^v)$ and $T_v = (t_1^v, \dots, t_d^v)$ of vertices in C_v such that $(s_i^v, t_i^v) = \mathbf{p}_i^v$ for every $i \in [d]$. For each v , we apply Lemma 1 and obtain a collection \mathcal{P}_v of edge-disjoint paths between the vertices of S_v and the vertices of T_v . It is now easy to observe that the subgraph of H consisting of the edges in the paths in \mathcal{P} (that are also edges of G') and the edges of the paths in \mathcal{P}_v for every $v \in W$ is the union of d edge-disjoint paths in H between s and t . \square

Given a graph H and a positive integer d , we say that a subgraph H of G is a *d -edge connected core of G* if every connected component of H is d -edge connected and, among all such subgraphs of G , H has maximum number of edges. The proof of the next lemma uses Proposition 1.

Lemma 3. *For every $d \in \mathbb{Z}_{>0}$, every graph G with $m \geq d \cdot (n - 1)$ contains a unique d -edge connected core that can be found in $O(d \cdot n^4)$ steps.*

Proof. The claimed d -edge connected core exists because of Proposition 1. Also, it is unique because if there are two d -edge connected cores J_1 and J_2 , then it can be easily checked that the graph $J_1 \cup J_2$ is also a d -edge connected core of G . The algorithm repetitively removes from G edges of min-cuts of size at most $d - 1$ in its connected components (each can be found in $O(d \cdot n^3)$ steps according to [31])

until this is not possible anymore (isolated vertices, when appearing during this procedure, are removed).

Note that the total number of steps of this procedure is bounded by the running time of the algorithm in [31] times the number of connected components of the resulting graph. This justifies the claimed running time. Let J be the d -edge connected core of G . Notice that none of the edges of J will be deleted by this procedure. Indeed, assuming the opposite, let G' be the graph where for the first time a cut (V_1, V_2) is found where the set F of crossing edges contains some edge $e = \{x, y\}$ in J . Let also C be the connected component of G' containing this cut and let C_J be a connected component of J that is a subgraph of C containing e .

Notice that x and y belong to different connected components of $C \setminus F$ and therefore also to different connected components of $C_J \setminus F$, contradicting the fact that C_J is d -edge connected. We just proved that the output of the algorithm will be a subgraph of J . Notice also that each connected component of this output is d -edge connected. By the maximality of J , this output is necessarily J . \square

Lemma 4. *There is an $O(d(I) \cdot n^4)$ -step algorithm that given in instance $I = (G, H, \lambda, \alpha)$ of ASLDH, outputs an equivalent instance $I' = (G', H, \lambda', \alpha)$ of the same problem where $|E(G')| = O(d(I) \cdot |V(G')|)$.*

Proof. Let \tilde{G} be the underlying graph of G (multiplicities of edges of opposite direction are summed up) and $d = d(I)$. If \tilde{G} does not contains a $(d + 1)$ -edge connected core, then, from Proposition 1, $|E(G)| = O(d \cdot |V(G)|)$.

Suppose now that \tilde{G} has a $(d + 1)$ -edge connected core J that, from Lemma 3, can be found in $O(d \cdot |V(G)|^4)$ steps. We create a new graph G' as follows: for each $C \in \mathcal{C}(J)$ we contract all vertices of C to a single vertex v_C and we update λ to λ' so that if $x \notin \{v_C \mid C \in \mathcal{C}(J)\}$, then $\lambda'(x) = \lambda(x)$ and if $x = v_C$, then $\lambda'(x) = \cap_{y \in V(C)} \lambda(y)$. We claim that $I' = (G', H, \lambda', \alpha)$ is an equivalent instance of ASLDH. Indeed, this is based on the fact that, given a λ -list H -homomorphism χ of G and a connected component C of J , all vertices of J should be the preimages via χ of the same vertex of H . To verify this fact, just observe that, if this is not the case, then the removal of the $\leq d$ crossing edges from C (i.e., edges with endpoints mapped to different vertices of H) will disconnect C , a contradiction to the $(d + 1)$ -edge-connectivity of C .

It now remains to prove that $|E(G')| = O(d \cdot |V(G')|)$. If $|E(G')| \geq (d + 1) \cdot (|V(G')| - 1)$, then, again from Proposition 1, G' contains a $(d + 1)$ -edge connected subgraph. This, because of Lemma 2, implies that G contains a subgraph that is $(d + 1)$ -edge connected and has more edges than J , a contradiction. \square

3.4 A reduction of ASLDH to LA

Given the results of the previous section we are now in position to prove the following.

Theorem 6. *If there is an algorithm that solves LA in $T(n, w(I))$ steps, then there exists an algorithm that solves ASLDH in $T(O(d(I) \cdot n), O(d(I))) + O(d(I) \cdot n^4)$ steps.*

Proof. Let $I = (G, H, \lambda, \alpha)$ be an instance of ASLDH. Using the algorithm of Lemma 4, we may assume that $|E(G)| = O(d(I) \cdot |V(G)|)$. We then use I to generate an instance $I' = (G', r, \lambda', \alpha')$ of LA, as follows:

- $G' = (V', E')$, where
 - $V' = V \cup V_F \cup V_L$, where $V = V(G)$, $V_F = \{f_{uv} \mid (u, v) \in E(G)\}$, and $V_L = \{\ell_{uv} \mid (u, v) \in E(G)\}$ and
 - $E' = E \cup E_F \cup E_L$, where $E = \{\{f_{uv}, \ell_{uv}\} \mid (u, v) \in E(G)\}$, $E_F = \{\{u, f_{uv}\} \mid (u, v) \in E(G)\}$, $E_L = \{\{\ell_{uv}, v\} \mid (u, v) \in E(G)\}$.

- $r = |V(H)| + 2 \cdot |E_2(H)|$ and $\sigma : V(\tilde{H}) \rightarrow [r]$ is a bijection where \tilde{H} is the graph obtained from H by subdividing twice each of its arcs that are not loops. For each arc $(x, y) \in E_2(H)$, we denote its corresponding path in \tilde{H} as P_{xy} , where $V(P_{xy}) = \{x, \tilde{f}_{xy}, \tilde{\ell}_{xy}, y\}$.

- $\lambda' : V(G') \rightarrow [r]$ such that

$$\lambda'(w) = \begin{cases} \{\sigma(x) \mid x \in \lambda(w)\} & \text{if } w \in V \\ \{\sigma(\tilde{f}_{xy}) \mid x \in \lambda(u) \wedge y \in \lambda(v) \wedge x \neq y\} \cup \\ \{\sigma(x) \mid x \in \lambda(u) \cap \lambda(v) \wedge (x, x) \in E_1(H)\} & \text{if } w = f_{uv} \in V_F \\ \{\sigma(\tilde{\ell}_{xy}) \mid x \in \lambda(u) \wedge y \in \lambda(v) \wedge x \neq y\} \cup \\ \{\sigma(x) \mid x \in \lambda(u) \cap \lambda(v) \wedge (x, x) \in E_1(H)\} & \text{if } w = \ell_{uv} \in V_L. \end{cases}$$

- $\alpha' : \binom{[r]}{2} \rightarrow \mathbb{Z}_{\geq 0}$ such that

$$\alpha'(i, j) = \begin{cases} \alpha(x, y) & \text{if there exists some } (x, y) \in E_2(H) \text{ such that} \\ & (i, j) \in \{(\sigma(x), \sigma(\tilde{f}_{xy})), (\sigma(\tilde{f}_{xy}), \sigma(\tilde{\ell}_{xy})), (\sigma(\tilde{\ell}_{xy}), \sigma(y))\} \\ 0 & \text{otherwise.} \end{cases}$$

Let $\chi : V(G) \rightarrow V(H)$ be a λ -list H -homomorphism of G where $\forall_{e \in E_2(H)} |C(e)| = \alpha(e)$. We construct an r -allocation \mathcal{V} of $V(G')$ as follows:

- for every $u \in V = V(G)$, u belongs to the part \mathcal{V}^i , where $i = \sigma(\chi(u))$
- for every $f_{uv} \in V_F$, f_{uv} belongs to the part \mathcal{V}^i , where

$$i = \begin{cases} \sigma(\chi(u)) & \text{if } \chi(u) = \chi(v) \\ \sigma(\tilde{f}_{xy}) & \text{if } x = \chi(u) \neq y = \chi(v) \end{cases}$$

- for every $\ell_{uv} \in V_L$, ℓ_{uv} belongs to the part \mathcal{V}^i , where

$$i = \begin{cases} \sigma(\chi(u)) & \text{if } \chi(u) = \chi(v) \\ \sigma(\tilde{\ell}_{xy}) & \text{if } x = \chi(u) \neq y = \chi(v) \end{cases}$$

It is easy to verify that \mathcal{V} is a solution for I' .

Now consider a solution \mathcal{V} for I' . From \mathcal{V} , we define a mapping $\chi : V(G) \rightarrow V(H)$ so that for every $u \in V$, we have that $\chi(u) = \sigma^{-1}(i)$ if and only if $u \in \mathcal{V}^i$. We claim that χ is a λ -list H -homomorphism of G where $\forall_{e \in E_2(H)} |C(e)| = \alpha(e)$. For this, we investigate χ upon two conditions: firstly, we verify that χ is a λ -list H -homomorphism, and secondly that $\forall_{e \in E_2(H)} |C(e)| = \alpha(e)$.

Let us prove that χ is a λ -list H -homomorphism. To see that $\chi(u) \in \lambda(u)$ for every $u \in V(G)$, let u be in the i -th part of \mathcal{V} . Since $i \in \lambda'(u)$, the construction of λ' implies that $\sigma^{-1}(i) \in \lambda(u)$, and thus $\chi(u) \in \lambda(u)$. To see that χ is an H -homomorphism, for an arbitrary edge $(u, v) \in E(G)$ we shall show that $(\chi(u), \chi(v)) \in E_1(H) \cup E_2(H)$. Let u and v respectively belong to $\sigma(x)$ -th and $\sigma(y)$ -th parts of \mathcal{V} , for some $x, y \in V(\tilde{H})$. Note that $x \in \lambda(u) \subseteq V(H)$ and $y \in \lambda(v) \subseteq V(H)$. There are two possibilities: $x \neq y$ or $x = y$.

Case 1: $x \neq y$. Since σ is a bijection, this means $\sigma(x) \neq \sigma(y)$. From the way we construct α' , the vertices f_{uv} and ℓ_{uv} can be only allocated into the $\sigma(\tilde{f}_{xy})$ -th part and the $\sigma(\tilde{\ell}_{xy})$ -th part, respectively, in the solution \mathcal{V} . Furthermore, the construction of α' also implies $(x, y) \in E_2(H)$.

Case 2: $x = y$. This means $\sigma(x) = \sigma(y)$. The construction of α' implies f_{uv} and ℓ_{uv} are allocated into the $\sigma(x)$ -th part of \mathcal{V} as well. This, in turn, means that $\sigma(x) \in \lambda'(f_{uv})$ and $\sigma(x) \in \lambda'(\ell_{uv})$. Recall that $\lambda'(f_{uv})$ contains $\sigma(x)$ only when $(x, x) \in E_1(H)$. Hence, $(x, y) \in E_1(H)$.

Now we verify that $\forall_{e \in E_2(H)} |C(e)| = \alpha(e)$. Consider an arc $e = (x, y) \in E_2(H)$. Note that for every directed edge (u, v) in the χ -arc charge $C(e)$, the (u, f_{uv}) of

$E(G')$ contributes to $\alpha'(\sigma(x), \sigma(\tilde{f}_{xy}))$ exactly by one unit. Conversely, for every edge (u, f_{uv}) of $E(G')$ which contributes to $\alpha'(\sigma(x), \sigma(\tilde{f}_{xy}))$, we have $\chi(v) = y$ and thus the directed arc (u, v) contributes to $C(e)$ by one unit. This establishes that $\forall_{e \in E_2(H)} |C(e)| = \alpha(e)$.

The claimed running time follows from the fact that $w(I') = \sum \alpha' = 3 \cdot \sum \alpha = O(d(I))$ and $|V(G')| = O(|E(G)|) = O(d(I) \cdot |V(G)|)$. \square

4 An FPT-algorithm for List Allocation

In this section we give the proof that LA admits an FPT-algorithm. Before we proceed with the details of the proof let us summarize the main steps of the proof that consists of a series of T-FPT-reductions.

1. LIST ALLOCATION is T-FPT-reduced to its restriction, called CLA, where G is a connected graph and only $O(w)$ boxes are used. This reduction takes care of the different ways connected components of G can entirely be placed into the boxes and is based on dynamic programming (see Subsection 4.2).
2. CLA is T-FPT-reduced to a restriction of it, called HCLA, where G is highly connected in the sense that there is no set of w edges that can separate G into two “big” connected components. This reduction is presented in detail in Subsection 4.3 and uses the technique of *recursive understanding*, introduced in [23] and further developed in [7] and [5] (see also [19]), for generalizations of the MULTIWAY CUT problem).
3. HCLA is T-FPT-reduced to a special enhancement of it, called S-HCLA, whose input additionally contains some set $S \subseteq V(G)$ and the problem asks for a solution where all vertices of S are placed in a unique “big” box and all vertices of this box which are incident to crossing edges are contained in S . This variant of the problem permits the application of the technique of *randomized contractions*, introduced in [5] (see Subsection 4.4).
4. Finally, S-HCLA is T-FPT-reduced to LIST ALLOCATION restricted to instances whose sizes are bounded by a function of the parameter. This is presented in Subsection 4.5. It is a dynamic programming based on the fact that an essentially equivalent instance of the problem can be constructed if, apart from S , we remove from G all but a bounded number of the connected components of $G \setminus S$.

4.1 Some (more) definitions

Given two sets A and B we denote by B^A the set containing every function $f : A \rightarrow B$. Given a function $h : A \rightarrow B$ and $S \subseteq A$, we define $h|_S = \{(x, y) \in h \mid x \in S\}$. Given two functions $f_1, f_2 : A \rightarrow \mathbb{Z}_{\geq 0}$ we define $f_1 + f_2 : A \rightarrow \mathbb{Z}_{\geq 0}$ such that $(f_1 + f_2)(x) = f_1(x) + f_2(x)$. Let X be a set and let ζ_1, ζ_2 be two functions mapping X to non-negative integers. We say that $\zeta_1 \leq \zeta_2$ if $\forall i \in X, \zeta_1(i) \leq \zeta_2(i)$. Given a (possibly partial) function $\zeta : X \rightarrow \mathbb{Z}_{\geq 0}$ we define $\mathfrak{F}_{\leq}(\zeta) = \{\zeta' : X \rightarrow \mathbb{Z}_{\geq 0} \mid \zeta' \leq \zeta\}$. Given a set $T \subseteq S$, we define the *restriction of \mathcal{V} to S* as the r -allocation $\mathcal{V} \cap T = (\mathcal{V}^{(1)} \cap T, \dots, \mathcal{V}^{(r)} \cap T)$. Notice that $\mathcal{V} \cap T$ is an r -allocation of T . Given two r -allocations $\mathcal{V}_1 = (V_1^1, \dots, V_r^1)$ and $\mathcal{V}_2 = (V_1^2, \dots, V_r^2)$, we define $\mathcal{V}_1 \cup \mathcal{V}_2 = (V_1^1 \cup V_1^2, \dots, V_r^1 \cup V_r^2)$.

Given two graphs G and G' we set $G \cup G' = (V(G) \cup V(G'), E(G) \cup E(G'))$. Given a graph G and a set $S \subseteq V(G)$, we define $\partial_G(S)$ as the set of all vertices in S that are adjacent to vertices in $V(G) \setminus S$.

Let G be a connected graph. A partition (V_1, V_2) of $V(G)$ is a (q, y) -*good separation* if $|V_1|, |V_2| > q$, $|\partial_G(V_1, V_2)| \leq y$, and $G[V_1]$ and $G[V_2]$ are both connected. A graph G is called (q, y) -*connected* if it does not contain any $(q, y - 1)$ -*good separation*. (Note that for $q = 0$, (q, y) -connectivity corresponds exactly to classical y -edge-connectivity.)

Proposition 2 (Chitnis et al. [5]). *There exists a deterministic algorithm that, with input a n -vertex connected graph G , a $q \in \mathbb{Z}_{\geq 1}$ and $y \in \mathbb{Z}_{\geq 0}$, either finds a (q, y) -good separation, or reports that no such separation exists, in $2^{O(\min\{q, y\} \cdot \log(q+y))} n^3 \log n$ steps.*

For a solution \mathcal{V} to an instance $I = (G, r, \lambda, \alpha)$ of LA, we define

$$E(\mathcal{V}) = \bigcup_{\{i, j\} \in \binom{[r]}{2}} \delta_G(\mathcal{V}^{(i)}, \mathcal{V}^{(j)}).$$

We say that H is (i, λ) -friendly if $i \in \bigcap_{v \in V(H)} \lambda(v)$.

Observation 1. *If \mathcal{V} is a solution for some instance $I = (G, r, \lambda, \alpha)$ of LA then every connected component of $G \setminus E(\mathcal{V})$ is also a connected component of $G[\mathcal{V}^{(i)}]$ for some $i \in [r]$.*

Observation 2. *If \mathcal{V} is a solution for an instance $I = (G, r, \lambda, \alpha)$ of LA, where G has ℓ connected components, then $G \setminus E(\mathcal{V})$ contains at most $w + \ell$ connected components.*

Lemma 5. *There exists an algorithm that, given an instance $I = (G, r, \lambda, \alpha)$ of LA, correctly solves the problem in $n^{O(w)} \cdot 2^{O((w+\ell) \cdot \log r)}$ steps, where ℓ is the number of connected components of G .*

Proof. The algorithm considers each subset F of $E(G)$ of size w . Notice that there are $n^{O(w)}$ such subsets. From Observation 2, $G \setminus F$ has at most $w + \ell$ connected components. From Observation 1, if \mathcal{V} is a solution of LA for I , and $E(\mathcal{V}) = F$, then the vertex set of each connected component of $G \setminus F$ is entirely contained in some $\mathcal{V}^{(i)}$. The algorithm considers all possible ways to assign the $\leq w + \ell$ connected components of G_F to the r indices of I and checks whether this creates a solution for I . As there are $2^{O((w+\ell) \cdot \log r)}$ such assignments, the claimed running time follows. \square

4.2 Connected list allocation

We define the CONNECTED LIST ALLOCATION problem (CLA, in short) as the LIST ALLOCATION with the additional demand that the input graph G is connected and $r \leq 2w$. The reason why we may assume that $r \leq 2w$ is the following. First, we may assume that $w \geq 1$ since otherwise, we can check whether G is (i, λ) -friendly for some $i \in [r]$ and solve the instance for CLA. Suppose $r > 2w > 0$. Then there exists an index $i \in [r]$ such that $\alpha(i, j) = 0$ for every $j \in [r] \setminus \{i\}$. As G is connected, no vertex can be allocated to $\mathcal{V}^{(i)}$ in any solution \mathcal{V} and thus we can remove the i -th part from the instance.

Lemma 6. *If there exists an algorithm solving CLA in $f(w) \cdot p(n)$ steps, then there is an algorithm that solves LA in $\ell \cdot 2^{2w} \cdot f(w) \cdot p(n)$ steps.*

Proof. We present a dynamic programming for LA using an algorithm for CLA as a subroutine. Let C_1, \dots, C_ℓ be the connected components of G and let $G_i = \bigcup_{j=1}^i C_j$. Define a table P for dynamic programming in which the entries $P(i, \alpha')$ run over all $1 \leq i \leq \ell$ and $\alpha' \in \mathfrak{F}_{\leq}(\alpha)$. The value of $P(i, \alpha')$ is YES if the instance $(G_i, r, \lambda|_{V(G_i)}, \alpha')$ is YES. Otherwise, $P(i, \alpha') = \text{NO}$. Note that the given instance (G, r, λ, α) is YES to LA if and only if $P(\ell, \alpha) = \text{YES}$.

For $i = 1$, $G_1 = C_1$ is connected and thus the value of $P(1, \alpha')$ can be correctly determined by solving CLA on the instance $(C_1, r, \lambda|_{V(C_1)}, \alpha')$. For $2 \leq i \leq \ell$, we assume that all values $P(j, \alpha'')$ have been determined for $j < i$ and $\alpha'' \in \mathfrak{F}_{\leq}(\alpha')$. Let g be a function mapping instances of CLA to $\{\text{YES}, \text{NO}\}$ in a canonical way. The following recursion for $P(i, \alpha')$ is easy to verify.

$$P(i, \alpha') = \bigvee_{\alpha'' \in \mathfrak{F}_{\leq}(\alpha')} P(i-1, \alpha' - \alpha'') \wedge g(C_i, r, \lambda|_{V(C_i)}, \alpha'').$$

As $w = \sum \alpha$ by definition, the table P consists of $\ell \cdot |\mathfrak{F}_{\leq}(\alpha)| \leq \ell \cdot 2^w$ entries. Determining each entry amounts to at most $|\mathfrak{F}_{\leq}(\alpha)| \leq 2^w$ table lookups and computations of g . The latter, equivalent to solving an instance to CLA, takes at most $f(w) \cdot p(n)$ steps. Overall, the entire entries of P can be determined in $\ell \cdot 2^{2w} \cdot f(w) \cdot p(n)$ steps. \square

4.3 Highly connected list allocation

We fix two functions $f_1(w) = 2^w \cdot (2w)^{2w}$ and $f_2(w) = w \cdot f_1(w) + 1$, which will appear through this section. We define the HIGHLY CONNECTED LIST ALLOCATION problem (HCLA, in short) as the CONNECTED LIST ALLOCATION problem with the only difference that we additionally demand that the input graph is $(f_2(w), w + 1)$ -connected, where w is the parameter of the problem.

We aim to shrink the size of a given instance $I = (G, r, \lambda, \alpha)$ of CLA by finding out a set E_C of edges such that I/E_C (formal definition given below) is equivalent to I . If it is possible to recursively contract edges so that the obtained instance is of size bounded by a function of w , then we can apply the algorithm of Lemma 5 to solve the final instance of CLA.

For an instance $I = (G, r, \lambda, \alpha)$ of CLA and $B \in \binom{V(G)}{\leq 2w}$, we set $\mathfrak{U}(I, B) = [r]^B \times \mathfrak{F}_{\leq}(\alpha)$. Given a $\mathbf{w} = (\psi, \alpha') \in \mathfrak{U}(I, B)$, we define the instance $I_{\mathbf{w}} = (G, \lambda', r, \alpha')$ of CLA, where $\lambda' = \lambda|_{V(G) \setminus B} \cup \psi$. The set $\tilde{\mathfrak{U}}(I, B)$ is a collection of all $\mathbf{w} \in \mathfrak{U}(I, B)$ such that $I_{\mathbf{w}}$ is a YES-instance of CLA.

Observation 3. *For every instance $I = (G, r, \lambda, \alpha)$ of CLA and $B \in \binom{V(G)}{\leq 2w}$, it holds that $|\mathfrak{U}(I, B)| \leq f_1(w)$.*

Given an instance $I = (G, r, \lambda, \alpha)$ of LA and a set of edges $E_C \subseteq E(G)$, we define the instance I/E_C of LA as $(G/E_C, r, \lambda', \alpha)$, where G/E_C is the graph obtained from G by contracting all the edges in E_C , and λ' is defined as follows: for each vertex $u \in V(G/E_C)$, let $V_u \subseteq V(G)$ be the set of vertices of G that have been identified into u after contracting the edges in E_C (note that, possibly, $V_u = \{u\}$). Then we define $\lambda'(u) := \bigcap_{v \in V_u} \lambda(v)$.

Let $I = (G, r, \lambda, \alpha)$ be an instance of CLA and let $Q \subseteq V(G)$. We set $I[Q] = (G[Q], r, \lambda|_Q, \alpha)$. For a bipartition (V_1, V_2) of $V(G)$, let E_C be a set of edges in $G[V_1]$. Then the gluing of $G[V_1]/E_C$ and $G[V_2]$ along $\delta_G(V_1, V_2)$, denoted as $G[V_1]/E_C \oplus_{\delta} G[V_2]$, can be naturally defined: starting from the disjoint union of $G[V_1]/E_C$ and $G[V_2]$, for each edge $e = (u, v) \in \delta_G(V_1, V_2)$ with $u \in V_1$, we add an edge e' whose one endpoint is the vertex into which u is identified and the other endpoint is v . Notice that $G[V_1]/E_C \oplus_{\delta} G[V_2] = G/E_C$.

The next lemma demonstrates the condition for a set of edges E_C under which I and I/E_C are equivalent.

Lemma 7. *Let $I = (G, r, \lambda, \alpha)$ be an instance of CLA, let (V_1, V_2) be a bipartition of $V(G)$ such that $I[V_1]$ is an instance of HCLA with $|V_1| > f_2(w)$, let $B \subseteq V_1$ with $|B| \leq 2w$, let $\mathcal{S} = \{\mathcal{V}_{\mathbf{w}} \mid \mathbf{w} \in \tilde{\mathfrak{U}}(I[V_1], B)\}$ be a collection of (arbitrary chosen) solutions to $I[V_1]_{\mathbf{w}}$ for every $\mathbf{w} \in \tilde{\mathfrak{U}}(I[V_1], B)$, and let $E_C = E(G[V_1]) \setminus \bigcup_{\mathcal{V}_{\mathbf{w}} \in \mathcal{S}} E(\mathcal{V}_{\mathbf{w}})$. Then $E_C \neq \emptyset$. Furthermore, I and I/E_C are equivalent instances of CLA.*

Proof. Note also that by Observation 3, $|\bigcup_{\mathcal{V}_{\mathbf{w}} \in \mathcal{S}} E(\mathcal{V}_{\mathbf{w}})| \leq w \cdot |\tilde{\mathfrak{U}}(I[V_1], B)| \leq w \cdot f_1(w)$. Since by hypothesis the graph $G[V_1]$ is connected and satisfies $|V_1| > f_2(w)$, it holds that $|E(G[V_1])| \geq f_2(w)$, and thus $|E_C| = |E(G[V_1]) \setminus \bigcup_{\mathcal{V}_{\mathbf{w}} \in \mathcal{S}} E(\mathcal{V}_{\mathbf{w}})| \geq f_2(w) - w \cdot f_1(w) = 1$, hence there exists at least one edge in E_C .

We need to prove that I is a YES-instance of CLA if and only if I/E_C is. First note that contracting edges does not harm the connectivity of G and, as r and α are the same in I and in I/E_C , $r \leq 2 \sum \alpha$ holds. Therefore I/E_C is indeed an instance of CLA.

Assume first that I is a YES-instance, and let \mathcal{V} be a solution of CLA for I . Let $\psi_B = \{(v, \mathcal{V}(v)) \mid v \in B\}$, where $\mathcal{V}(v)$ denotes the integer $i \in [r]$ such that $v \in \mathcal{V}^{(i)}$. Let also α_1 be the element of $\mathfrak{F}_{\leq}(\alpha)$ such that for any two distinct integers $i, j \in [r]$, $\alpha_1(i, j) = |\delta_{G[V_1]}(\mathcal{V}^{(i)}, \mathcal{V}^{(j)})|$, and let $\mathbf{w}_1 = (\psi_B, \alpha_1)$. Note that by the definition of \mathbf{w}_1 and since we assume that $I[V_1]$ is an instance of HCLA, it holds that $\mathbf{w}_1 \in \tilde{\mathfrak{U}}(I[V_1], B)$. Let $\mathcal{V}_{\mathbf{w}_1}$ be the solution to $I[V_1]_{\mathbf{w}_1}$ in the collection \mathcal{S} and note that by the definition of the set E_C , the endpoints of any edge in E_C belong to the same part of $\mathcal{V}_{\mathbf{w}_1}$. We now proceed to define an r -allocation \mathcal{V}' for I/E_C . For each vertex $u \in V(G/E_C)$, let $V_u \subseteq V(G)$ be the set of vertices of G that have been identified into u after contracting the edges in E_C , so each of the sets V_u belongs entirely to the same part of $\mathcal{V}_{\mathbf{w}_1}$. For every $u \in V_2$, we define $\mathcal{V}'(u) = \mathcal{V}(u)$, and for every $u \in V(G/E_C) \setminus V_2$, we define $\mathcal{V}'(u) = \mathcal{V}_{\mathbf{w}_1}(v)$, for an arbitrary vertex $v \in V_u$. The above discussion implies that the r -allocation \mathcal{V}' is well-defined and constitutes a solution of CLA for I/E_C .

Conversely, assume now that I/E_C is a YES-instance, and let \mathcal{V}' be a solution of CLA for I' . For each vertex $u \in V(G/E_C)$, let again $V_u \subseteq V(G)$ be the set of vertices of G that have been identified into u after contracting the edges in E_C . We proceed to define an r -allocation \mathcal{V} for I . For every $v \in V(G)$, if $v \in V_u$ for a vertex $u \in V(G/E_C)$, we define $\mathcal{V}(v) = \mathcal{V}'(u)$. We claim that the r -allocation \mathcal{V} is a solution of CLA for I . Indeed, by definition of \mathcal{V}' of the instance I/E_C , we have that for every vertex $v \in V(G)$ belonging to a set $V_u \subseteq V(G)$, it holds that $\mathcal{V}(v) \in \lambda'(u) = \bigcap_{w \in V_u} \lambda(w) \subseteq \lambda(v)$. On the other hand, since edge multiplicities are summed up when contracting edges, it holds that for any two distinct integers $i, j \in [r]$, $|\delta_G(\mathcal{V}^{(i)}, \mathcal{V}^{(j)})| = |\delta_{G/E_C}(\mathcal{V}'^{(i)}, \mathcal{V}'^{(j)})| = \alpha(i, j)$. \square

Due to Lemma 7, an edge set E_C to be contracted can be obtained if we can compute a collection of solutions $\mathcal{S} = \{\mathcal{V}_{\mathbf{w}} \mid \mathbf{w} \in \tilde{\mathfrak{U}}(I[V_1], B)\}$ to the instances $I_{\mathbf{w}}$ of HCLA. This can be done by solving the instance $I_{\mathbf{w}}$ for each $\mathbf{w} \in \mathfrak{U}([V_1], B)$ and this requires at most $f_1(w)$ iterations of HCLA-solver by Observation 3. This point is formalized in the next observation.

Observation 4. *If there exists an algorithm that can find, if it exists, a solution of HCLA in $f(w) \cdot p(n)$ steps, then there is an algorithm that, given an instance*

$I = (G, r, \lambda, \alpha)$ of HCLA and a set $B \subseteq V(G)$ where $|B| \leq 2 \cdot w$, computes the set $\tilde{\mathcal{U}}(I, B)$, a set $\mathcal{S} = \{\mathcal{V}_{\mathbf{w}} \mid \mathbf{w} \in \tilde{\mathcal{U}}(I, B)\}$ in case $\tilde{\mathcal{U}}(I, B) \neq \emptyset$, and the set $E_{\mathcal{C}} = E(G[V_1]) \setminus \bigcup_{\mathcal{V}_{\mathbf{w}} \in \mathcal{S}} E(\mathcal{V}_{\mathbf{w}})$ in $f(w) \cdot p(n) \cdot f_1(w)$ steps.

Algorithm : **shrink**(H, B)

Input : A graph H and a set $B \subseteq V(H)$ s.t. $|B| \leq 2 \cdot w$, $|V(H)| > f_2(w)$.

Output : A graph H^{new} having at most $f_2(w)$ vertices or a report that I is a NO-instance.

Global Variable: An instance I' of CLA.

1. **if** H has a $(f_2(w), w)$ -separation (V_1, V_2) **then**
2. **let** i be an integer in $\{1, 2\}$ such that $|B \cap V_i| \leq w$
3. **let** $B' = (B \cap V_i) \cup (V(\delta(V_1, V_2)) \cap V_i)$
4. **let** $H' = \text{shrink}(H[V_i], B')$, $H^{\text{new}} = H' \oplus_{\delta} H[V_{3-i}]$
5. **let** B^{new} be the vertices of H^{new} onto which the vertices of B are identified
6. **if** $|V(H^{\text{new}})| > f_2(w)$ **then**
7. **return** **shrink**($H^{\text{new}}, B^{\text{new}}$)
8. **end**
9. **return** H^{new}
10. **else**
11. compute $\tilde{\mathcal{U}}(I'[V(H)], B)$
12. **if** $\tilde{\mathcal{U}}(I'[V(H)], B) = \emptyset$ **then**
13. **report** that I is a NO-instance
14. **else**
15. compute $\mathcal{S} = \{\mathcal{V}_{\mathbf{w}} \mid \mathbf{w} \in \tilde{\mathcal{U}}(I'[V(H)], B)\}$,
- $E_{\mathcal{C}} = E(H) \setminus \bigcup_{\mathcal{V}_{\mathbf{w}} \in \mathcal{S}} E(\mathcal{V}_{\mathbf{w}})$
16. **let** $I' \leftarrow I'/E_{\mathcal{C}}$
17. **return** $H/E_{\mathcal{C}}$
18. **end**
19. **end**

CLA is solved by reducing the instance size via iteratively contracting edges, and finding contractible edges boils down to solving HCLA by Observation 4. The next lemma formalize this as a TFT-reduction from CLA to HCLA. Although the idea of the reduction itself is straightforward, we provide rather a nontrivial reduction to achieve a better running time for CLA.

Lemma 8. *If HCLA can be solved in $f(w) \cdot p(n)$ steps, then CLA can be solved in $\max\{2^{O(w^2 \cdot \log w)} \cdot n^4 \cdot \log n, f(w) \cdot p(n) \cdot 2^{O(w \cdot \log w)}\}$ steps.*

Proof. Let $I = (G, r, \lambda, \alpha)$ be an instance of CLA. If G has less than $f_2(w)$ vertices then, because of Lemma 5 and the fact that $r \leq 2w$, the problem can be solved in $2^{O(w^2 \cdot \log w)}$ steps. If not, we call the algorithm $\mathbf{shrink}(G, \emptyset)$ with the given instance $I = (G, r, \lambda, \alpha)$ as the global variable I' . The global variable I' is initialized (only once) at the initial call $\mathbf{shrink}(G, \emptyset)$, and is considered to be out of the scope of the subsequent calls $\mathbf{shrink}(H, B)$. Hence, the subsequent calls share the access to the same global variable at line 16.

Let $G(I')$ refer to the input graph of the current global variable I' . We need the following claim, which ascertains the property of $\mathbf{shrink}(G, \emptyset)$ and also ensure that $I'[V(H)]$ referred to at line 15 is a valid instance.

Claim 1. *The follow statements hold.*

- (a) *Throughout the execution of $\mathbf{shrink}(G, \emptyset)$, whenever a call $\mathbf{shrink}(H, B)$ is made, the graph H is an induced subgraph of $G(I')$ for the current global variable I' .*
- (b) *The graph⁵ returned by each call $\mathbf{shrink}(H, B)$ can be obtained by contracting edges of H and has at most $f_2(w)$ vertices.*

PROOF OF THE CLAIM: Let m be the number of calls for $\mathbf{shrink}(H, B)$ during the performance of $\mathbf{shrink}(G, \emptyset)$, including the initial call itself. At any step in $\mathbf{shrink}(G, \emptyset)$, let i be the number of calls invoked so far (in Lines 4 and 7) and j be the number of return calls (in Lines 7, 9, and 17). Intuitively, (i, j) stands for the current position in the recursion tree during the course of the algorithm $\mathbf{shrink}(G, \emptyset)$. Before we make the first call $\mathbf{shrink}(G, \emptyset)$, we have $(i, j) = (0, 0)$. Clearly, $j \leq i$ during the entire execution, and the inequality is strict unless $\mathbf{shrink}(G, \emptyset)$ terminates. We prove the statements (a) and (b) by induction on $i + j$. Notice that the algorithm traverses from (i, j) to $(i + 1, j)$ exactly when a $(i + 1)$ -st call $\mathbf{shrink}(H, B)$ is made, and it traverses from (i, j) to $(i, j + 1)$ when a call $\mathbf{shrink}(H, B)$ returns a $(j + 1)$ -st output. Therefore, we prove the following modification of (a) and (b).

(a') Whenever the traversal $(i, j) \rightarrow (i + 1, j)$ is made by a call $\mathbf{shrink}(H, B)$, H is an induced subgraph of $G(I')$, where I' is the current global variable.

(b') Whenever the traversal $(i, j) \rightarrow (i, j + 1)$ is made by a return of $\mathbf{shrink}(H, B)$, the returned graph can be obtained by contracting edges of H , and has at most $f_2(w)$ vertices (unless $\mathbf{shrink}(H, B)$ reports that I is a NO-instance).

⁵For notational convenience, we abuse $\mathbf{shrink}(H, B)$ also to denote the graph returned by the call $\mathbf{shrink}(H, B)$ whenever it is clear from the context.

In the base case, that is, when the first call $\mathbf{shrink}(G, \emptyset)$ makes the traversal $(0, 0) \rightarrow (1, 0)$, then it is clear that the statement (a') holds. Now we consider the case when a traversal is made to (i, j) . As an induction hypothesis, we assume that any traversal to (i', j') with $i' + j' < i + j$ satisfies either (a') or (b') , depending on whether it increase the first or the second coordinate.

Case A: when the traversal $(i, j) \rightarrow (i + 1, j)$ is made at Line 4: Let $\mathbf{shrink}(H[V_i], B')$ be the $(i + 1)$ -st call invoked at line 4. By induction hypothesis, H is an induced subgraph of $G(I')$, $H[V_i]$ is an induced subgraph of H , and I' does not changes between i -th and $(i + 1)$ -st calls. Hence, the statement (a') holds.

Case B: when the traversal $(i, j) \rightarrow (i, j + 1)$ is made at Line 17: Clearly the graph returned by $\mathbf{shrink}(H, B)$ is obtained by contracting edges of H . The number of vertices in the graph returned by $\mathbf{shrink}(H, B)$ is at most $f_2(w)$ since we contract all edges except for those in $\bigcup_{\mathcal{V}_w \in \mathcal{S}} E(\mathcal{V}_w)$, whose size is bounded by $w \cdot f_1(w) < f_2(w)$, and the obtained graph is connected. Therefore, the statement (b') holds in this case.

Case C: when the traversal $(i, j) \rightarrow (i, j + 1)$ is made at Line 9: Here we consider the case when $\mathbf{shrink}(H, B)$ returns an output at line 9. By induction hypothesis, H' is obtained by contracting edges of $H[V_i]$. Notice that $H^{\text{new}} = H' \oplus_{\delta} H[V_{3-i}]$ can be obtained from H by contracting the same set of edges that have been contracted in $H[V_1]$, resulting in H' . It clearly contains at most $f_2(w)$ vertices, thereby satisfying the statement (b') .

Case D: when the traversal $(i, j) \rightarrow (i + 1, j)$ is made at Line 7: Let I' be the global variable when $\mathbf{shrink}(H, B)$ is called, and I'' be the global variable when $\mathbf{shrink}(H^{\text{new}}, B^{\text{new}})$ is called at Line 7. By induction hypothesis, H' and thus H^{new} can be obtained by contracting edges of H . During the traversal from the call $\mathbf{shrink}(H, B)$ and the call $\mathbf{shrink}(H^{\text{new}}, B^{\text{new}})$, the contracted edges that transformed I' to I'' are exactly those which transformed $H[V_1]$ to H' , and equivalently H to H^{new} . Since H is an induced subgraph of $G(I')$, we conclude that H^{new} is an induced subgraph of $G(I'')$, and the statement (a') holds.

Case E: when the traversal $(i, j) \rightarrow (i, j + 1)$ is made at Line 7: Here we consider the case when $\mathbf{shrink}(H, B)$ returns an output at Line 7. By case D, we know that H^{new} is an induced subgraph of the current global variable I' . By induction hypothesis, the graph returned by $\mathbf{shrink}(H^{\text{new}}, B^{\text{new}})$ can be obtained by contracting edges of H^{new} . Since H^{new} itself can be obtained by contraction edges of H by the same argument as in Case C, the graph returned can be obtained by contracting edges of H . To see that the number of vertices in the returned graph is at most $f_2(w)$, we resort to the induction hypothesis, Case B and C. Therefore, (b') holds as well.

◇

In order to establish the correctness of the algorithm $\mathbf{shrink}(H, B)$, we need

the following claim.

Claim 2. *While the initial call $\mathbf{shrink}(G, \emptyset)$ is carried out, the global variable I' remains equivalent to I .*

PROOF OF THE CLAIM: Claim 1 implies that at Line **15**, I' and $I'[V(H)]$ are indeed instances of CLA and HCLA meeting the conditions of Lemma 7. Hence, any current global variable I' and the new global variable I'/E_C updated at line 16 are equivalent by Lemma 7. As $I' = I$ at the outset of $\mathbf{shrink}(G, \emptyset)$, the current global variable I' is equivalent to I during the course of $\mathbf{shrink}(G, \emptyset)$. \diamond

If $\mathbf{shrink}(G, \emptyset)$ reports that I is a NO-instance, it means that $\tilde{\mathfrak{U}}(I'[V(H)], B) = \emptyset$ for some call $\mathbf{shrink}(H, B)$. Note that if I' is a YES-instance, $\tilde{\mathfrak{U}}(I'[V(H)], B) \neq \emptyset$ for every subgraph H of $G(I')$. By Claim 1, the graph H is indeed a subgraph of $G(I')$ for any call $\mathbf{shrink}(H, B)$ incurred in the course of $\mathbf{shrink}(G, \emptyset)$ and thus I' is a NO-instance. Together with Claim 2, this implies that I indeed a NO-instance.

Suppose that $\mathbf{shrink}(G, \emptyset)$ returns a graph (i.e. does not report that I is a NO-instance). By Claim 1 this means that the graph $\mathbf{shrink}(G, \emptyset)$ can be obtained from G by contracting edges of G , and has at most $f_2(w)$ vertices. Let I^{new} be the final global variable when $\mathbf{shrink}(G, \emptyset)$ terminates. The sequence of edge contractions applied to G leading to $\mathbf{shrink}(G, \emptyset)$ is also applied to the initial global variable $I' = I$. Therefore, $G(I^{\text{new}})$ contains no more vertices than the graph $\mathbf{shrink}(G, \emptyset)$ does, which is at most $f_2(w)$. From Claim 2, I^{new} is indeed equivalent to I . Therefore, by applying the algorithm of Lemma 5 to I^{new} , we can correctly solve the instance I .

What remains is to prove that CLA can be solved in the claimed running time, assuming an algorithm which solves HCLA in $f(w) \cdot p(n)$ steps. Let now $T(n, w)$ be the running time of Algorithm \mathbf{shrink} when it runs on an instance $I = (G, r, \lambda, \alpha)$ where $|V(G)| = n$ and $w = \sum \alpha$. Notice that

$$T(n, w) \leq \max_{f_1(w) \leq n' \leq n - f_1(w)} \{T_1(n, w) + T(n', w) + T(f_2(w) + n - n', w), T_2(n, w)\},$$

where T_1 is the running time of required by line **1** and T_2 is the running time required to compute E_C in line **15**. From Proposition 2, $T_1(n, w) = 2^{O(w^2 \cdot \log w)} \cdot n^3 \cdot \log n$ and, from Observation 4, $T_2(n, w) = f(w) \cdot p(n) \cdot f_1(w)$. By resolving the above recursion, we obtain that $T(n, w) = \max\{T_1(n, w) \cdot n, T_2(n, w)\}$. Lastly, solving an instance with at most $f_2(w)$ vertices using an algorithm of Lemma 5 requires at most $2^{O(w^2 \cdot \log w)}$ steps, which yields the claimed running time. \square

4.4 Split highly connected list allocation

Given a graph G , an integer $r \in \mathbb{Z}_{\geq 1}$, an allocation $\mathcal{V} = \{\mathcal{V}^{(1)}, \dots, \mathcal{V}^{(r)}\}$ of $V(G)$ and two integers $j \in [r]$ and $x \in \mathbb{Z}_{\geq 0}$, we say that \mathcal{V} is x -bounded out of j if

$\sum_{i \in [r] \setminus \{j\}} |\mathcal{V}^{(i)}| \leq x$. We define the SPLIT HIGHLY CONNECTED LIST ALLOCATION problem (S-HCLA, in short) so that its instances are as the instances of HIGHLY CONNECTED LIST ALLOCATION enhanced with some subset S of $V(G)$ and where we impose that $|V(G)| > 2w \cdot f_2(w)$ and that the requested solution \mathcal{V} , additionally, satisfies the following condition: There exists some $j \in [r]$, such that

- A. \mathcal{V} is $w \cdot f_2(w)$ -bounded out of j and
- B. $\partial_G(\mathcal{V}^{(j)}) \subseteq S \subseteq \mathcal{V}^{(j)}$.

Lemma 9. *Let \mathcal{V} be a solution of HCLA for an instance $I = (G, r, \lambda, \alpha)$ where $|V(G)| > 2w \cdot f_2(w)$. Then there is a unique $j \in [r]$ such that \mathcal{V} is $w \cdot f_2(w)$ -bounded out of j and a unique $C \in \mathcal{C}(G \setminus E(\mathcal{V}))$ with $|V(C)| > f_2(w)$. Moreover, for such C and j , C is a subgraph of $G[\mathcal{V}^{(j)}]$.*

Proof. Let C be a connected component of $G \setminus E(\mathcal{V})$ that has maximum number of vertices. As $G \setminus E(\mathcal{V})$ has at most $w + 1$ connected components and $2w \cdot f_2(w) \geq (w + 1) \cdot f_2(w) + 1$, we deduce that $|V(C)| > f_2(w)$. Using Observation 1, we know that C belongs entirely in some $\mathcal{V}^{(j)}$. As G is $(f_2(w), w + 1)$ -connected, every connected component of $G \setminus E(\mathcal{V})$ that is different from C has at most $f_2(w)$ vertices. This implies that the union of the parts of \mathcal{V} that are different from $\mathcal{V}^{(j)}$ contains at most $w \cdot f_2(w)$ vertices. Moreover j is unique as, otherwise, $|V(G)| \leq 2w \cdot f_2(w)$. \square

Proposition 3 (Chitnis et al. [5]). *There exists an algorithm that given a set U of size n and two integers $a, b \in [0, n]$, outputs a set $\mathcal{F} \subseteq 2^U$ with $|\mathcal{F}| = 2^{O(\min\{a, b\} \cdot \log(a+b+1))} \cdot \log n$ such that for every two sets $A, B \subseteq U$, where $A \cap B = \emptyset$ and $|A| \leq a$ and $|B| \leq b$, there exists a set $S \in \mathcal{F}$ with $A \subseteq S$ and $B \cap S = \emptyset$, in $2^{O(\min\{a, b\} \cdot \log(a+b+1))} \cdot n \cdot \log n$ steps.*

The proof of the following lemma uses Proposition 3 and Lemmata 5 and 9.

Lemma 10. *Given an algorithm solving S-HCLA in $f(w) \cdot p(n)$ steps, then there is an algorithm solving HCLA in $f(w) \cdot 2^{O(w^2 \cdot \log w)} \cdot \log n \cdot \max\{n, p(n)\}$ steps.*

Proof. Let I be an instance of HCLA. If $|V(G)| \leq 2w \cdot f_2(w)$, HCLA can be solved in $(2w \cdot f_1(w))^w \cdot 2^{O(w \cdot \log w)} = 2^{O(w^2 \cdot \log w)}$ steps because of Lemma 5 (applied for $\ell = 1$ and $r \leq 2w$).

Let \mathcal{F} be a family of subsets of $V(G)$ such that the condition of Proposition 3 is satisfied for $a = w$ and $b = w \cdot f_2(w)$. We claim that I is a YES-instance of HCLA if and only if for some $S \in \mathcal{F}$, (I, S) is a YES-instance of S-HCLA. Recall that (I, S) is an instance of S-HCLA, as $|V(G)| > 2w \cdot f_2(w)$.

In the non-trivial direction, assume that \mathcal{V} is a solution for I . By applying Lemma 9 on I , we know that there is a unique j such that \mathcal{V} is $w \cdot f_2(w)$ -bounded out of j . Let $A = \partial_G(\mathcal{V}^{(j)})$ and $B = \bigcup_{i \in [r] \setminus \{j\}} \mathcal{V}^{(i)}$. Clearly, $|A| \leq w$ and $|B| \leq$

$w \cdot f_2(w)$. By the definition of \mathcal{F} , there exists some set $S \in \mathcal{F}$ such that $A \subseteq S$ and $B \cap S = \emptyset$. Therefore $\partial_G(\mathcal{V}^{(j)}) \subseteq S \subseteq \mathcal{V}^{(j)}$ and (I, S) is a YES-instance of S-HCLA as required.

Suppose now that A is an algorithm that solves S-HCLA in $f(w) \cdot p(n)$ steps. To solve HCLA, we apply A on (I, S) for all $S \in \mathcal{F}$. If we obtain a solution to (I, S) for some $S \in \mathcal{F}$ we output this solution as a solution to I , otherwise we output that I is a NO-instance of HCLA. As $|\mathcal{F}| = 2^{O(w \cdot \log(w \cdot f_2(w)))} \cdot \log n = 2^{O(w^2 \cdot \log w)} \cdot \log n$, this algorithm runs in $2^{O(w^2 \cdot \log w)} \cdot \log n \cdot n + 2^{O(w^2 \cdot \log w)} \cdot \log n \cdot f(w) \cdot p(n)$ steps as required. \square

4.5 An algorithm for solving S-HCLA

Below we present a dynamic programming algorithm for solving S-HCLA.

Lemma 11. S-HCLA can be solved in $2^{O(w^2 \cdot \log w)} \cdot n$ steps.

Proof. We present a dynamic programming for S-HCLA using the brute-force algorithm of Lemma 5 as a subroutine. Let (I, S) be an instance of SHCLA where $I = (G, r, \lambda, \alpha)$ and $S \subseteq V(G)$, and let C_1, \dots, C_ℓ be the vertex sets of the graphs in $\mathcal{C}(G \setminus S)$. Let $G_i = G[S \cup \bigcup_{1 \leq i' \leq i} C_{i'}]$ for $1 \leq i \leq \ell$ and specifically $G_0 = G[S]$. For $s \in [r]$, we define two functions $\lambda_s, \lambda_s^* : V(G) \rightarrow 2^{[r]}$ such that

$$\lambda_s(x) = \begin{cases} \{s\} & x \in S \\ \lambda(x) & x \in V(G) \setminus S \end{cases}$$

and

$$\lambda_s^*(x) = \begin{cases} \{s\} & x \in S, \\ \lambda(x) \setminus \{s\} & x \in V(G) \setminus S. \end{cases}$$

For each $s \in [r]$, we have a table P_s for dynamic programming in which the entries $P_s(i, \alpha', c')$ are either YES or NO, and run over all $0 \leq i \leq \ell$, $\alpha' \in \mathfrak{F}_{\leq}(\alpha)$ and $0 \leq c' \leq w \cdot f_2(w)$. The entries of P_s are determined recursively as follows.

- $P_s(0, \alpha', c') = \text{YES}$ if and only if $\alpha' = \mathbf{0}$, $c' = 0$ and $G[S]$ is (s, λ) -friendly.
- For $1 \leq i \leq \ell$, $P_s(i, \alpha', c') = \text{YES}$ if and only if
 - (i) $P_s(i-1, \alpha', c') = \text{YES}$ and $G[C_i]$ is (s, λ) -friendly, or
 - (ii) $|C_i| \leq c'$, and there exists $\alpha'' \in \mathfrak{F}_{\leq}(\alpha')$ such that $P_s(i-1, \alpha'', c' - |C_i|) = \text{YES}$ and $(G[S \cup C_i], r, \lambda_s^*|_{S \cup C_i}, \alpha' - \alpha'')$ is a YES-instance for LA.

Claim 3. $P_s(i, \alpha', c') = \text{YES}$ if and only if the instance $(G_i, r, \lambda_s|_{V(G_i)}, \alpha')$ admits a solution \mathcal{V} for LA such that $\sum_{i \in [r] \setminus \{s\}} |\mathcal{V}^{(i)}| = c'$, and either $C_j \subseteq \mathcal{V}^{(s)}$ or $C_j \cap \mathcal{V}^{(s)} = \emptyset$ for each $1 \leq j \leq i$.

PROOF OF THE CLAIM: When $i = 0$, it is tedious to verify the claim. We prove by induction on i .

Firstly, we prove the forward direction. Suppose that $P_s(i, \alpha', c') = \text{YES}$ and consider the instance $(G_i, r, \lambda_s|_{V(G_i)}, \alpha')$ of LA. If case (i) of the recursion holds, then by induction hypothesis, there exists a solution \mathcal{V}' to $(G_{i-1}, r, \lambda_s|_{V(G_{i-1})}, \alpha') = (G_i \setminus C_i, r, \lambda_s|_{V(G_i) \setminus C_i}, \alpha')$ such that $\sum_{i \in [r] \setminus \{s\}} |\mathcal{V}'^{(i)}| = c'$, and either $C_j \subseteq \mathcal{V}'^{(s)}$ or $C_j \cap \mathcal{V}'^{(s)} = \emptyset$ for each $1 \leq j \leq i-1$. Let \mathcal{V} be an r -allocation obtained from \mathcal{V}' by adding all vertices of C_i to the part $\mathcal{V}^{(s)}$. Since $G[C_i]$ is (s, λ) -friendly and $C_i \cap \partial_{G_i}(\mathcal{V}^{(s)}) = \emptyset$, it follows that \mathcal{V} is a solution to $(G_i, r, \lambda_s|_{V(G_i)}, \alpha')$. Note that \mathcal{V} meets the two conditions of our claim.

Suppose that case (i) of the recursion does not hold for the entry $P_s(i, \alpha', c') = \text{YES}$, but case (ii) does. From $P_s(i-1, \alpha'', c' - |C_i|) = \text{YES}$ and induction hypothesis, there exists a solution \mathcal{V}' to $(G_{i-1}, r, \lambda_s|_{V(G_{i-1})}, \alpha'') = (G_i \setminus C_i, r, \lambda_s|_{V(G_i) \setminus C_i}, \alpha'')$ such that $\sum_{i \in [r] \setminus \{s\}} |\mathcal{V}'^{(i)}| = c' - |C_i|$, and either $C_j \subseteq \mathcal{V}'^{(s)}$ or $C_j \cap \mathcal{V}'^{(s)} = \emptyset$ for each $1 \leq j \leq i-1$. Let \mathcal{V}'' be a solution to $(G[S \cup C_i], r, \lambda_s^*|_{S \cup C_i}, \alpha' - \alpha'')$, and let \mathcal{V} be $\mathcal{V}' \cup \mathcal{V}''$. Indeed, \mathcal{V} is an r -allocation of $V(G_i)$ since $V(G_{i-1}) \cap (S \cup C_i) = S$, $S \subseteq \mathcal{V}^{(s)}$ and $S \subseteq \mathcal{V}''^{(s)}$. It is easy to see that \mathcal{V} is a solution to $(G_i, r, \lambda_s|_{V(G_i)}, \alpha')$. Furthermore, due to the definition of λ_s^* , we have $\sum_{i \in [r] \setminus \{s\}} |\mathcal{V}''^{(i)}| = |C_i|$, and thus $\sum_{i \in [r] \setminus \{s\}} |\mathcal{V}^{(i)}| = \sum_{i \in [r] \setminus \{s\}} |\mathcal{V}'^{(i)}| + \sum_{i \in [r] \setminus \{s\}} |\mathcal{V}''^{(i)}| = (c' - |C_i|) + |C_i| = c'$. Notice that $\mathcal{V}^{(s)} = \mathcal{V}'^{(s)} \cup \mathcal{V}''^{(s)}$ as $C_i \cap \mathcal{V}''^{(s)} = \emptyset$. This implies that we have $C_j \subseteq \mathcal{V}^{(s)} = \mathcal{V}'^{(s)}$ or $C_j \cap \mathcal{V}^{(s)} = C_j \cap \mathcal{V}'^{(s)} = \emptyset$ for $1 \leq j \leq i-1$. It remains to observe that $C_i \cap \mathcal{V}''^{(s)} = \emptyset$ also implies $C_i \cap \mathcal{V}^{(s)} = \emptyset$.

Secondly, let us prove the opposite direction. Let \mathcal{V} be a solution to $(G_i, r, \lambda_s|_{V(G_i)}, \alpha')$ meeting the conditions of the claim. Consider the two cases.

Case 1: Suppose $C_i \subseteq \mathcal{V}^{(s)}$. Clearly, $G[C_i]$ is (s, λ) -friendly. We argue that $P_s(i-1, \alpha', c') = \text{YES}$, which implies $P_s(i, \alpha', c') = \text{YES}$ by the recursion case (i) for P_s . Let \mathcal{V}' be the restriction of \mathcal{V} to $V(G_{i-1})$, i.e. $(\mathcal{V}^{(1)} \setminus C_i, \dots, \mathcal{V}^{(r)} \setminus C_i)$. By induction hypothesis, in order to prove $P_s(i-1, \alpha', c') = \text{YES}$, it suffices to show that \mathcal{V}' is a solution to $(G_{i-1}, r, \lambda_s|_{V(G_{i-1})}, \alpha')$ such that $\sum_{j \in [r] \setminus \{s\}} |\mathcal{V}'^{(j)}| = c'$, and either $C_j \subseteq \mathcal{V}'^{(s)}$ or $C_j \cap \mathcal{V}'^{(s)} = \emptyset$ for each $1 \leq j \leq i-1$. Indeed, \mathcal{V}' is a solution to $(G_{i-1}, r, \lambda_s|_{V(G_{i-1})}, \alpha')$ for $C_i \subseteq \mathcal{V}^{(s)} \setminus \partial_{G_i}(\mathcal{V}^{(s)})$, which implies $\delta_{G_{i-1}}(\mathcal{V}'^{(j)}, \mathcal{V}'^{(k)}) = \delta_{G_i}(\mathcal{V}^{(j)}, \mathcal{V}^{(k)})$ for every $1 \leq j < k \leq r$. Note that $\sum_{j \in [r] \setminus \{s\}} |\mathcal{V}'^{(j)}| = \sum_{j \in [r] \setminus \{s\}} |\mathcal{V}^{(j)}| = c'$. Moreover, from $\mathcal{V}'^{(s)} = \mathcal{V}^{(s)} \setminus C_i$, $\mathcal{V}'^{(j)} = \mathcal{V}^{(j)}$ for $j \neq s$, and the fact that either $C_j \subseteq \mathcal{V}^{(s)} = \mathcal{V}'^{(s)} \cup C_i$ or $C_j \cap \mathcal{V}^{(s)} = \emptyset$ holds for each $1 \leq j \leq i-1$, we have either $C_j \subseteq \mathcal{V}'^{(s)}$ or $C_j \cap \mathcal{V}'^{(s)} = C_j \cap (\mathcal{V}^{(s)} \setminus C_i) = C_j \cap \mathcal{V}^{(s)} = \emptyset$ for each $1 \leq j \leq i-1$.

Case 2: Suppose $C_i \cap \mathcal{V}^{(s)} = \emptyset$. For $\sum_{j \in [r] \setminus \{s\}} |\mathcal{V}^{(j)}| = c'$, we have $|C_i| \leq c'$. Let \mathcal{V}' be the restriction of \mathcal{V} to $V(G_{i-1})$, i.e. $(\mathcal{V}^{(1)} \setminus C_i, \dots, \mathcal{V}^{(r)} \setminus C_i)$ and let \mathcal{V}'' be the restriction of \mathcal{V} to $S \cup C_i$. Also let α'' be such that $\alpha''(j, k) = |\delta_{G_{i-1}}(\mathcal{V}'^{(j)}, \mathcal{V}'^{(k)})|$ for

every $1 \leq j < k \leq r$. In order to show $P_s(i, \alpha', c') = \text{YES}$, it suffices to verify that $P_s(i-1, \alpha'', c' - |C_i|) = \text{YES}$ and $(G[S \cup C_i], r, \lambda_s^*|_{S \cup C_i}, \alpha' - \alpha'')$ is a YES-instance for LA.

To verify $P_s(i-1, \alpha'', c' - |C_i|) = \text{YES}$, notice that \mathcal{V}' is a solution to $(G_{i-1}, r, \lambda_s|_{V(G_i)}, \alpha'')$ and $\sum_{i \in [r] \setminus \{s\}} |\mathcal{V}'^{(i)}| = \sum_{i \in [r] \setminus \{s\}} |\mathcal{V}^{(i)} \setminus C_i| = \sum_{i \in [r] \setminus \{s\}} |\mathcal{V}^{(i)}| - |C_i| = c' - |C_i|$. For each $1 \leq j \leq i-1$, if $C_j \subseteq \mathcal{V}^{(s)}$, then $C_j \subseteq \mathcal{V}'^{(s)}$ since $\mathcal{V}^{(s)} = \mathcal{V}'^{(s)} \cup C_i$ and $C_j \cap C_i = \emptyset$. Otherwise, $C_j \cap \mathcal{V}'^{(s)} = C_j \cap \mathcal{V}^{(s)} = \emptyset$ for $\mathcal{V}'^{(s)} = \mathcal{V}^{(s)}$. By induction hypothesis, that $P_s(i-1, \alpha'', c' - |C_i|) = \text{YES}$ follows. It is routine to check that \mathcal{V}' is a solution to $(G[S \cup C_i], r, \lambda_s^*|_{S \cup C_i}, \alpha' - \alpha'')$ is a YES-instance for LA. \diamond

The next claim, together with Claim 3, asserts that we can correctly solve S-HCLA by computing the tables P_s for $s \in [r]$.

Claim 4. *The given instance (I, S) is YES to S-HCLA if and only if $P_s(\ell, \alpha, c) = \text{YES}$ for some $s \in [r]$ and $c \leq w \cdot f_2(w)$.*

PROOF OF THE CLAIM: To see the forward direction, let \mathcal{V} be a solution to (I, S) of S-HCLA. By definition of S-HCLA there exists an index $j \in [r]$ such that the two conditions **A.** \mathcal{V} is $w \cdot f_2(w)$ -bounded out of s and **B.** $\partial_G(\mathcal{V}^{(s)}) \subseteq S \subseteq \mathcal{V}^{(s)}$ are met. Let $c := \sum_{i \in [r] \setminus \{s\}} |\mathcal{V}^{(i)}|$. Notice that \mathcal{V} is a solution to the instance $(G, r, \lambda_s, \alpha)$ of LA as $S \subseteq \mathcal{V}^{(s)}$, and $c \leq w \cdot f_2(w)$ by Condition **A.** Hence, to prove that $P_s(\ell, \alpha, c) = \text{YES}$, it suffices to verify that $C_j \subseteq \mathcal{V}^{(s)}$ or $C_j \cap \mathcal{V}^{(s)} = \emptyset$ holds for every $1 \leq j \leq \ell$ by Claim 3. Suppose that $C_j \not\subseteq \mathcal{V}^{(s)}$ and $C_j \cap \mathcal{V}^{(s)} \neq \emptyset$ for some j . Then $C_j \cap \mathcal{V}^{(s)}$ contains a vertex of $\partial_G(\mathcal{V}^{(s)})$, and thus contains a vertex of S by **B.** However, this contradicts the fact that C_j and S are disjoint.

For the backward implication, suppose $P_s(\ell, \alpha, c) = \text{YES}$ for some $s \in [r]$ and $c \leq w \cdot f_2(w)$. Then, there exists a solution \mathcal{V} to the instance $(G_\ell, r, \lambda_s, \alpha)$ satisfying the condition of Claim 3. It suffices to show that $\partial_G(\mathcal{V}^{(s)}) \subseteq S \subseteq \mathcal{V}^{(s)}$. If $\partial_G(\mathcal{V}^{(s)}) \setminus S \neq \emptyset$, this means that there exists a vertex sets C of $\mathcal{C}(G \setminus S)$ such that $C \setminus \mathcal{V}^{(s)} \neq \emptyset$ and $C \cap \mathcal{V}^{(s)} \neq \emptyset$, a contradiction to the second condition of Claim 3. The fact that $S \subseteq \mathcal{V}^{(s)}$ is an immediate consequence of the definition of λ_s . \diamond

In the recursion for P_s , verifying (i) takes $O(|C_i|)$ steps and verifying (ii) amounts to solving an instance to LA whose instance size is at most $c \leq w \cdot f_2(w)$. The latter takes $2^{O(w^2 \cdot \log w)}$ using the algorithm of Lemma 5. As the size of each table P_s is $(\ell + 1) \cdot |\mathfrak{F}_\leq(\alpha)| \cdot w \cdot f_2(w)$, we obtain the claimed running time. \square

Composing the running times of Lemmata 6, 8, 10, and 11 and the fact that $f_1(w) = 2^{O(w \cdot \log w)}$ and $f_2(w) = 2^{O(w \cdot \log w)}$ we can derive the correctness of Theorem 3.

5 Further research

In the definition of LIST ALLOCATION we ask for a λ -list H -homomorphism of G where $\sum_{e \in E(H)} |C(e)| \leq \ell$. A different parameterization of LIST ALLOCATION, that is similar in flavor to MIN-MAX MULTIWAY CUT, may instead ask for a λ -list H -homomorphism of G where $\max_{v \in V(H)} \sum_e$ is incident to v $|C(e)| \leq \ell$. We call this new problem MAX BOUNDED LIST DIGRAPH HOMOMORPHISM (in short MBLDH). As it is straightforward to prove an analogue of Theorem 5, where BLDH is now replaced by MBLDH and instead of $2^{O(\ell \log h)} \cdot T(n, \ell)$ steps we now have a reduction that takes $2^{O(\ell^2 \log h)} \cdot T(n, \ell)$ steps. This implies that MBLDH, when parameterized by ℓ and h admits an FPT-algorithm that runs in $2^{O(\ell^2 \cdot \max\{\log \ell, \log h\})} \cdot n^4 \cdot \log n$ steps.

A natural research direction is to improve the running time of our FPT-algorithms for MIN-MAX MULTIWAY CUT and BOUNDED LIST DIGRAPH HOMOMORPHISM. If we want to improve our running times using the techniques used in this paper it seems that we need to crucially improve upon the recursive understanding and randomized contractions technique.

Acknowledgement. We would like to thank the anonymous referees of an earlier version of this paper for their thorough remarks and suggestions that improved the presentation and some proofs of the paper.

References

- [1] N. Bansal, U. Feige, R. Krauthgamer, K. Makarychev, V. Nagarajan, J. S. Naor, and R. Schwartz. Min-max graph partitioning and small set expansion. In *Proc. of the 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 17–26. IEEE Computer Society, 2011.
- [2] R. Brown, I. Morris, J. Shrimpton, and C. D. Wensley. Graphs of morphisms of graphs. *Electronic Journal of Combinatorics*, 15(1), 2008.
- [3] C. Chekuri, S. Guha, and J. Naor. The steiner k -cut problem. *SIAM Journal on Discrete Mathematics*, 20(1):261–271, 2006.
- [4] R. Chitnis, M. Hajiaghayi, and D. Marx. Fixed-parameter tractability of directed multiway cut parameterized by the size of the cutset. In *Proc. of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1713–1725. SIAM, 2012.
- [5] R. H. Chitnis, M. Cygan, M. Hajiaghayi, M. Pilipczuk, and M. Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. In *Proc. of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 460–469. IEEE Computer Society, 2012.

- [6] R. H. Chitnis, L. Egri, and D. Marx. List H -coloring a graph by removing few vertices. In *Proc. of the 21st Annual European Symposium on Algorithms (ESA)*, volume 8125 of *LNCS*, pages 313–324, 2013.
- [7] M. Cygan, D. Lokshtanov, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. Minimum bisection is fixed parameter tractable. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 323–332. ACM, 2014.
- [8] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, (4):864–894, 1994.
- [9] J. Díaz, M. Serna, and D. M. Thilikos. Efficient algorithms for counting parameterized list H -colorings. *Journal of Computer and System Sciences*, 74(5):919–937, 2008.
- [10] J. Díaz, M. J. Serna, and D. M. Thilikos. (H, C, K) -coloring: Fast, easy, and hard cases. In *Proc. of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 2136 of *LNCS*, pages 304–315, 2001.
- [11] J. Díaz, M. J. Serna, and D. M. Thilikos. Fixed parameter algorithms for counting and deciding bounded restrictive list h -colorings. In *Proc. of the 12th Annual European Symposium on Algorithms (ESA)*, volume 3221 of *LNCS*, pages 275–286, 2004.
- [12] L. Egri, P. Hell, B. Larose, and A. Rafiey. Space complexity of list H -colouring: a dichotomy. In *Proc. of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 349–365. SIAM, 2014.
- [13] L. Egri, A. Krokhin, B. Larose, and P. Tesson. The complexity of the list homomorphism problem for graphs. *Theory of Computing Systems*, 51(2):143–178, 2012.
- [14] G. Even, J. Naor, B. Schieber, and M. Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.
- [15] T. Feder, P. Hell, and J. Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19(4):487–505, 1999.
- [16] T. Feder, P. Hell, and J. Huang. Bi-arc graphs and the complexity of list homomorphisms. *Journal of Graph Theory*, 42(1):61–80, 2003.

- [17] N. Garg, V. V. Vazirani, and M. Yannakakis. Multiway cuts in node weighted graphs. *Journal of Algorithms*, 50(1):49–61, 2004.
- [18] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of th ACM*, 42(6):1115–1145, 1995.
- [19] M. Grohe, K. Kawarabayashi, D. Marx, and P. Wollan. Finding topological subgraphs is fixed-parameter tractable. In *Proc. of the 43rd ACM Symposium on Theory of Computing (STOC)*, pages 479–488. ACM, 2011.
- [20] F. Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM Journal on Computing*, 4(3):221–225, 1975.
- [21] P. Hell and J. Nešetřil. *Graphs and homomorphisms*, volume 28 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2004.
- [22] P. Hell and A. Rafiey. The dichotomy of list homomorphisms for digraphs. In *Proc. of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1703–1713. SIAM, 2011.
- [23] K. ichi Kawarabayashi and M. Thorup. The minimum k -way cut of bounded size is fixed-parameter tractable. In *Proc. of the 54th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 160–169. IEEE Computer Society, 2013.
- [24] D. R. Karger, P. Klein, C. Stein, M. Thorup, and N. E. Young. Rounding algorithms for a geometric embedding of minimum multiway cut. In *Proc. of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, pages 668–678. ACM, 1999.
- [25] L. M. Kirousis, M. Serna, and P. Spirakis. Parallel complexity of the connected subgraph problem. *SIAM Journal on Computing*, 22(3):573–586, 1993.
- [26] D. Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394 – 406, 2006.
- [27] D. Marx, B. O’Sullivan, and I. Razgon. Finding small separators in linear time via treewidth reduction. *ACM Transactions on Algorithms*, 9(4):30, 2013.
- [28] D. Marx and I. Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. *SIAM Journal on Computing*, 43(2):355–388, 2014.

- [29] H. Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proc. of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 255–264. ACM, 2008.
- [30] R. Ravi and A. Sinha. Approximating k -cuts via network strength. In *Proc. of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 621–622. SIAM, 2002.
- [31] M. Stoer and F. Wagner. A simple min-cut algorithm. *Journal of the ACM*, 44(4):585–591, 1997.
- [32] Z. Svitkina and É. Tardos. Min-max multiway cut. In *Proc. of the 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX) and 8th International Workshop on Randomization and Computation (RANDOM)*, volume 3122 of *LNCS*, pages 207–218, 2004.