Flow-augmentation II: Undirected graphs*

EUN JUNG KIM, School of Computing, KAIST, South Korea and Centre Nationale de la Recherche Scientifique, France
STEFAN KRATSCH, Humboldt-Universität zu Berlin, Germany
MARCIN PILIPCZUK, University of Warsaw, Poland
MAGNUS WAHLSTRÖM, Royal Holloway, University of London, UK

We present an undirected version of the recently introduced *flow-augmentation* technique: Given an undirected multigraph G with distinguished vertices $s, t \in V(G)$ and an integer k, one can in randomized $k^{O(1)} \cdot (|V(G)| + |E(G)|)$ time sample a set $A \subseteq \binom{V(G)}{2}$ such that the following holds: for every inclusion-wise minimal st-cut Z in G of cardinality at most k, Z becomes a *minimum-cardinality* cut between s and t in $G + A$ (i.e., in the multigraph G with all edges of A added) with probability $2^{-O(k \log k)}$.

Compared to the version for directed graphs [STOC 2022], the version presented here has improved success probability ($2^{-O(k \log k)}$ instead of $2^{-O(k^4 \log k)}$), linear dependency on the graph size in the running time bound, and an arguably simpler proof.

An immediate corollary is that the Bi-objective st-Cut problem can be solved in randomized FPT time $2^{O(k \log k)}(|V(G)| + |E(G)|)$ on undirected graphs.

CCS Concepts: • **Theory of computation → Fixed parameter tractability**.

Additional Key Words and Phrases: fixed-parameter tractability, flow-augmentation, graph separation problems

ACM Reference Format:
Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. 2018. Flow-augmentation II: Undirected graphs. *ACM Trans. Algor.* 1, 1, Article 1 (January 2018), 26 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Fixed-parameter tractable algorithms for graph separation problems has been an important topic in parameterized complexity, and after more than a decade of intense study it would seem that we should by now know of all the major techniques necessary for the design of such algorithms. Certainly, there is an impressive toolbox, leading to the resolution of central problems such as FPT algorithms for Multicut [1, 13] and Minimum Bisection [5].

Yet despite this progress, several open problems remained until very recently. Many of these relate to directed graph cuts, such as the existence of FPT algorithms for the notorious ℓ-Chain SAT problem identified by Chitnis et al. [3], weighted variants of classic problems such as Directed Feedback Vertex Set, or the deceptively simple-looking problem of Bi-objective (s, t)-cut [12]. In the last problem, the input is a digraph $D = (V, A)$ with arc weights w and $s, t \in V$, and two

*A preliminary version of this work was presented at SODA 2021 [8].

Authors' addresses: Eun Jung Kim, eun-jung.kim@dauphine.fr, School of Computing, KAIST, Daejeon, South Korea;, Centre Nationale de la Recherche Scientifique, France; Stefan Kratsch, kratsch@informatik.hu-berlin.de, Humboldt-Universität zu Berlin, Berlin, Germany; Marcin Pilipczuk, malcin@mimuw.edu.pl, University of Warsaw, Warsaw, Poland; Magnus Wahlström, Magnus.Wahlstrom@rhul.ac.uk, Royal Holloway, University of London, Egham, UK.

ACM 1549-6325/2018/1-ART1
https://doi.org/XXXXXXX.XXXXXXX

budgets k, W. The task is to find an (s,t)-cut $Z \subseteq A$ such that $|Z| \leq k$ and $w(Z) \leq W$. Despite the simplicity of the problem, the existence of an FPT algorithm was open for a long time.

This paper is the second one in a series that introduces a new algorithmic technique called *flow-augmentation* and explores its applications. The first part [7, 9] introduced the technique in full generality in directed graphs and applied it to show fixed-parameter tractability of ℓ-Chain SAT and weighted Directed Feedback Vertex Set. The third one [10, 11] uses the technique to show two new tractability isles in the area of parameterized algorithms for Constraint Satisfaction Problems, parameterized by the number of unsatisfied clauses, and completes a complexity dichotomy in the Boolean domain. The main goal of the present second part is to show a counterpart in undirected graphs that is simpler and has improved guarantees as compared to the (more general) directed version of part one [7, 9].

To state formally the main result, we provide some notation. Consider an undirected graph $G = (V,E)$ with two vertices $s,t \in V$, and an unknown (s,t)-cut $Z \subseteq E$. Furthermore, let $Z_{s,t} \subseteq Z$ be those edges with one endpoint reachable from s and the other reachable from t in $G - Z$. We say that Z is a *proper* (s,t)*-cut* if $Z_{s,t}$ is an (s,t)-cut, and *eligible for* (s,t) if additionally every edge of Z has its endpoints in different connected components of $G - Z$. In particular, any minimal, not necessarily minimum (s,t)-cut is eligible for (s,t). Let $k = |Z|$, $\lambda^* = |Z_{s,t}|$, and let $\lambda_G(s,t) \leq \lambda^*$ be the value of an (s,t)-max flow in G. (See Figure 1 for an illustration.) We show the following (reformulated slightly from the more formal version in Section 3).

Theorem 1.1. *There is a randomized algorithm that, given an undirected graph $G = (V,E)$ with $s,t \in V$ and two integers $k \geq \lambda^* \geq \lambda_G(s,t)$, in time $k^{O(1)}(|V| + |E|)$ outputs an edge multiset A with $\lambda_{G+A}(s,t) \geq \lambda^*$ and a flow $\widehat{\mathcal{P}}$ in $G + A$ of cardinality λ^*, such that for any (s,t)-cut Z in G eligible for (s,t) with $|Z| = k$ and $|Z_{s,t}| = \lambda^*$, with probability $2^{-O(k \log k)}$, the following holds: for every $uv \in A$, u and v are connected in $G - Z$; and for every path $P \in \widehat{\mathcal{P}}$, $|E(P) \cap Z| = 1$.*

In particular, in any successful run, in $G + A$ the paths $\widehat{\mathcal{P}}$ will be an (s,t) max-flow, and $Z_{s,t}$ will be an (s,t)-min cut.

A quick comparison of Theorem 1.1 with the directed version of [7] is in order.

- There is a better success probability bound: $2^{-O(k \log k)}$ instead of $2^{-O(k^4 \log k)}$.
- There is an explicit linear dependency on the graph size in the running time bound, instead of just a polynomial of unspeficied degree of [7].
- The notion of an eligible cut is a bit more general than the natural casting of the notion of *star st-cut* of [7] to undirected graphs (it allows some part of $Z \setminus Z_{s,t}$ to separate a bunch of vertices from t, even though these vertices are already separated from s by $Z_{s,t}$).
- The algorithm and the proof is arguably simpler than the one of [7] (albeit it involves a good amount of tedious calculations in the probability analysis to reach the $2^{-O(k \log k)}$ bound).
- While in directed graphs we provided a deterministic counterpart with the expected $2^{O(k^4 \log k)}$ parametric factor in the running time bound, we do not present an analogous result here. All random steps in the presented algorithm can be replaced by with branching or standard derandomization tools for color-coding, so obtaining some deterministic counterpart is definitely possible. However, to achieve $2^{-O(k \log k)}$ success probability we needed to carefully optimise probability distributions in a few places and it is not clear to us that the standard derandomization would match the desired $2^{O(k \log k)}$ parametric factor in the running time bound. Furthermore, the derandomization tools for color-coding steps will introduce a number of $O(\log n)$ factors in the running time analysis, turning the linear dependency on the graph size into a near-linear one $(|V| + |E|)^{1+o(1)}$. We remark also that for any complexity classification results (such those in [10, 11]), the deterministic version of the more general

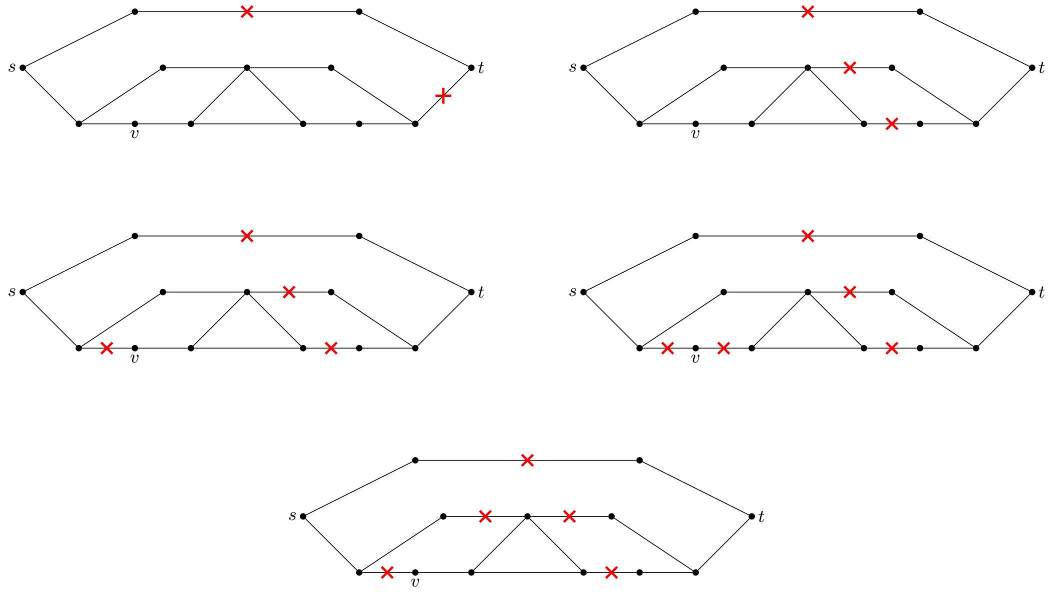

Fig. 1. Examples of (s, t)-cuts, from left to right: a minimum (s, t)-cut, a minimal (s, t)-cut that is not minimum, a proper (s, t)-cut that is not minimal nor eligible (v is reachable from s after the cut), an eligible (s, t)-cut that is not minimal, and an (s, t)-cut that is not proper (the component of v after the cut is not reachable neither from s nor from t).

directed case of [7] is sufficient. Finally, presenting the deterministic counterpart along the randomized proof would significantly cloud the picture.

Recall the Bi-objective (s, t)-Cut problem. Papadimitriou and Yannakakis showed that this is strongly NP-hard, even for undirected graphs, and also showed partial approximation hardness [14]. The directed version, with $\ell \geq 2$ distinct budgets, was recently considered from a parameterized perspective by Kratsch et al. [12], who showed that the problem is FPT if all budgets are included in the parameter, but W[1]-hard if at least two budgets k_i are not included in the parameter. The case of a single budget not being included in the parameter, which includes the Bi-objective (s, t)-Cut problem parameterized by k, has been open prior to our work (in directed graphs).

If k equals the minimum cardinality of an (s, t)-cut, the problem can be easily solved via any polynomial-time minimum cut algorithm: set the capacity of every edge to be a large number (much larger than any weight of an edge) plus the weight of an edge and ask for a minimum capacity cut. Hence, flow-augmentation yields a simple randomized FPT algorithm: We prepend the step above with flow augmentation (Theorem 1.1 in undirected graphs and the version of [7, 9] in directed graphs), with newly added edges assigned prohibitively large weights. For undirected graphs, this gives the following corollary.

Corollary 1.2. *Bi-objective (s, t)-Cut in undirected graphs can be solved in randomized FPT time $2^{O(k \log k)} \cdot (|V(G)| + |E(G)|)$.*

We remark that although it is a quite standard exercise to provide an FPT algorithm for Bi-objective (s, t)-Cut in *undirected graphs* within the framework of randomized contractions [2], and recent improvements would also give $2^{O(k \log k)}$ parametric factor [4], these techniques do not give any explicit bound on the polynomial factor in the running time bound, not to mention guaranteeing a linear one.

2 PRELIMINARIES

In this work we consider only (finite) undirected *multi-graphs without loops*. In particular, different edges connecting the same pair of vertices are considered to be identifiable and non-interchangeable.[1] Formally, a *multi-graph* could be captured as $G = (V, E, \pi)$ where V and E are finite sets and $\pi \colon E \longrightarrow \binom{V}{2}$ assigns each edge in E an unordered pair of endpoints. To keep notation within reason, we will treat multi-graphs as pairs $G = (V, E)$ where V is a finite set and E is a multi-subset of $\binom{V}{2}$ but understanding that cuts X (to be defined in a moment) could involve deleting particular (identifiable) copies of virtually the same edge uv. For a multi-graph G and A a multi-set of edges on V, the graphs $G + A$ and $G - A$ are accordingly understood as starting from G and, respectively, adding all edges in A that are not yet in G or removing from G all edges that are also in A; again, note that this may include different edges with the same two endpoints. For a vertex set S, we denote by $\delta(S)$ the multi-set of edges that have precisely one endpoint in S, and by $\partial(S)$ the set of vertices in S that are incident with at least one edge in $\delta(S)$. By a *connected component* we mean a maximal set $S \subseteq V$ that induces a connected subgraph of G. In all other aspects we follow standard graph notation as set out by Diestel [6].

Throughout this paragraph let $G = (V, E)$ be an arbitrary multi-graph, let $S, T \subseteq V$, and let $X \subseteq E$. Define $R_S(X)$ as the set of vertices that are reachable from any vertex in S in $G - X$. The set X is an *(S, T)-cut* if $R_S(X) \cap R_T(X) = \emptyset$; note that no such cut exists if $S \cap T \neq \emptyset$. A *minimum (S, T)-cut* is any (S, T)-cut of minimum possible cardinality; whereas X is a *minimal (S, T)-cut* if no proper subset of X is an (S, T)-cut. (We will crucially need both minimum and minimal cuts.) By the well-known duality of cuts and flows in graphs (Menger's theorem suffices here) we get that the cardinality of any minimum (S, T)-cut is equal to the maximum number of edge-disjoint paths from S to T in G or, equivalently, to the maximum unit-capacity (S, T)-flow. By $\lambda_G(S, T)$ we denote the maximum flow from S to T or, equivalently, the minimum size of an (S, T)-cut in G; we omit the subscript G when it is clear from context. We mostly apply these notions for the special cases of $S = \{s\}$ and $T = \{t\}$ and then write, e.g., (s, t)-cut rather than $(\{s\}, \{t\})$-cut for succinctness. In particular, we write $\lambda_G(s, t)$ rather than $\lambda_G(\{s\}, \{t\})$ and, when G, s, and t are understood, we usually abbreviate this to λ. We say that an (S, T)-cut X is *closest to S* if for every other (S, T)-cut X' with $R_S(X') \subseteq R_S(X)$ we have $|X'| > |X|$. Clearly, if X is an (S, T)-cut closest to S then X must in particular be minimal.

Let us recall two useful facts about edge cuts in graphs.

Proposition 2.1. *Let X be a minimal (S, T)-cut. Then $X = \delta(R_S(X)) = \delta(R_T(X))$.*

Proof. By definition of $R_S(X)$ we must have $\delta(R_S(X)) \subseteq X$. As X is an (S, T)-cut, we have $T \cap R_S(X) = \emptyset$ and, thus, $\delta(R_S(X))$ is also an (S, T)-cut. Minimality of X now implies that $X = \delta(R_S(X))$; the other equation works symmetrically. □

Proposition 2.2. *There is a unique minimum (S, T)-cut that is closest to S.*

Proof. We use the well-known fact that the cut function $f \colon 2^V \longrightarrow \mathbb{N} \colon Z \mapsto |\delta(Z)|$ is submodular. Suppose that there are two different minimum (S, T)-cuts X and Y that are both closest to S. We

[1]This generality seems necessary to cover a largest set of applications. Multiple copies of the same edge in G might arise in the reduction of some problem to an appropriate cut problem. The different copies may have wildly different behavior regarding contribution to solution cost. Our goal will be to ensure that all solutions of a certain cardinality in terms of cut size have a good probability of being preserved, thereby remaining oblivious to many unnecessary details of the application.

must have $R_S(X) \neq R_S(Y)$ or else $X = \delta(R_S(X)) = \delta(R_S(Y)) = Y$ by Proposition 2.1 as X and Y must be minimal (S,T)-cuts. Using submodularity of f for the sets $R_S(X)$ and $R_S(Y)$ we get

$$|\delta(R_S(X))| + |\delta(R_S(Y))| \geq |\delta(R_S(X) \cap R_S(Y))| + |\delta(R_S(X) \cup R_S(Y))|. \tag{1}$$

Clearly, both $\delta(R_S(X) \cap R_S(Y))$ and $\delta(R_S(X) \cup R_S(Y))$ are (S,T)-cuts and by (1) they must both be minimum (S,T)-cuts. Now, however, because $R_S(X) \neq R_S(Y)$ we must have $R_S(X) \cap R_S(Y) \subsetneq R_S(X)$ or $R_S(X) \cap R_S(Y) \subsetneq R_S(Y)$ contradicting the assumption that X and Y are both closest to S. □

The simple argument used in the proof of Proposition 2.2 is called the *uncrossing of minimum cuts*: From the minimum cuts $X = \delta(R_S(X))$ and $Y = \delta(R_S(Y))$ we obtain minimum cuts $\delta(R_S(X) \cap R_S(Y))$ and $\delta(R_S(X) \cup R_S(Y))$. While the reachable sets $R_S(X)$ and $R_S(Y)$ are in general incomparable, it is clear that $R_S(X) \cap R_S(Y) \subseteq R_S(X) \cup R_S(Y)$, and equality can only hold if $\delta(R_S(X)) = \delta(R_S(Y))$.

3 UNDIRECTED FLOW-AUGMENTATION

Proper cuts, eligible cuts, compatibility, and flow augmentation. Let $G = (V,E)$ be a connected, undirected multi-graph, and let vertices $s,t \in V$. For $Z \subseteq E$, let $Z_{s,t} \subseteq Z$ be the set of edges with one endpoint in $R_s(Z)$ and one endpoint in $R_t(Z)$.

The following notions are crucial for this section.

Definition 3.1 (proper cut). We say that an (s,t)-cut Z is *proper* if $Z_{s,t}$ is an (s,t)-cut. That is, the set of edges $Z_{s,t} \subseteq Z$ with one endpoint in $R_s(Z)$ and one endpoint in $R_t(Z)$ is also an (s,t)-cut.

Note that proper (s,t)-cuts generalize minimal (s,t)-cuts.

In this section, we focus on solutions that are proper (s,t)-cuts with additional technical properties.

Definition 3.2 (eligible cut). We say that an (s,t)-cut Z is *eligible for* (s,t) if

(1) Z is proper,
(2) each edge of Z has its endpoints in different connected components of $G - Z$, and
(3) Z contains no edge incident with s or t.

For an integer λ^*, we say that an (s,t)-cut Z is λ^*-eligible if Z is eligible and additionally $|Z_{s,t}| = \lambda^*$.

We remark that the last property of an eligible cut is only for convenience and is not really a restriction. It can be easily achieved by adding an extra terminal s' connected with s with $k+1$ edges, adding an extra terminal t' connected with t with $k+1$ edges, and asking for (s',t')-cuts instead.

The next two definitions formalize two properties we want from a set of edges that we add to the graph: (i) it does not break the solution, and (ii) it increases the flow from s to t.

Definition 3.3 (compatible set). A multi-subset A of $\binom{V}{2}$ is *compatible* with a set $Z \subseteq E$ if for every $uv \in A$, u and v are connected in $G - Z$.

Definition 3.4 (flow-augmenting set). For an integer $\lambda^* \geq \lambda_G(s,t)$, a multi-subset A of $\binom{V}{2}$ is λ^**-flow-augmenting* if $\lambda_{G+A}(s,t) \geq \lambda^*$.

Intuitively, the role of Z will be played by an unknown solution to the cut problem in question and compatibility of A with Z means that A cannot add connectivity that was removed by Z (or that was not present in the first place). The challenge is to find a flow-augmenting set that with good probability is consistent with at least one solution Z, without knowing Z beforehand.

It will be convenient to take edges in A as being *undeletable* or, equivalently, as unbounded (or infinite) capacity. Clearly, if A is flow-augmenting and compatible with an (eligible) set Z then A

remains flow-augmenting and compatible with Z after adding an arbitrary number of copies of any edges in A. In particular, having a total of $k+1$ copies of every edge in A will make those edges effectively undeletable for sets Z of size k, that is, the endpoints of any edge in A cannot be separated by Z. Note that for applications, since edges in A are in addition to the original input, one will usually not be interested in deleting edges of A anyway (and costs may not be defined), and they only help to increase the flow to match an (unknown) solution. For the purpose of flow and path packings, edges in A may, accordingly, be shared by any number of (flow) paths, fully equivalent to simply having $k+1$ copies of each edge.

Witnessing flow. Similarly as in the directed case, in addition to returning a flow-augmenting set, we will also attempt to return an (s,t)-max flow in the augmented graph which intersects $Z_{s,t}$ in a particularly structured way.

In the following, let G be a connected graph with $s,t \in V(G)$, and let Z be an (s,t)-cut in G which contains an (s,t)-min cut. A *witnessing (s,t)-flow for Z* in G is an (s,t)-max flow $\widehat{\mathcal{P}}$ in G such that every edge of $Z_{s,t}$ occurs on a path of $\widehat{\mathcal{P}}$, and every path of $\widehat{\mathcal{P}}$ intersects Z in precisely one edge.

We make a few observations. First, since Z is an (s,t)-cut, every (s,t)-path in G intersects Z in at least one edge. Second, if additionally $\lambda_G(s,t) = |Z_{s,t}|$, then every (s,t)-max flow in G is witnessing *for* $Z_{s,t}$. Hence, if Z is a minimum (s,t)-cut, then finding a witnessing flow is no harder than finding a flow-augmenting set. However, if Z is a proper and only $Z_{s,t}$ is a minimum (s,t)-cut, then a witnessing flow is a more restrictive notion; see Figure 2 for an illustration.

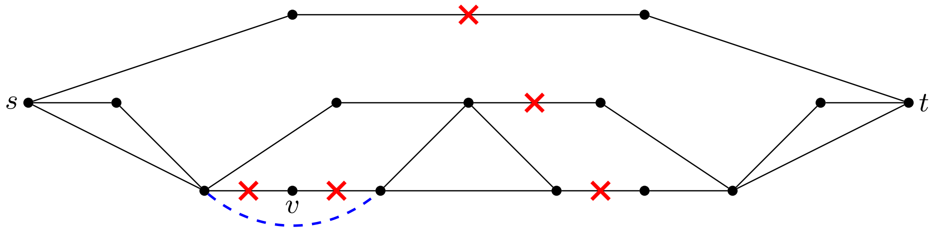

Fig. 2. An example that a witnessing flow is a more restrictive notion than a maximum flow. The red crosses denote an eligible (s,t)-cut with the three edges not incident with v being $Z_{s,t}$, which is an (s,t)-mincut. However, every (s,t)-flow of size 3 needs to use the two edges of Z incident with v, and is thus not a witnessing flow for Z. To obtain a witnessing flow, one needs to augment the graph, for example with the edge denoted by a dashed blue arc.

We now observe that for every proper (s,t)-cut Z, one can augment G with a set compatible with Z such that $Z_{s,t}$ becomes a (s,t)-min cut and $G+A$ admits a witnessing flow for G.

Lemma 3.5. *Let $G=(V,E)$ be a multi-graph, let $s,t \in V$ with $s \neq t$, let $Z \subseteq E$ be a proper (s,t)-cut of size k, and let $\lambda^* = |Z_{s,t}|$. Then there exists a λ^*-flow-augmenting set A compatible with Z and a witnessing flow $\widehat{\mathcal{P}}$ for Z in $G+A$.*

Proof. For each pair u and v of vertices in the same connected component of $G-Z$, add to A a set of $k+1$ copies of the edge uv. Clearly, A is compatible with Z. For every $e = uv \in Z_{s,t}$ with $u \in R_s(Z)$ and $v \in R_t(Z)$, let P_e be a path in $G+A$ consisting of the edges $su \in A$, $uv \in Z_{s,t}$, and $vt \in A$. Then, $\widehat{\mathcal{P}} := \{P_e \mid e \in Z_{s,t}\}$ is a witnessing flow for Z in $G+A$ of cardinality λ^*. Hence, A is λ^*-flow-augmenting. $\square$

A few remarks are in place. The proof of Lemma 3.5 shows that a set $Z \subseteq E$ admits a λ^*-flow-augmenting set A if and only if Z does not contain an (s,t)-cut of cardinality less than λ^*. Indeed, in one direction such a cut $C \subseteq Z$ remains an (s,t)-cut in $G+A$, preventing the flow from increasing above $|C|$, and in the other direction the set A constructed in the proof of Lemma 3.5 is in some sense "maximum possible" and all (s,t)-cuts of cardinality at most k in $G+A$ are contained in Z. Furthermore, even if Z is a proper (s,t)-cut where $Z_{s,t}$ is an (s,t)-min cut (so no flow increase is possible), while Z may not admit a witnessing flow in G, it is possible to augment G with a set of edges compatible with Z so that a witnessing flow exists.

Lemma 3.5 motivates the following extension of the definition of compatibility.

Definition 3.6 (compatible pair). A pair $(A, \widehat{\mathcal{P}})$ is *compatible* with a proper (s,t)-cut Z if A is a λ^*-flow-augmenting set compatible with Z for $\lambda^* = |Z_{s,t}|$ and $\widehat{\mathcal{P}}$ is a witnessing flow for Z in $G+A$.

Problem formulation. The proof of Lemma 3.5 shows that the task of finding a compatible flow-augmenting set and a witnessing flow would be trivial if only we knew Z in advance. Not knowing Z, we will have to place additional edges more sparingly than in the proof of Lemma 3.5 to arrive at a sufficient success probability. Let us formally define our goal, taking into account that the set Z is not known.

In the FLOW-AUGMENTATION SAMPLING problem we are given an instance (G,s,t,k,λ^*) consisting of an undirected connected multi-graph $G=(V,E)$, vertices $s,t \in V$, and integers k and λ^* such that $k \geq \lambda^* \geq \lambda := \lambda_G(s,t)$. The goal is to find (in probabilistic polynomial-time) a multi-set A of $\binom{V}{2}$ and an (s,t)-flow $\widehat{\mathcal{P}}$ in $G+A$ such that the following holds:

- $\lambda_{G+A}(s,t) \geq \lambda^*$, $|\widehat{\mathcal{P}}| = \lambda^*$, and
- for each λ^*-eligible (s,t)-cut Z of size exactly k, the output $(A, \widehat{\mathcal{P}})$ is compatible with Z with probability at least p.

The function p (that may depend on k or λ) is called the *success probability*.

In order to relax some corner cases, we allow for the event that $\lambda_{G+A}(s,t) > \lambda^*$, and note that if Z is an eligible (s,t)-cut with $|Z_{s,t}| = \lambda^*$ then for any such output $(A, \widehat{\mathcal{P}})$ such that A is compatible with Z we must have $\lambda_{G+A}(s,t) = \lambda^*$.

Results. We can now formulate the main result of this section.

THEOREM 3.7. *There is a randomized polynomial-time algorithm that, given a* FLOW-AUGMENTATION SAMPLING *instance (G,s,t,k,λ^*) with $\lambda_G(s,t) \leq \lambda^* \leq k$, outputs a set A with $\lambda_{G+A}(s,t) \geq \lambda^*$ and a flow $\widehat{\mathcal{P}}$ in $G+A$ of cardinality λ^*, such that the following holds: for each set $Z \subseteq E$ of size k that is λ^*-eligible for (G,s,t,k), the output A is compatible with Z and $\widehat{\mathcal{P}}$ is a witnessing flow for Z in $G+A$ with success probability $2^{-O(k \log k)}$. The algorithm can be implemented to run in time $k^{O(1)}O(m)$.*

The rest of the section is devoted to proving Theorem 3.7. For clarity, we will only argume polynomial-time running time bound through the proof, and only discuss how to reach $k^{O(1)}O(m)$ bound in the end.

We begin by introducing an appropriate decomposition of (the vertex set of) G into what we call *bundles*, which in turn consist of what is called *blocks*. We then present our recursive flow-augmentation algorithm, splitting the presentation into an "outer loop" and an "inner loop." Note that in FLOW-AUGMENTATION SAMPLING we assume that the input multi-graph G is connected as this somewhat simplifies presentation, but we will circumvent this assumption in applications. (As a side remark, observe that if G is disconnected then the task of Theorem 3.7 is trivial if s and t are in different connected components, and if s and t are in the same connected component, one can focus only on the said component. Thus we do not lose anything interesting by restricting to connected inputs.)

3.1 Blocks and bundles

Given an instance (G, s, t, k, λ^*) of FLOW-AUGMENTATION SAMPLING, it should come as no surprise that the minimum (s, t)-cuts of G will be crucial for flow augmentation. Recall, however, that even structurally simple graphs may exhibit an exponential number of possibly crossing minimum (s, t)-cuts. We will use the notion of closest cuts (and implicitly the well-known uncrossing of minimum (s, t)-cuts as used in Proposition 2.2) to identify a sequence of non-crossing minimum (s, t)-cuts. The parts between consecutive cuts will be called blocks; we will also define a partition of blocks into consecutive groups called bundles. The decomposition of G into bundles will guide the choice of edges for the flow-augmenting set A in our algorithm and will be used to capture parts of G to recurse on.

For convenience, let us fix an instance (G, s, t, k, λ^*) and let $\lambda := \lambda_G(s, t) \leq k$ for use in this subsection. Accordingly, in G there is a packing of λ edge-disjoint (s, t)-paths $P_1, \ldots, P_\lambda$ (and no larger packing exists). Clearly, every minimum (s, t)-cut in G contains exactly one edge from each path P_j and no further edges. As noted earlier, we assume for now that G is connected.

Blocks. We first define a sequence $C_0, \ldots, C_p$ of non-crossing minimum (s, t)-cuts; recall that minimum (s, t)-cuts in G all have cardinality λ. To start, let C_0 be the unique minimum (s, t)-cut that is closest to s. Inductively, for $i \geq 1$, let C_i be the minimum (s, t)-cut closest to s among all cuts that fulfil $N[R_s(C_{i-1})] \subseteq R_s(C_i)$. The cut C_i is well-defined (i.e., unique) by an easy variant of Proposition 2.2: Minimum cuts X fulfilling the requirement that $N[R_s(C_{i-1})] \subseteq R_s(X)$ uncross into minimum cuts fulfilling the same requirement. Intuitively, the construction is equivalent to asking that each C_i is closest to s among minimum (s, t)-cuts that do not intersect $C_0 \cup \ldots \cup C_{i-1}$ but this would need a formal proof and we do not require it.

We can now define the *blocks* $V_0, \ldots, V_{p+1} \subseteq V$, which will be seen to form a partition of V. Block V_0 is simply set to $R_s(C_0)$. For $i \in \{1, \ldots, p\}$, we define block V_i as the set of vertices reachable from s in $G - C_i$ but not in $G - C_{i-1}$, i.e., $V_i := R_s(C_i) \setminus R_s(C_{i-1})$. Finally, V_{p+1} contains all vertices reachable from s in G but not in $G - C_p$ which, since G is connected, equates to $V_{p+1} = V \setminus R_s(C_p)$. By construction of the cuts C_i we clearly have $s \in R_s(C_0) \subsetneq R_s(C_1) \subsetneq \ldots \subsetneq R_s(C_p) \subseteq V \setminus \{t\}$, so the blocks V_i are all nonempty and clearly form a partition of V; see Figure 3.

Let us point out that blocks V_i do not need to be connected even though G is connected. It will be useful to note, however, that blocks V_0 and V_{p+1} both are connected: The graph G is connected and each minimum (s, t)-cut C_i will therefore separate it into exactly two connected components $R_s(C_i)$ and $R_t(C_i)$. Blocks $V_0 = R_s(C_0)$ and $V_{p+1} = V \setminus R_s(C_p) = R_t(C_p)$ are therefore connected. Moreover, each block is at least somewhat connected through subpaths of the flow paths $P_1, \ldots, P_\lambda$ that are contained therein. We establish a bit more structure via the following two propositions.

PROPOSITION 3.8. *For each (s, t)-flow path $P_j \in \{P_1, \ldots, P_\lambda\}$, seen as being directed from s to t, the edges of the minimum (s, t)-cuts $C_0, \ldots, C_p$ appear in order of the cuts. These edges define a partition of the flow path P_j into $P_j^0, \ldots, P_j^{p+1}$ so that P_j^i is contained in block V_i for $i \in \{0, \ldots, p+1\}$.*

PROOF. Fix an (s, t)-flow path P_j and let e_i denote the unique edge of P_j that is contained in the minimum (s, t)-cut C_i, for all $i \in \{0, \ldots, p\}$. Moreover, let u_i and v_i be the endpoints of e_i in order of appearance on P_j; note that $v_i = u_{i+1}$ is possible. (We tacitly assume that the flow paths are cycle free.)

Clearly, the vertices of the subpath P_j^0 of P_j from s to u_0 are contained in $R_s(C_0)$ because no further edge of P_j is in C_0. Thus, all vertices of P_j up to and including v_0 are contained in $N[R_s(C_0)] \subseteq R_s(C_1)$. Accordingly, the edge $e_1 \in C_1$ on P_j must be part of the subpath from v_0 to t or else, combined with $v_0 \in N[R_s(C_0)] \subseteq R_s(C_1)$ there would be a path from s to t. Thus, e_1 appears after e_0 on P_j, when seeing P_j as directed from s to t. Observe that e_0 and e_1 together define

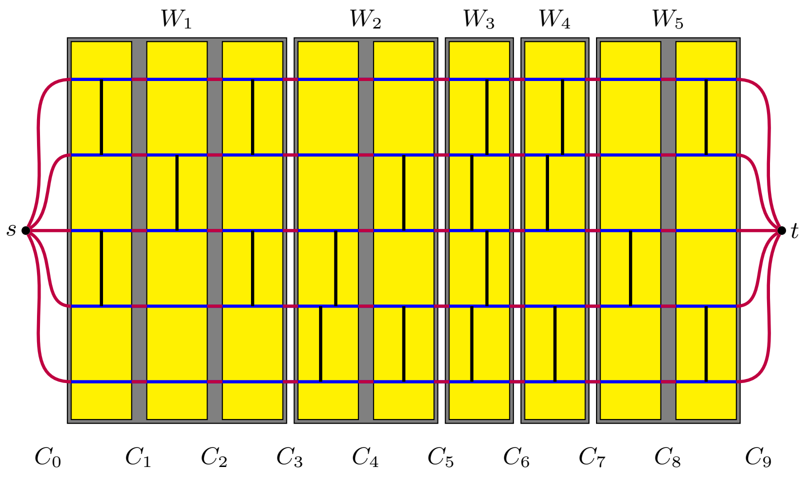

Fig. 3. A schematic picture of nine blocks (yellow) partitioned into five bundles (gray). The extremal blocks and bundles, containing s and t, are not depicted. The $s-t$ flow of value 5 is depicted using blue flow paths, with purple edges being the edges of the consecutive cuts C_i. The bundle W_3 is a connected bundle.

a subpath P_j^1 from v_0 to u_1 on P_j that is contained in $R_s(C_1) \setminus R_s(C_0) = V_1$. Iterating this argument for increasing i completes the proof. (Note that the final subpath of P_j, denoted P_j^{p+1} starts with v_p and ends in t. Clearly, it is contained in $V \setminus R_s(C_p)$.) □

Using the fact that, for each (s, t)-flow path P_j, the blocks V_i contain consecutive subpaths of P_j, we can prove that each block has at most λ connected components. Moreover, each such component in a block V_i, with $i \in \{1, \ldots, p\}$ is incident with some number of edges of C_{i-1} and the same number of edges in C_i.

Proposition 3.9. *Each block V_i has at most λ connected components. Moreover, each connected component in a block V_i, with $i \in \{1, \ldots, p\}$, is incident with $c \geq 1$ edges in C_{i-1} and with exactly c edges in C_i. (Clearly, V_0 is incident with all λ edges of C_0, and V_{p+1} is incident with all λ edges of C_p.)*

Proof. We already know that V_0 and V_{p+1} are connected. Consider now a block V_i with $i \in \{1, \ldots, p\}$. Clearly, $R_s(C_i)$ is connected in $G - C_i$ because all of its vertices are reachable from s. At the same time, the vertices in $V_i \subseteq R_s(C_i)$ are not reachable from s in $G - C_{i-1}$ by definition of $V_i = R_s(C_i) \setminus R_s(C_{i-1})$, so paths from s to V_i must use at least one edge in C_{i-1}. Thus, each connected component K of V_i must be incident with at least one edge of C_{i-1}; let $c \geq 1$ be the number of such edges. Now, recall that the edges in C_{i-1} together with those in C_i define the subpaths of $P_1, \ldots, P_\lambda$ that are in block V_i. This implies that the c edges of C_{i-1} that are incident with component K in block V_i correspond to exactly c subpaths of paths $P_j \in \{P_1, \ldots, P_\lambda\}$ that are part of component K. This of course implies that K must be incident by the c edges of C_i that define those paths. Since connected components of V_i do not share vertices and edges of C_{i-1} and C_i have exactly one endpoint in V_i each, no two connected components of block V_i can share their incident edges in C_{i-1} or C_i. Thus, there are at most λ connected components in each block V_i. This completes the proof. □

It can be easily verified that the decomposition into blocks can be computed in polynomial time.

Proposition 3.10. *Given a multi-graph $G = (V, E)$ and vertices $s, t \in V$, the unique sequence of cuts $C_0, \ldots, C_p$ and decomposition of blocks $V_0, \ldots, V_{p+1}$ can be computed in polynomial time.*

Proof. This comes down to computing a polynomial number of closest minimum cuts. A closest minimum (S, T)-cut C can be computed by a standard maximum (unit capacity) flow algorithm based on maintaining a residual graph: When the maximum (S, T)-flow is reached, let $R \supseteq S$ be the set of vertices that are reachable from S in the residual graph. Clearly, when viewing packed paths as being directed from S to T, there is no path edge entering R from $V \setminus R$ because that would yield an edge leaving R in the residual graph. As the flow is maximum, T may not be reachable in the residual graph, so $R \cap T = \emptyset$. Consequently, each path in the packing leaves R exactly once and does not return. Thus, the cardinality of C is equal to the number of paths in the packing, say λ, making it a minimum (S, T)-cut.

To see that C is closest to S, assume that there was a minimum (S, T)-cut $C' \neq C$ with $R_S(C') \subseteq R_S(C)$. Since both C and C' are also minimal cuts, $R_S(C') = R_S(C)$ would imply $C' = C$ by Proposition 2.1, so assume that $R_S(C') \subsetneq R_S(C)$ and let $v \in R_S(C) \setminus R_S(C')$. Since the cardinality of C' is equal to the size of the path packing, all of its edges are used by (S, T)-paths leaving $R_S(C')$. Thus, in the residual graph, there is no edge leaving $R_S(C')$ and hence no path from $S \subseteq R_S(C')$ to $v \notin R_S(C')$; a contradiction. □

Bundles. We will now inductively define a decomposition of V into *bundles* $W_0, \ldots, W_{q+1}$; see also Figure 3. The first bundle W_0 is simply equal to the (connected) block V_0, which contains s. For $i \geq 1$, supposing that blocks $V_0, \ldots, V_{j-1}$ are already parts of previous bundles,

- let $W_i := V_j$ if V_j is connected (i.e., if $G[V_j]$ is connected) and call it a *connected bundle*
- otherwise, let $W_i := V_j \cup \ldots \cup V_{j'}$ be the union of contiguous blocks, where j' is maximal such that $G[V_j \cup \ldots \cup V_{j'}]$ is *not* connected and call it a *disconnected bundle.*

Observe that the final bundle is $W_{q+1} = V_{p+1}$ because V_{p+1} is connected and, due to the included subpaths of (s, t)-flow paths (cf. Proposition 3.9), any union $V_j \cup \ldots \cup V_{p+1}$ induces a connected graph (see also Proposition 3.11). We use $\mathbf{B}(W_i)$ to denote the set of blocks whose union is equal to W_i, i.e., $\mathbf{B}(W_i) = \{V_j\}$ and $\mathbf{B}(W_i) = \{V_j, \ldots, V_{j'}\}$ respectively in the two cases above. We say that two bundles W_i and $W_{i'}$ are *consecutive* if $|i - i'| = 1$.

Intuitively, bundles are defined as maximal sequences of blocks that permit a good argument to apply recursion in our algorithm. In case of a single block, if we augment the edges incident with the block, then in the recursive step the cardinality of the maximum flow $\lambda_G(s, t)$ increases. In case of a union of contiguous blocks that does not induce a connected subgraph, if we recurse into every connected component independently, we split the budget k in a nontrivial way, as every connected component contains the appropriate part of at least one flow path of $\mathcal{P}$.

Clearly, the bundles $W_0, \ldots, W_{q+1}$ are well defined and they form a partition of the vertex set V of G. We emphasize that $W_0 = V_0 \ni s$ and $W_{q+1} = V_{p+1} \ni t$ and that they are both connected bundles. We note without proof that the bundles inherit the connectivity properties of blocks because the cuts between blocks combined into a bundle connect their subpaths of (s, t)-flow paths $P_1, \ldots, P_\lambda$ into longer subpaths, whereas the incidence to the preceding and succeeding cuts stays the same (see Proposition 3.11). For ease of reference, let us denote by $C'_0, \ldots, C'_q$ those cuts among $C_0, \ldots, C_p$ that have endpoints in two different (hence consecutive) bundles, concretely, with C'_i having endpoints in both W_i and W_{i+1}; note that $C'_0 = C_0$ as $W_0 = V_0$ and $C'_q = C_p$ as $W_{q+1} = V_{p+1}$.

Proposition 3.11. *Each bundle W_i has at most λ connected components. Moreover, each connected component in a bundle W_i, with $i \in \{1, \ldots, q\}$, is incident with $c \geq 1$ edges in C'_{i-1} and with c edges in C'_i. (Clearly, $W_0 = V_0$ is incident with all λ edges of $C'_0 = C_0$, and $W_{q+1} = V_{p+1}$ is incident with all λ edges of $C'_q = C_p$.)*

Let us introduce some more notation for bundles: For $0 \leq a \leq b \leq q+1$ let $W_{a,b} := \bigcup_{i=a}^{b} W_i$. Let $W_{\leq a} := W_{0,a}$ and $W_{\geq a} := W_{a,q+1}$. For any (union of consecutive bundles) $W_{a,b}$ we define the *left interface* $\mathrm{left}(W_{a,b})$ as $\partial(W_{\geq a}) \cap W_{\geq a}$ when $a \geq 1$ and as $\{s\}$ when $a = 0$. (I.e., when $a \geq 1$ then $\mathrm{left}(W_{a,b})$ are those vertices of $W_{a,b}$ that are incident with the cut C'_{a-1} that precedes bundle W_a). Similarly, we define the *right interface* $\mathrm{right}(W_{a,b})$ as $\partial(W_{\leq b}) \cap W_{\leq b}$ when $b \leq q$ and as $\{t\}$ when $b = q+1$. (I.e., when $b \leq q$ then $\mathrm{right}(W_{a,b})$ are those vertices of $W_{a,b}$ that are incident with the cut C'_b that succeeds bundle W_b.) For single bundles W_i the same notation applies using $W_i = W_{i,i}$. A consecutive subsequence of bundles is called a *stretch* of bundles, or simply a stretch.

While a union of consecutive blocks may be disconnected, this is not true for bundles where, as can be easily checked, any two consecutive bundles together induce a connected subgraph of G.

Proposition 3.12. *For any two consecutive bundles W_i and W_{i+1} the graph $G[W_i \cup W_{i+1}]$ is connected.*

Proof. If W_i is a connected bundle then, using Proposition 3.11, we immediately get that $G[W_i \cup W_{i+1}]$ is connected. If W_i is a disconnected bundle then $G[W_i \cup V_{j'+1}]$ is connected, where $W_i = V_j \cup \ldots \cup V_{j'}$ and $V_{j'+1}$ is the first block of W_{i+1}. Using Propositions 3.9 and 3.11 we again directly get that adding the remaining blocks of W_{i+1} to $G[W_i \cup V_{j'+1}]$ does not break connectivity. □

Clearly, the decomposition into bundles can be efficiently computed from the one into blocks.

Proposition 3.13. *Given a multi-graph $G = (V, E)$ and vertices $s, t \in V$, the unique sequence of cuts $C'_0, \ldots, C'_q$ and decomposition of bundles $W_0, \ldots, W_{q+1}$ can be computed in polynomial time.*

Affected and unaffected bundles. We will later need to reason about the interaction of a proper (s,t)-cut $Z \subseteq E$ and $G = (V, E)$ and, hence, about the interaction with the bundles of G. We say that a bundle W is *unaffected by Z* if $N[W]$ is contained in a single connected component of $G - Z$; otherwise we say that W is *affected by Z*. As an example, the cut $Z = C'_i$ affects both W_i and W_{i+1} but no other bundles. Similarly, a cut Z entirely confined to $G[W_i]$ affects only W_i, since $W_{\leq i-1}$ and $W_{\geq i+1}$ are both connected and do not contain an endpoint of an edge of Z. The more interesting/difficult cuts Z affect several bundles in a non-trivial way.

The following observation limits the number and arrangement of affected bundles. It will be important for reducing the general case (probabilistically) to the case where G decomposes into a bounded number of bundles. Concretely, this is the purpose of the outer-loop part of our algorithm, which is presented in the following section.

Lemma 3.14. *Let $Z \subseteq E$ be an (s,t)-cut of size at most k. Let $0 \leq a \leq b \leq q+1$ and let ℓ be the number of indices $a \leq i \leq b$ such that the bundle W_i is affected. Then,*

$$\ell \leq 2\left|E(G[W_{a-1,b+1}]) \cap Z\right|.$$

In particular, at most $2k$ bundles are affected by Z.

Proof. We argue that a bundle $W_i \in \mathcal{W}$ is unaffected if Z contains no edge from C'_i and no edge with both endpoints in $W_{i-1} \cup W_i$ (Condition $\star$). First of all, we have $G[W_{i-1} \cup W_i] - Z = G[W_{i-1} \cup W_i]$ in this case, which is connected by Proposition 3.12 and all vertices of $W_{i-1} \cup W_i$ are in the same component of $G - Z$. As Z is furthermore disjoint from the min-cut C'_i succeeding W_i, all vertices of $N[W_i]$ are in the same connected component. Thus, W_i is not affected. A symmetric argument holds for W_i if Z contains no edge of C'_{i-1} and no edge with both endpoints in $W_i \cup W_{i+1}$ (Condition †).

To bound the number of affected bundles, first note that W_0 and W_{q+1} are affected only if Z contains an edge of $G[W_0]$ or C_0, respectively of $G[W_{q+1}]$ or C'_q. For any remaining bundle W_i, say that W_i is *potentially affected* if neither of the conditions ($\star$) and (†) in the previous paragraph applies. A bundle W_i, $i \in [q]$ is thus potentially affected if (1) Z contains an edge of $G[W_i]$ or intersects $C'_{i-1} \cup C'_i$, or (2) Z contains both an edge of $G[W_{i-1}]$ and of $G[W_{i+1}]$. For each potentially affected bundle W_i, let us charge some edge of Z as described by 1 point if the former case applies, and otherwise charge one edge of Z in $G[W_{i-1}]$ and in $G[W_{i+1}]$ by 1/2 point each. Then the total amount of allocated charge equals the number of potentially affected bundles, and it can be seen that every edge is charged at most 2 points. The lemma follows. □

Lemma 3.15. *Let $Z \subseteq E$ be an (s,t)-cut. There is at most one maximal stretch $W_{a,b}$ of bundles such that every bundle W_i, $a \leq i \leq b$ is affected by Z and such that $W_{a,b}$ contains both a vertex reachable from s and a vertex reachable from t (in $G - Z$). Moreover, all vertices in the left interface of $W_{a,b}$ are reachable from s and all vertices in the right interface are reachable from t. Finally, if Z is a proper (s,t)-cut then there must be such a stretch.*

Proof. If there is no such stretch of bundles then the lemma holds vacuously. Else, let $W_{i,j}$ be a maximal stretch of consecutive affected bundles such that $W_{i,j}$ contains a vertex p that is reachable from s in $G - Z$ and a vertex q that is reachable from t in $G - Z$. Because the left interface X of the stretch separates the stretch from s, at least one vertex of X must be reachable from s in $G - Z$. The preceding bundle, namely W_{i-1}, is unaffected (by choosing the stretch as maximal). Since $N[W_{i-1}] \supseteq X$ contains a vertex reachable from s in $G - Z$ it follows that all vertices in $N[W_{i-1}]$ are reachable from s in $G - Z$. Since Z separates s from t, no vertex is reachable from both s and t in $G - Z$. In particular, this holds for all vertices in X and, hence, for all bundles preceding W_i. By a symmetric argument, all vertices in the right interface of the stretch, say Y, are reachable from t but not from s.

For the final part of the lemma, consider any flow path P from s to t in G. Since $Z_{s,t} \subseteq Z$ is also an (s,t)-cut, the path P must contain some edge $e \in Z_{s,t}$. By definition of $Z_{s,t}$, e has one endpoint reachable from s in $G - Z$ and one reachable from t in $G - Z$. Clearly, there is a bundle W with both endpoints of e in $N[W]$ (as the bundles are a partition of the vertex set, some bundle W contains an endpoint of e and, hence, $N[W]$ must contain both endpoints). This bundle W is affected and contains a vertex reachable from s and one reachable from t in $G - Z$, namely the endpoints of e. By definition of stretch, W is contained in precisely one maximal stretch of affected bundles, and this stretch meets the conditions of the lemma. □

The previous lemma says that each proper (s,t)-cut Z yields exactly one maximal stretch of affected bundles in which it separates s from t (and possibly creates further connected components). We say that Z *strongly affects* that stretch. For all other maximal affected stretches of bundles we say that they are *weakly affected by Z*. Note that a non-proper cut such as $C'_i \cup C'_j$ for $j \geq i + 3$ may contain no strongly affected stretch.

Let us make some useful observations about bundles not in the strongly affected stretch.

Proposition 3.16. *Let Z be a proper (s,t)-cut and let $W_{a,b}$ be the unique strongly affected stretch. Then the following hold.*

(1) *For every $i < a$, if W_i is an unaffected bundle then $W_i \subseteq R_s(Z)$.*
(2) *For every $i > b$, if W_i is an unaffected bundle then $W_i \subseteq R_t(Z)$.*
(3) *If $W_{i,j}$ is a (maximal) weakly affected stretch with $j < a$, then*

$$\mathrm{left}(W_i) \cup \mathrm{right}(W_j) \subseteq R_s(Z) \quad \text{and} \quad (j - i + 1) \leq 2|Z \cap E(G[W_{i,j}])|.$$

(4) *If $W_{i,j}$ is a (maximal) weakly affected stretch with $i > b$, then*

$$\mathrm{left}(W_i) \cup \mathrm{right}(W_j) \subseteq R_t(Z) \quad \text{and} \quad (j - i + 1) \leq 2|Z \cap E(G[W_{i,j}])|.$$

3.2 The outer loop of the algorithm

Our algorithm Sample consists of an outer loop (to be explained in this section), which is applied first to an input instance (G, s, t, k, λ^*) and also to certain instances in recursive calls, and an inner loop, which is applied only to short sequences of bundles. The outer loop part uses a color-coding approach to guess weakly and strongly affected stretches of bundles in G, and calls the inner-loop subroutine called Short-separation on the latter. This subroutine (to be described in detail in the following section) then seeks to recursively find an output $(A, \widehat{\mathcal{P}})$, using the assumption that whenever it is called on a stretch $W_{a,b}$, then either Z is disjoint from the stretch $W_{a,b}$ or $W_{a,b}$ is precisely the unique strongly affected stretch in G.

Each call to our algorithm will return a pair $(A, \widehat{\mathcal{P}})$ for the instance in question, where $(A, \widehat{\mathcal{P}})$ may or may not be compatible for an arbitrary (unknown) (s, t)-cut Z. A crucial observation for the correctness of our algorithm is that any flow-augmentation set guessed for an unaffected stretch of bundles will always be compatible with Z. This allows us to focus our attention in the analysis on the guesses made while processing affected bundles. This is essential in bounding the success probability purely in terms of k.

We will argue that for some sufficiently large constants $c_1 \gg c_2 \gg 0$, Sample(G, s, t, k, λ^*) returns an output $(A, \widehat{\mathcal{P}})$ which is, with probability at least $e^{-g(\lambda_G(s,t),k)}$, compatible with an (unknown) eligible (s, t)-cut Z, where $g(\lambda, k) = (c_1 k - c_2)(1 + \ln k) + c_2 \max(0, k - \lambda)$.

The main (outer loop) algorithm is shown in Figure 4.

Interface of the inner loop algorithm. The inner-loop algorithm expects as input an instance $(G', s', t', k', \lambda')$ that has two additional properties and will return a pair $(A', \widehat{\mathcal{P}}')$. A *valid input* $(G', s', t', k', \lambda')$ *for the inner loop algorithm* has the following properties:

(1) The graph G' decomposes into bundles $W'_0, \ldots, W'_{q+1}$, with $1 \le q \le 2k'$, and such that $W'_0 = \{s'\}$ and $W'_{q+1} = \{t'\}$. If $q = 1$, then we say that the instance is a *single-bundle* instance, otherwise if $q > 1$ it is a *multiple-bundle* instance.
(2) We have $\lambda_{G'}(s', t') < \lambda' \le k'$, i.e., the maximum (s', t')-flow in G' is lower than the target flow value λ' after augmentation.

Furthermore, let Z' be an (s', t')-cut in G'. We say that Z' is a *valid cut for* $(G', s', t', k', \lambda')$ if the following hold.

(1) Z' is an eligible (s', t')-cut in G' with $|Z'| = k$ and $|Z'_{s,t}| = \lambda'$;
(2) Z' affects precisely the bundles $W'_1, \ldots, W'_q$ in G'

In the following section we will describe a realization of this interface by two algorithms called Short-separation-single and Short-separation with the following success guarantee:

- for a valid single-bundle instance $(G', s', t', k', \lambda')$, the algorithm Short-separation-single returns a flow-augmenting set A' with $\lambda_{G'+A'}(s', t') \ge \lambda'$ and an (s, t)-flow $\widehat{\mathcal{P}}'$ in $G + A$ of size λ' such that for every valid cut Z', $(A', \widehat{\mathcal{P}}')$ is compatible with Z' with probability at least $32 \cdot e^{-g(\lambda_{G'}(s',t'),k')}$;
- for a valid multiple-bundle instance $(G', s', t', k', \lambda')$, the algorithm Short-separation returns a flow-augmenting set A' with $\lambda_{G'+A'}(s', t') \ge \lambda'$ and an (s, t)-flow $\widehat{\mathcal{P}}'$ in $G + A$ of size λ' such that for every valid cut Z', $(A', \widehat{\mathcal{P}}')$ is compatible with Z' with probability at least $32(k')^3 \cdot e^{-g(\lambda_{G'}(s',t'),k')}$.

Correctness of the outer loop part. We are now ready to prove correctness of the outer loop algorithm Sample assuming a correct realization of the inner loop algorithm according to the interface stated above.

```
Algorithm Sample($G, s, t, k, \lambda^*$)
(1) If it does not hold that $\lambda_G(s,t) \leq \lambda^* \leq k$, then set $A$ to be $\max(k+1, \lambda^*)$ copies of $\{s,t\}$, $\widehat{\mathcal{P}}$ to be any
    $\lambda^*$ of these copies, and return $(A, \widehat{\mathcal{P}})$.
(2) Initialize $A = \emptyset$ and $\widehat{\mathcal{P}}$ to be a set of $\lambda^*$ zero-length paths starting in $s$.
(3) Compute the partition $V = W_0 \cup \ldots \cup W_{q+1}$ of $G$ into bundles.
(4) Go into single mode or multiple mode with probability $1/2$ each.
    • In single mode, set $p_{\mathrm{blue}} = p_{\mathrm{red}} = 1/2$.
    • In multiple mode, set $p_{\mathrm{blue}} = 1/k$, $p_{\mathrm{red}} = 1 - 1/k$.
(5) Randomly color each bundle blue or red; blue with probability $p_{\mathrm{blue}}$ and red with probability $p_{\mathrm{red}}$.
(6) Randomly sample an integer $\lambda^* \leq k' \leq k$ as follows: set $k' = k$ with probability $1/2$ and with remaining
    probability sample $\lambda^* \leq k' < k$ uniformly at random.
(7) For every maximal stretch $W_{a,b}$ of bundles colored with the same color, do the following in consecutive
    order starting with $a = 0$, and maintaining the property that at the begining of the loop $\widehat{\mathcal{P}}$ is a family of
    $\lambda^*$ edge-disjoint paths in $G + A$ starting in $s$ and ending in $\mathrm{left}(W_a)$:
  (a) If $a > 0$, then add to $A$ all edges $uv$ for $u, v \in \mathrm{right}(W_{a-1}) \cup \mathrm{left}(W_a)$; (We henceforth refer to the
      edges added in this step as *link edges.*)
  (b) If the stretch is colored red and consists of one bundle in single mode, or at least two and at most $2k'$
      bundles in multiple mode, then perform the following:
    (i) Let $G'$ be the graph $G[N[W_{a,b}]]$ with vertices of $W_{\leq a-1}$ contracted to a single vertex $s'$ and vertices
        of $W_{\geq b+1}$ contracted to a single vertex $t'$. If $a = 0$, and hence $W_{\leq a-1} = \emptyset$, then instead add a new
        vertex $s'$ and connect it to $s \in W_a$ via $\lambda$ parallel edges $\{s, s'\}$. Similarly, if $b = q + 1$ then $W_{\geq b+1} = \emptyset$
        and we instead add a new vertex $t'$ and connect it to $t \in W_b$ via $\lambda$ parallel edges $\{t, t'\}$. Observe
        that $\deg(s') = \deg(t') = \lambda$.
    (ii) Do a recursive call:
        • In single mode, let $(A', \widehat{\mathcal{P}}') \longleftarrow$ Short-separation-single($G', s', t', k', \lambda^*$).
        • In multiple mode, let $(A', \widehat{\mathcal{P}}') \longleftarrow$ Short-separation($G', s', t', k', \lambda^*$).
    (iii) Update $A$ as follows:
        • Add to $A$ all edges of $A'$ that are not incident with $s'$ or $t'$.
        • For every edge $s'v \in A'$, add to $A$ a separate edge $uv$ for each vertex $u \in \mathrm{right}(W_{\leq a-1})$. If $a = 0$
          then ignore edges $s's \in A'$ and for each edge $s'v \in A'$ add $sv$ to $A$;
        • Analogously, for every edge $vt' \in A'$, add to $A$ a separate edge $vw$ for each vertex $w \in \mathrm{left}(W_{b+1})$.
          If $b = q + 1$ then ignore edges $tt' \in A'$ and for each edge $vt' \in A'$ add $vt$ to $A$.
    (iv) Update $\widehat{\mathcal{P}}$ as follows: For every path $P' \in \widehat{\mathcal{P}}'$, if the first or last edge of $P'$ belongs to $A'$, replace it
        with one of its corresponding edges in $A$, and then pick a distinct path $P \in \widehat{\mathcal{P}}$ and append $P'$ at the
        end of $P$, using a link edge to connect the endpoints of $P$ and $P'$ if necessary.
  (c) Otherwise:
    (i) Add to $A$, with multiplicity $k + 1$, all edges $\{u, w\}$ with $u \in \mathrm{right}(W_{a-1})$, taking $u = s$ if $a = 0$, and
        $w \in \mathrm{left}(W_{b+1})$, taking $w = t$ if $b = q + 1$.
    (ii) Prolong every path $P \in \widehat{\mathcal{P}}$ with a link edge (if $a > 0$) and an edge of $A$, so that $P$ ends in $\mathrm{left}(W_{b+1})$,
        or in $t$ if $b = q + 1$.
```

Fig. 4. The outer loop algorithm

It is straightforward to verify the invariant stated in the loop: at every step, $\widehat{\mathcal{P}}$ is a family of λ^* edge-disjoint paths in $G + A$, starting in s and ending in $\mathrm{left}(W_a)$. It is also straightforward to verify the feasibility of the updates of $\widehat{\mathcal{P}}$. Furthermore, observe that after the last iteration of the loop, all paths of $\widehat{\mathcal{P}}$ end in t. Thus, at the end of the algorithm $\widehat{\mathcal{P}}$ is indeed a family of λ^* edge-disjoint paths from s to t in $G + A$.

We now prove that, in a well-defined sense, most edges in the returned set A are compatible with most minimal (s,t)-cuts Z.

Lemma 3.17. *Let $W_{a,b}$ be a stretch processed by* Sample *such that every bundle of the stretch is unaffected by Z. Then every edge added to A while processing $W_{a,b}$ is compatible with Z.*

Proof. Note that every vertex of $N[W_{a,b}]$ is in the same connected component of $G - Z$. Hence it suffices to observe that every edge added to A in this phase has both endpoints in $N[W_{a,b}]$. □

Now we are set to prove correctness of the outer loop algorithm assuming a correct realization of the inner-loop interface.

Lemma 3.18. *Assume that an algorithm* Short-separation *correctly realizes the above interface such that for every valid single-bundle (multiple-bundle) instance $(G', s', t', k', \lambda')$ with $k' \leq k$, the returned pair $(A', \widehat{\mathcal{P}}')$ is compatible with a fixed valid cut Z' with probability $32e^{-g(\lambda_{G'}(s',t'),k')}$ ($32(k')^3e^{-g(\lambda_{G'}(s',t'),k')}$). Then for any (G, s, t, k, λ^*),* Sample *returns an (s, t)-flow-augmenting set A such that $\lambda_{G+A}(s, t) \geq \lambda^*$ and for any eligible (s, t)-cut Z in G of size k and with $|Z_{s,t}| = \lambda^*$, the returned pair $(A, \widehat{\mathcal{P}})$ is compatible with Z with probability at least $e^{-g(\lambda_G(s,t),k)}$.*

Proof. The lemma holds essentially vacuously if Sample(G, s, t, k, λ^*) stops at step 1. Hence we assume $\lambda \leq \lambda^* \leq k$. Since G is connected, $\lambda \geq 1$, hence $k \geq 1$.

We first prove that all calls to Short-separation or Short-separation-single are made for valid instances $(G', s', t', k, \lambda^*)$. Let $(G', s', t', k, \lambda^*)$ be an instance on which Short-separation or Short-separation-single is called and let $W_{a,b}$ be the stretch that the call corresponds to. It can be verified that G', relative to minimum (s', t')-cuts, decomposes into bundles $\{s'\}, W_a, \ldots, W_b, \{t'\}$. A key point here is that s' and t' are both incident with precisely λ edges in G', and $\lambda_{G'}(s', t') = \lambda$. This makes $\delta(s')$ the unique closest minimum (s', t')-cut. From this point on, the sequence of closest minimum (s', t')-cuts that define blocks and bundles is identical to ones between the blocks that form bundles $W_a, \ldots, W_b$ in G. Clearly, $G'[W_a \cup \ldots W_b] \cong G[W_a \cup \ldots \cup W_b]$ (canonically) so we arrive at the same decomposition into bundles. At the end, $\delta(t')$ can be seen to be final closest minimum (s', t')-cut that arises when computing blocks and bundles for (G', s', t'), using a symmetric argument to the one for $\delta(s')$.

Now, we show the compatibility property. Let Z be any λ^*-eligible (s, t)-cut of size k. By Lemma 3.15, there is a unique strongly affected stretch $W_{a,b}$, and by Lemma 3.14 at most $2|Z|$ bundles are affected in total. Let $\ell = b - a + 1$ be the number of bundles in $W_{a,b}$ and let $Z' = Z \cap W_{a,b}$. We have $\ell \leq 2|Z'|$ and $\lambda^* \leq |Z'| \leq k$.

We are interested in the following success of the random choices made by the algorithm: the algorithm goes into mode `single` if $a = b$ and into mode `multiple` otherwise, $k' = |Z'|$, and the coloring of bundles in the loop is such that every bundle of $W_{a,b}$ is red, while W_{a-1}, W_{b+1}, and every other affected bundle is blue. Since there are at most $2(k - |Z \cap W_{a,b}|)$ affected bundles that are not in $W_{a,b}$, the above success happens with probability at least

- if $a = b$ and $k = |Z'|$: 2^{-5};
- if $a = b$ and $k > |Z'|$:
$$2^{-5}(k - \lambda^*)^{-1}2^{-2(k-|Z'|)} \geq 2^{-5}k^{-1}2^{-2(k-|Z'|)};$$
- if $a < b$:
$$\begin{aligned}&(k - \lambda^* + 1)^{-1} \cdot k^{-2-2(k-|Z'|)} \cdot (1 - 1/k)^{\ell}\\ &\geq k^{-3-2(k-|Z'|)} \cdot (1 - 1/k)^{2k} \geq 2^{-4}k^{-3-2(k-|Z'|)}.\end{aligned}$$

Henceforth we assume that the above success indeed happens.

If this is the case, then for every two consecutive bundles W_i and W_{i+1} of different colors, either W_i or W_{i+1} is unaffected. In particular, all endpoints of the edges of $E(W_i, W_{i+1})$ are in the same connected component of $G - Z$. Thus, all link edges added to A are compatible with Z.

Let us now consider the processing of some maximal monochromatic stretch $W_{c,d}$ other than $W_{a,b}$. If $W_{c,d}$ is red, then by assumption on the coloring it is a stretch of unaffected bundles, and any edges added are compatible with Z by Lemma 3.17. Furthermore, any flow $\widehat{\mathcal{P}}'$ does not intersect Z, so the edges appended in the paths of $\widehat{\mathcal{P}}$ are disjoint with Z.

Otherwise, if $W_{c,d}$ is blue, then we claim that $\mathrm{left}(W_c) \cup \mathrm{right}(W_d)$ are contained in the same connected component in $G - Z$. Indeed, by assumption on the coloring, any affected bundle in $W_{c,d}$ is contained in some weakly affected stretch $W_{c',d'}$ where the stretch is contained in $W_{c,d}$ in its entirety. By Prop. 3.16 the endpoints of such a stretch are contained in the same component of $G - Z$, as are the endpoints of any stretch of unaffected bundles. The claim follows. Thus the edges added by Sample for $W_{c,d}$ are compatible with Z. Furthermore, in this case all edges appended to the paths of $\widehat{\mathcal{P}}$ are from A.

Now consider the strongly affected stretch $W_{a,b}$. Observe that Sample will make a recursive call to Short-separation or Short-separation-single for this stretch; let the resulting instance be $(G', s', t', k', \lambda^*)$. Note that Z' are the edges of Z contained in G' and that Z' is a valid cut for $(G', s', t', k', \lambda^*)$. Furthermore, $\lambda = \lambda_G(s, t) = \lambda_{G'}(s', t')$. Indeed, by Lemma 3.15 $\mathrm{left}(W_a) \subseteq R_s(Z)$ and $\mathrm{right}(W_b) \subseteq R_t(Z)$, and since W_{a-1} (if any) and W_{b+1} (if any) are unaffected, these are entirely contained in $R_s(Z)$ respectively $R_t(Z)$ as well. Hence Z' is an eligible (s', t')-cut in G'. Finally, $|Z'| = k'$ and $|Z'_{s',t'}| = |Z_{s,t}| = \lambda^*$, and by assumption Z' affects every bundle W_i, $1 \le i \le q$, of G'.

Thus, since Short-separation and Short-separation-single implement the inner-loop interface, with probability at least $32(k')^3 e^{-g(\lambda,k')}$ in case of Short-separation and $32e^{-g(\lambda,k')}$ in case of Short-separation-single, it returns a pair $(A', \widehat{\mathcal{P}}')$ that is compatible with Z' in G'.

We verify that the edges added to A for A' are compatible with Z. The connected components of $G[W_{a-1,b+1}] - Z$ are the same as those of $G' - Z'$ except that the component of s' has W_{a-1} in place of s', and the component of t' contains W_{b+1} instead of t' (respectively, are identical but are missing s' and t' if $a = 0$ and/or $b = q + 1$). Thus, the only edges in A that could, in principle, be incompatible with Z are those that were added in place of edges in A' that are incident with s' or t'. But in all cases, the endpoint replacing s' respectively t' is contained in $R_s(Z)$ respectively $R_t(Z)$, implying that they are compatible with Z in G if they are compatible with Z' in G'.

For the family of paths $\widehat{\mathcal{P}}$, note that if $(A', \widehat{\mathcal{P}}')$ is compatible with Z', then for every $P' \in \widehat{\mathcal{P}}'$, the path P' intersects Z' in precisely one edge and that edge belongs to $Z'_{s,t} = Z_{s,t}$. Hence, by appending P' to a path $P \in \widehat{\mathcal{P}}$ we add one intersection of P with Z and that intersection belongs to $Z_{s,t}$. Since there is only one strongly affected stretch and in all other cases the edges appended to the paths of $\widehat{\mathcal{P}}$ are disjoint with Z, $\widehat{\mathcal{P}}$ is a witnessing flow for Z in $G + A$ as desired.

Furthermore, the existence of $\widehat{\mathcal{P}}$ implies that $\lambda_{G+A}(s, t) \ge |\widehat{\mathcal{P}}| = \lambda^*$.

In summary, Sample produces a pair $(A, \widehat{\mathcal{P}})$ that is compatible with Z with probability at least (assuming $c_1 \ge 5$):

- if $a = b$ and $k = |Z'|$:

$$2^{-5} \cdot 32 \cdot e^{-g(\lambda,k)} = e^{-g(\lambda,k)};$$

- if $a = b$ and $k > |Z'|$:

$$2^{-5} k^{-1} 2^{-2(k-k')} e^{-g(\lambda,k')} \ge e^{-5-\ln k - 2(k-k')} e^{c_2(k-k')(1+\ln k)} e^{-g(\lambda,k')} \ge e^{-g(\lambda,k')};$$

- if $a < b$:

$$\begin{aligned}&\frac{1}{16}k^{-3-2(k-k')} \cdot 16(k')^3 e^{-g(\lambda,k')}\\ &\quad\geq e^{-g(\lambda,k)} \cdot k^{c_1(k-k')} \cdot (k')^3 \cdot k^{-3-2(k-k')}\\ &\quad\geq e^{-g(\lambda,k)} \cdot k^{(c_1-2)(k-k')} \cdot (k'/k)^3\\ &\quad\geq e^{-g(\lambda,k)}.\end{aligned}$$

This finishes the proof of the lemma. □

3.3 Cut splits and the inner loop

```
Algorithm SHORT-SEPARATION-SINGLE$(G, s, t, k, \lambda^*)$
(1) If $(G, s, t, k, \lambda^*)$ is not a valid input, or it does not hold that $\lambda_G(s,t) \leq \lambda^* \leq k$, then set $A$ to be
    $\max(k+1, \lambda^*)$ copies of $\{s,t\}$, $\widehat{\mathcal{P}}$ to be any $\lambda^*$ of these copies, and return $(A, \widehat{\mathcal{P}})$.
(2) Let $V = W_0 \cup W_1 \cup W_2$ be the partition of $G$ into bundles.
(3) If $W_1$ is a connected bundle:
    (a) Let $A_0 = \delta(s) \cup \delta(t)$.
    (b) Compute $(A, \widehat{\mathcal{P}}) \longleftarrow$SAMPLE$(G + A_0, s, t, k, \lambda^*)$.
    (c) Return $(A_0 \cup A, \widehat{\mathcal{P}})$.
(4) Otherwise:
    (a) Let $W_1 = W_1^{(1)} \cup \ldots \cup W_1^{(c)}$ be the partition of $G[W_1]$ into connected components, and for each
        $i \in [c]$ let $\lambda_i$ be the amount of $(s,t)$-flow routed through $W_1^{(c)}$; i.e., $\lambda = \lambda_1 + \ldots + \lambda_c$ where $\lambda_i > 0$ for
        each $i \in [c]$
    (b) Randomly sample partitions $\lambda^* = \lambda_1^* + \ldots + \lambda_c^*$ and $k = k_1 + \ldots k_c$ such that $\lambda_i \leq \lambda_i^* \leq k_i$ for each
        $i \in [c]$.
    (c) For every $i \in [c]$, let $G^{(i)} = G[W_1^{(i)} \cup \{s,t\}]$ and compute $(A_i, \widehat{\mathcal{P}}_i) \longleftarrow$SAMPLE$(G^{(i)}, s, t, k_i, \lambda_i^*)$.
    (d) Return $(A := \bigcup_{i=1}^c A_i, \widehat{\mathcal{P}} := \bigcup_{i=1}^c \widehat{\mathcal{P}}_i)$.
```

Fig. 5. Inner loop: Algorithm for a single bundle

3.3.1 Single-bundle case. We will now describe an algorithm SHORT-SEPARATION-SINGLE that realizes the first half of the inner-loop interface from the previous section. Given a valid single-bundle instance (G, s, t, k, λ^*) where G decomposes into bundles $W_0 \cup W_1 \cup W_2$, $W_0 = \{s\}$ and $W_2 = \{t\}$, it will run in (probabilistic) polynomial time and always return a λ^*-flow augmenting set A. Moreover, for each (s,t)-cut Z that is valid for (G, s, t, k, λ^*), the set A is compatible with Z with probability at least $32e^{-g(\lambda_G(s,t),k)}$. We call $W_0 = \{s\}$ and $W_2 = \{t\}$ trivial bundles, W_1 is the non-trivial bundle. The algorithm is given in Figure 5.

Let us start with an intuition. There is only one bundle to care about, namely W_1. If $G[W_1]$ is connected, we observe that after duplicating all edges incident with s or t (which are assumed to be disjoint with the sought cut Z) the flow from s to t increases. This allows us to recompute the flow and recurse. Otherwise, if $G[W_1]$ is disconnected, we want to independently consider each connected component in a recursive call. To this end, we need to guess how many edges Z and $Z_{s,t}$ contains in each component of $G[W_1]$. The probability of successful guess is charged to the fact that the budget k gets nontrivially split between the subinstances in the recursive calls.

Let us now proceed with the formal arguments. A few remarks are in place. First, if the algorithm exists at Step 1, then no valid cut Z exists and we can deterministically output a trivially correct answer. Second, sampling of values $(\lambda_1^*, \ldots, \lambda_c^*, k_1, \ldots, k_c)$ does not need to be uniform, but we require that each valid output $(\lambda_1^*, \ldots, \lambda_c^*, k_1, \ldots, k_c)$ is sampled with probability at least k^{-2c}. Note that there are at most k^{2c} valid outputs. This can be achieved by, e.g., sampling each λ_i^* and k_i uniformly at random from $\{1, 2, \ldots, k\}$ and, if the sampled values do not satisfy the requirements, return one fixed partition instead.

Let us now analyse the case when W_1 is connected.

Lemma 3.19. *Let (G, s, t, k, λ^*) and W_1 be as above, and let Z' be a valid cut for (G, s, t, k, λ^*). If W_1 is a connected bundle, then $\delta(s) \cup \delta(t)$ is a $(\lambda_G(s, t) + 1)$-flow-augmenting set compatible with Z'.*

Proof. Let $A = \delta(s) \cup \delta(t)$. Since Z' is a valid cut, $Z \cap A = \emptyset$ and A is compatible with Z'. Furthermore, if W_1 is a connected bundle, then it consists of a single block. Assume for a contradiction that $G+A$ has an (s, t)-cut C of size $\lambda_G(s, t)$. Then $C \cap A = \emptyset$, and C is an (s, t)-min cut in G disjoint from $\delta(s) \cup \delta(t)$. This contradicts the assumption that W_1 a block. Thus every (s, t)-min cut in G intersects $\delta(s) \cup \delta(t)$ in at least one edge e. Since A contains a copy of e, C is no longer an (s, t)-cut in $G + A$. Hence $G + A$ has no (s, t)-cuts of size $\lambda_G(s, t)$, and $\lambda_{G+A}(s, t) > \lambda_G(s, t)$. □

Lemma 3.20. *Assume $c_2 > \ln 32$, i.e., $e^{c_2} > 32$. Furthermore, assume that Sample is correct for all inputs $(G', s', t', k', \lambda')$ where either $k' < k$ or $k' = k$ but $\lambda_{G'}(s', t') > \lambda_G(s, t)$, with a success probability of at least $e^{-g(\lambda_{G'}(s',t'),k')}$ for any eligible (s, t)-cut Z. Then Short-separation-single(G, s, t, k, λ^*) is correct, with a success probability of at least $32e^{-g(\lambda_G(s,t),k)}$.*

Proof. Assume that (G, s, t, k, λ^*) is a valid input. As discussed, we can assume $\lambda_G(s, t) \leq \lambda^* \leq k$. Let Z be a valid cut. If W_1 is a connected bundle, then $A = \delta(s) \cup \delta(t)$ is flow-augmenting and compatible with Z by Lemma 3.19. For the success probability bound, the statement is trivial if $\lambda_{G+A}(s, t) > \lambda^*$ (there is no such Z in this case). Otherwise note that $\lambda_{G+A}(s, t) > \lambda_G(s, t)$ so

$$g(\lambda_G(s, t), k) > g(\lambda_{G+A}(s, t), k) + c_2.$$

Hence, the probability bound follows as we assumed $e^{c_2} > 32$.

If W_1 is a disconnected bundle, let $W_1 = W_1^{(1)} \cup \ldots \cup W_1^{(c)}$ be as in the algorithm. For $i \in [c]$, let $\lambda_i^* = |Z_{s,t} \cap E(W_1^{(i)})|$ and $k_i = |Z \cap E(W_1^{(i)})|$; then by assumption $\lambda^* = \lambda_1^* + \ldots + \lambda_c^*$, $k = k_1 + \ldots k_c$, and $\lambda_i \leq \lambda_i^* \leq k_i$. We note that the algorithm guesses the correct values of k_i and λ_i^* with probability at least k^{-2c}.

Consider some $i \in [c]$ and let $G^{(i)} = G[W_1^{(i)} \cup \{s, t\}]$. Let $Z^{(i)} = Z \cap E(G^{(i)})$, and note that $Z^{(i)}$ is an (s, t)-cut in $G^{(i)}$, with endpoints in different connected components of $G^{(i)} - Z^{(i)}$, and with $Z^{(i)} \cap (\delta(s) \cup \delta(t)) = \emptyset$. Thus $Z^{(i)}$ is eligible for $G^{(i)}$. Furthermore by assumption $|Z_{s,t}^{(i)}| = \lambda_i^*$ and $|Z^{(i)}| = k_i < k$. Thus each call to Sample $(G', s, t, k_i, \lambda_i^*)$ will by assumption return a set A_i such that $\lambda_{G+A_i}(s, t) \geq \lambda_i^*$; since $E(G)$ are partitioned across the instances $G^{(i)}$, it follows that $A = A_1 \cup \ldots \cup A_c$ is a flow-augmenting set with $\lambda_{G+A}(s, t) \geq \lambda^*$. Furthermore, for every $i \in [c]$, with probability at least $e^{-g(\lambda_i,k_i)}$ the set A_i is compatible with $Z^{(i)}$. Now $(A, \widehat{\mathcal{P}})$ is compatible with Z if every pair $(A_i, \widehat{\mathcal{P}}_i)$ is compatible with the respective set $Z^{(i)}$. Hence, the success probability is lower bounded by:

$$\begin{aligned}
k^{-2c} \cdot \prod_{i=1}^{c} e^{-g(\lambda_i,k_i)} &= \exp\left(-2c\ln k - \sum_{i=1}^{c} (c_1(2k_i - \lambda_i) - c_2)\,(1+\ln k_i)\right) \\
&\geq \exp\left(-2c\ln k - (1+\ln k)\sum_{i=1}^{c} c_1(2k_i - \lambda_i) - c_2\right) \\
&= \exp\left(-2c\ln k + (1+\ln k)\,(c_1(2k-\lambda) - c_2) + (1+\ln k)(c-1)c_2\right) \\
&\geq 32e^{-g(\lambda,k)} \cdot \exp\left(((c-1)c_2 - 2c)\ln k + ((c-1)c_2 - \ln 32)\right) \\
&\geq 32e^{-g(\lambda,k)}.
\end{aligned}$$

In the above we have used that $c_1 > c_2$ and, in the last inequality, that $c \geq 2$, $c_2 \geq 4 > \ln 32$. This finishes the proof of the lemma. □

3.3.2 Multiple-bundle case. We will now describe an algorithm Short-separation that realizes the second inner-loop interface from the previous section. Given a valid multiple-bundle instance (G, s, t, k, λ^*) where G decomposes into bundles $W_0 \cup \ldots \cup W_{q+1}$, with $2 \leq q \leq 2k$, and $W_0 = \{s\}$ and $W_{q+1} = \{t\}$, and with $\lambda := \lambda_G(s,t) < \lambda^*$ it will run in (probabilistic) polynomial time and always return an (s,t)-flow augmenting set A. Moreover, for each (s,t)-cut Z that is valid for (G, s, t, k, λ^*), the set A is compatible with Z with probability at least $32k^3e^{-g(\lambda_G(s,t),k)}$. We call $W_0 = \{s\}$ and $W_{q+1} = \{t\}$ trivial bundles; all others are called non-trivial bundles.

We start with some intuition. We focus on the cut C between W_1 and W_2 and want to guess, for each edge $e \in C$ and each endpoint $v \in e$, the *type* of v: whether v lies in the connected component of $G - Z$ that contains s (type s), that contains t (type t), or some other component of $G - Z$ (type $\perp$). The type of an edge $e \in C$ is the pair of the types of its endpoints. Having guessed the types, we are able to construct instances for the recursive call for W_1 and $W_{2,q}$ separately. The assumption that all bundles are affected ensures not all edges of C are of type (s,s) and not all edges of C are of type (t,t), which implies that both instances in the recursion are nontrivial and of strictly smaller budget k. However, to obtain the promised success probability, we need to be careful with the distribution from which we sample types of endpoints of edges of C.

We now proceed to the formal arguments.

The algorithm is shown in Figure 6, but to discuss it we need a few results. Recall that in the inner loop we will be interested only in valid cuts, and a valid cut affects every non-trivial bundle $W_1, \ldots, W_q$ of G.

Let C be the min-cut between W_1 and W_2. We define a *cut labelling* $\varphi_Z \colon V(C) \to \{s, t, \perp\}$ of C by Z as

$$\varphi_Z(v) = \begin{cases} s & v \in R_s(Z) \\ t & v \in R_t(Z) \\ \perp & \text{otherwise.} \end{cases}$$

For every edge $uv \in C$ with $u \in V(W_1)$ and $v \in V(W_2)$, the *type* of the edge uv is the pair $\varphi_Z(uv) := (\varphi_Z(u), \varphi_Z(v))$. Let $\Gamma = \{s,t,\perp\} \times \{s,t,\perp\}$ be the set of types. For a type $\gamma \in \Gamma$, let λ_γ be the number of edges $e \in C$ with $\varphi_Z(e) = \gamma$. Within the set of types Γ we distinguish four sets: $\Gamma_= = \{(\alpha,\beta) \in \Gamma \mid \alpha = \beta\} = \{(s,s),(t,t),(\perp,\perp)\}$, $\Gamma_{\neq} = \{(\alpha,\beta) \in \Gamma \mid \alpha \neq \beta\} = \Gamma \setminus \Gamma_=$, $\Gamma_{\leftarrow} = \{t,\perp\} \times \{s,t,\perp\}$, and $\Gamma_{\rightarrow} = \{s,t,\perp\} \times \{s,\perp\}$. We denote $\lambda_= = \sum_{\gamma \in \Gamma_=} \lambda_\gamma$ and similarly we define $\lambda_{\neq}$, $\lambda_{\leftarrow}$, and $\lambda_{\rightarrow}$.

Let $Z_1 = Z \cap E(W_1)$, $Z_2 = Z \cap E(W_{2,q})$ and $Z_C = Z \cap C$. Note that $Z = Z_1 \cup Z_2 \cup Z_C$ is a partition of Z. We make some simple observations.

```
Algorithm Short-separation($G, s, t, k, \lambda^*$)
(1) If $(G, s, t, k, \lambda^*)$ is not a valid multiple-bundle input, then return $k+1$ copies of the edge $\{s, t\}$ and stop.
(2) Let $V = W_0 \cup \ldots \cup W_{q+1}$ be the partition of $G$ into bundles. Let $C$ be the min-cut between $W_1$ and $W_2$.
(3) Randomly sample values $0 \leq \lambda_\gamma \leq \lambda$ for $\gamma \in \Gamma$ such that $\sum_{\gamma \in \Gamma} \lambda_\gamma = \lambda = |C|$. Denote $\lambda_0 = \sum_{\gamma \in \Gamma_0} \lambda_\gamma$.
(4) For every edge $uv \in C$ with $u \in V(W_1)$ and $v \in V(W_2)$, guess a label $\varphi(uv) \in \Gamma$ with the probability of $\varphi(uv) = \gamma$ being $\lambda_\gamma/\lambda$.
    (a) Define $\varphi(u)$ and $\varphi(v)$ such that $(\varphi(u), \varphi(v)) = \varphi(uv)$. If a vertex $x$ obtains two distinct values $\varphi(x)$ in this process, return $A$ being $k+1$ edges $st$ and stop.
    (b) Let $\lambda_C^*$ be the number of edges $e \in C$ such that $\varphi(e) \in \{(s,t), (t,s)\}$.
(5) Let $A_{st}$ contain $k+1$ copies of each edge $\{u, v\}$ with $u, v \in \{s\} \cup \varphi^{-1}(s)$ or with $u, v \in \{t\} \cup \varphi^{-1}(t)$
(6) Compute a set $\widehat{\mathcal{P}}_C$ of size $\lambda_C^*$ as follows: for every $e \in C$ such that $\varphi(e) \in \{(s,t), (t,s)\}$, let $e = uv$ be such that $\varphi(u) = s$ and $\varphi(v) = t$, and add to $\widehat{\mathcal{P}}_C$ a three-edge path $P_e$ consisting of the edges $su \in A_{st}$, $e$, and $tv \in A_{st}$.
(7) Randomly sample a partition $\lambda^* = \lambda_1^* + \lambda_C^* + \lambda_2^*$ subject to the following constraints:
    (a) $\lambda_1^* \geq \lambda_{(t,s)} + \lambda_{(t,t)} + \lambda_{(t,\perp)}$ and $\lambda_1^* = 0$ if $\lambda_{(t,s)} = \lambda_{(t,t)} = \lambda_{(t,\perp)} = 0$.
    (b) $\lambda_2^* \geq \lambda_{(s,s)} + \lambda_{(t,s)} + \lambda_{(\perp,s)}$ and $\lambda_2^* = 0$ if $\lambda_{(s,s)} = \lambda_{(t,s)} = \lambda_{(\perp,s)} = 0$.
(8) Randomly sample a partition $k = k_1 + k_C + k_2$ subject to the following constraints:
    (a) $\lambda_1^* \leq k_1$, $\lambda_0 + \lambda_{(t,t)} \leq k_1 + k_C$, $1 \leq k_1$, $\lambda_\leftarrow \leq k_1$;
    (b) $\lambda_2^* \leq k_2$, $\lambda_0 + \lambda_{(s,s)} \leq k_2 + k_C$, $1 \leq k_2$, $\lambda_\rightarrow \leq k_2$;
    (c) $\sum_{\gamma \in \Gamma_0 \setminus \{(\perp,\perp)\}} \lambda_\gamma \leq k_C \leq \sum_{\gamma \in \Gamma_0} \lambda_\gamma$.
(9) Construct a flow-augmenting set $A_1$ and a flow in $W_1$:
    (a) Let $G_1 = (G + A_{st})[W_1 \cup \{s, t\}]$;
    (b) Compute $(A_1, \widehat{\mathcal{P}}_1) \leftarrow$ Sample($G_1, s, t, k_1, \lambda_1^*$).
(10) Construct a flow-augmenting set $A_2$ in $W_{2,q}$:
    (a) Let $G_2 = (G + A_{st})[W_{2,q} \cup \{s, t\}]$.
    (b) Compute $(A_2, \widehat{\mathcal{P}}_2) \leftarrow$ Sample($G_2, s, t, k_2, \lambda_2^*$).
(11) Return $(A = A_{st} \cup A_1 \cup A_2, \widehat{\mathcal{P}} = \widehat{\mathcal{P}}_C \cup \widehat{\mathcal{P}}_1 \cup \widehat{\mathcal{P}}_2)$.
```

Fig. 6. The inner loop algorithm for multiple-bundle case.

Proposition 3.21. *Assume that Z is a valid (s,t)-cut in G. Then, the following hold.*

(1) *$Z_{s,t} \cap C = \{e \in C \mid \phi_Z(e) \in \{(s,t), (t,s)\}\}$;*
(2) *If $Z_{s,t} \cap E(W_1) \neq \emptyset$, then there exists $u \in V(W_1) \cap V(C)$ with $\varphi_Z(u) = t$;*
(3) *If $Z_{s,t} \cap E(W_{2,q}) \neq \emptyset$, then there exists $v \in V(W_2) \cap V(C)$ with $\varphi_Z(v) = s$;*
(4) *For every $uv \in C$ such that $\varphi_Z(u) \neq \varphi_Z(v)$, we have $uv \in Z_C$. Conversely, if $uv \in Z_C$, then $\varphi_Z(u) \neq \varphi_Z(v)$ or $\varphi_Z(u) = \varphi_Z(v) = \perp$.*
(5) *$Z_1 \cup Z_C \neq \emptyset$ and $Z_2 \cup Z_C \neq \emptyset$.*
(6) *$|Z_1 \cup Z_C| \geq \lambda_{\neq} + \lambda_{(\perp,\perp)} + \lambda_{(t,t)}$ and $|Z_2 \cup Z_C| \geq \lambda_{\neq} + \lambda_{(\perp,\perp)} + \lambda_{(s,s)}$.*
(7) *$|Z_1| \geq \lambda_\leftarrow$ and $|Z_2| \geq \lambda_\rightarrow$.*

Proof. *1.* Holds by definition of $Z_{s,t}$ and φ_Z.

2–3. These proofs are symmetric, so we show only the first. Let $\{u, v\} \in Z_{s,t} \cap E(W_1)$ where $v \in R_t(Z)$. Then there is a path P from v to t in $G - Z$. This path must pass through C through a vertex w with $w \in R_t(Z)$.

4. This is straightforward from the assumption that every edge of Z connects two distinct connected components of $G - Z$.

5. If this does not hold, then some non-trivial bundle of G is unaffected.

6–7. Consider a maximum (s,t)-flow $(P_e)_{e \in C}$ where $e \in E(P_e)$ for $e = u_1u_2 \in C$, $u_1 \in W_1$, $u_2 \in W_2$. The path P_e first goes from s via W_1 to e and then continues via $W_{2,q}$ to t. If there no edge of

$E(P_e) \cap Z$ between s and u_1, then $\varphi_Z(u_1) = s$. IF additionally $e \notin Z$, then $\varphi_Z(e) = (s, s)$. This proves the two inequalities of 6–7. concering Z_1. The argument for Z_2 is symmetric. □

We use this to show the correctness of the algorithm.

Lemma 3.22. *Assume that Sample is correct for all inputs $(G', s', t', k', \lambda')$ where either $k' < k$ or $k' = k$ but $(k' - \lambda_{G'}(s', t')) < (k - \lambda_G(s, t))$, with a success probability of at least $e^{-g(\lambda_{G'}(s',t'),k')}$. Then Short-separation$(G, s, t, k, \lambda^*)$ is correct, with a success probability of at least $32k^3 e^{-g(\lambda_G(s,t),k)}$.*

Proof. First observe that if a call $(G_i, s, t, k_i, \lambda_i^*)$ is made to Sample, then s and t are connected in G_i. Indeed, $G[W_1 \cup \{s\}]$ is connected, and if $\varphi^{-1}(t) \cap V(W_1) = \emptyset$ then the algorithm always guesses $\lambda_1^* = 0$, hence no recursive call is made. Similarly, $G[W_{2,q} \cup \{t\}]$ is connected and if s is not adjacent to $V(W_2)$ in $G + A_{st}$ then no recursive call into G_2 is made. Hence each recursive call is only made to a connected graph G_i and we can assume that $\widehat{\mathcal{P}}_i$ is a flow of size λ_i^* in $G_i + A_i$. We show that $\widehat{\mathcal{P}}$ is a flow of size λ^* in $G + A$, which implies that $\lambda_{G+A}(s, t) \geq \lambda^*$. Indeed, the paths of $\widehat{\mathcal{P}}_1 \cup \widehat{\mathcal{P}}_2$ exist in $G + A$ and are pairwise edge-disjoint. Furthermore, for every edge $e \in C$ with $\varphi(e) \in \{(s, t), (t, s)\}$, the constructed path $P_e \in \widehat{\mathcal{P}}_C$ is a path from s to t disjoint from $\widehat{\mathcal{P}}_1 \cup \widehat{\mathcal{P}}_2$. Since $|\widehat{\mathcal{P}}_C| = \lambda_C^*$ and $\lambda^* = \lambda_1^* + \lambda_C^* + \lambda_2^*$, $\widehat{\mathcal{P}}$ is as desired.

Next, we consider the probability that $(A, \widehat{\mathcal{P}})$ is compatible with Z, where Z is a fixed valid (s, t)-cut. The algorithm correctly guesses (in every bullet, we condition on the previous guesses being correct):

- values λ_γ for $\gamma \in \Gamma$ with probability at least $(1 + \lambda)^{-|\Gamma|} \geq k^{-9}$;
- $\varphi = \varphi_Z$ with probability

$$\prod_{e \in C} \frac{\lambda_{\varphi_Z(e)}}{\lambda} = \prod_{\gamma \in \Gamma} \left(\frac{\lambda_\gamma}{\lambda}\right)^{\lambda_\gamma} = \exp\left(-\sum_{\gamma \in \Gamma} \lambda_\gamma \ln(\lambda/\lambda_\gamma)\right).$$

- values $\lambda_1^* = |Z_{s,t} \cap E(W_1)|$ and $\lambda_2^* = |Z_{s,t} \cap E(W_{2,q})|$ with probability at least k^{-2};
- values $k_1 = |Z \cap E(W_1)|$, $k_2 = |Z \cap E(W_{2,q})|$, $k_C = |Z \cap C|$ with probability at least k^{-2}, as there are at most k^2 possible values of (k_1, k_2).

Proposition 3.21 ensures that in all of the above guesses, the correct value of is among one of the options with positive probability. Furthermore, $\lambda_C^* = |Z_{s,t} \cap C|$ is computed (deterministically) by the algorithm.

It was argued above that each recursive call on a graph G_i, $i = 1, 2$, is made only if G_i is connected. We claim that furthermore $Z_1 := Z \cap E(W_1)$ is an eligible (s, t)-cut in G_1. Indeed, $Z_1 \cap \delta(s) = \emptyset$ by assumption, and $Z_1 \cap \delta(t) = \emptyset$ since all edges of $\delta(t)$ in G_1 are from A_{st}. Furthermore, by assumption, for every vertex u of $N_{G_1}(s)$ and every vertex v of $N_{G_1}(t)$, we have $u \in R_s(Z)$ and $v \in R_t(Z)$. Hence Z_1 in particular cuts every path from u to v in $G[W_1]$, and by cutting all these paths Z_1 must cut s from t in G_1. Finally, no edge of Z_1 goes within a connected component of $G_1 - Z_1$, since the only paths that are added to $G[W_1]$ go between vertices of the same component (either $R_s(Z)$ or $R_t(Z)$) in $G - Z$. Hence with probability at least $e^{-g(\lambda_{G_1}(s,t),k_1)}$ (or 1 if $\lambda_1^* = 0$) the pair $(A_1, \widehat{\mathcal{P}}_1)$ is compatible with Z_1. All these arguments can also be made symmetrically to argue that with probability at least $e^{-g(\lambda_{G_2}(s,t),k_2)}$ (or 1 if $\lambda_2^* = 0$), $(A_2, \widehat{\mathcal{P}}_2)$ is compatible with Z_2.

By assumption, A_{st} is compatible with Z. Also, if $\varphi = \varphi_Z$, then every path $P \in \widehat{\mathcal{P}}_C$ intersects Z in exactly one edge and this edge belongs to $Z_{s,t}$.

It remains to wrap up the proof of the bound the probability that $(A = A_{st} \cup A_1 \cup A_2, \widehat{\mathcal{P}} = \widehat{\mathcal{P}}_1 \cup \widehat{\mathcal{P}}_2 \cup \widehat{\mathcal{P}}_C)$ is compatible with Z. First, consider a corner case when $\lambda_{(s,s)} = \lambda$, that is, φ_Z is constant at (s, s). Then $k_1 \geq 1$, $k_2 \leq k - 1$, $k_C = 0$, $\lambda_{G_2}(s, t) \geq \lambda_G(s, t)$, and the recursive call on

G_1 is not made. Furthermore, once $\lambda_{(s,s)} = \lambda$ is guessed, φ is defined deterministically. Hence, for sufficiently large constant c_1, $(A, \widehat{\mathcal{P}})$ is compatible with Z with probability at least

$$k^{-13} e^{-g(\lambda_{G_2}(s,t),k_2)} \geq k^{-13} e^{-g(\lambda,k-1)} \geq \exp\left(-16 \ln k - \ln 16 + c_2(1 + \ln 4)\right) 16k^3 e^{-g(\lambda,k)} \geq e^{-g(\lambda,k)}.$$

A symmetric argument holds if $\lambda_{(t,t)} = \lambda$, that is, φ_Z is constant at (t,t).

For the general case, observe that even if the recursive call on G_i is not invoked due to $\lambda_i^* = 0$, then $k_i \geq 1$ and $\lambda_{G_i}(s,t) \leq k_i$ so $e^{-g(\lambda_{G_i}(s,t),k_i)} \leq 1$. Thus, we can use $e^{-g(\lambda_{G_i}(s,t),k_i)}$ as a lower bound on the success probability of the recursive call regardless of whether it was actually invoked.

By the above discussion, the probability that A is compatible with Z is at least

$$k^{-13} \cdot \exp\left(-\sum_{\gamma \in \Gamma} \lambda_\gamma \ln(\lambda/\lambda_\gamma)\right) \cdot e^{-g(\lambda_{G_1}(s,t),k_1)} e^{-g(\lambda_{G_2}(s,t),k_2)}. \tag{2}$$

We start by analysing the second term of the above bound.

We will need the following standard entropy maximization result. Intuitively, it says that once we know the values $(\lambda_\gamma)_{\gamma\in\Gamma}$, guessing each $\phi_Z(e) = \gamma$ with probability λ_γ/λ maximizes the probability of a fully correct guess.

Lemma 3.23. *Let $n \geq 1$ be an integer, $s > 0$, $a_1, a_2, \ldots, a_n \geq 0$ be reals with $\sum_{i=1}^n a_i > 0$, and for reals $x_1, \ldots, x_n \geq 0$ with $\sum_{i=1}^n x_i = s$ define $F(x_1, x_2, \ldots, x_n) = \sum_{i=1}^n a_i \ln x_i$, with the convention that $0 \cdot \ln 0 = 0$. Then, the value of $F(x_1, x_2, \ldots, x_n)$ attains maximum (subject to $\sum_{i=1}^n x_i = s$) when the values x_i are proportional to the values a_i, that is, when $x_i = sa_i/\sum_{j=1}^n a_j$.*

Proof. The case $n = 2$ follows by standard calculus methods, e.g., analysing the derivative of $x \mapsto a_1 \ln x + a_2 \ln(s - x)$. The general case follows from the $n = 2$ case by straightforward induction. ◻

The summands of (2) for $\gamma \in \Gamma_{\neq}$ will not cause much trouble, as if $\phi_Z(e) \in \Gamma_{\neq}$ then $e \in Z$, and we will be able to charge the probability of guessing correctly $\phi_Z(e)$ to the decrease in the budget k in the recursive calls. That is, the following simple estimate will suffice.

$$\lambda_= \ln(\lambda/\lambda_=) + \sum_{\gamma\in\Gamma_{\neq}} \lambda_\gamma \ln(\lambda/\lambda_\gamma) \leq 5k_C \ln k. \tag{3}$$

Indeed, note that (3) is trivial if $\lambda_{\neq} = 0$ or $\lambda \leq 1$, while otherwise we have $k_C \geq 1$ and $k \geq 2$ and we can estimate as follows.

$$\begin{aligned}
&\lambda_= \ln(\lambda/\lambda_=) + \sum_{\gamma\in\Gamma_{\neq}} \lambda_\gamma \ln(\lambda/\lambda_\gamma) && \text{by Lemma 3.23} \\
&\leq -\lambda_= \ln(1 - 1/k) - \sum_{\gamma\in\Gamma_{\neq}} \lambda_\gamma \ln(1/(|\Gamma_{\neq}|k)) && \lambda_= \leq k, |\Gamma_{\neq}| = 6 \\
&\leq -k \ln(1 - 1/k) - \lambda_{\neq} \ln(1/(6k)) && k \geq 2, \lambda_{\neq} \leq k_C \\
&\leq -2 \ln(1 - 1/2) - k_C \ln(1/(6k)) && k_C \geq 1, k \geq 2 \\
&\leq 5k_C \ln k.
\end{aligned}$$

For the terms $\lambda_{(s,s)}$, $\lambda_{(t,t)}$, and $\lambda_{(\bot,\bot)}$ we need to be more careful. Thanks to Proposition 3.21(4.), we have $k_1 \geq \lambda_{(t,t)} + \lambda_{(\bot,\bot)}$ and $k_2 \geq \lambda_{(s,s)} + \lambda_{(\bot,\bot)}$. Recall $k_1 + k_2 = k - k_C$. Intuitively, we would like to charge the probability of guessing $\phi_Z(e)$ correctly for $\phi_Z(e) \in \Gamma_=$ to the fact that the budget $k - k_C$ got split in the recursive calls.

To this end, denote $x_1 = k_1 - \lambda_{(\bot,\bot)}$, $x_2 = k_2 - \lambda_{(\bot,\bot)}$, $x_0 = \lambda_{(\bot,\bot)}$. Note that $x_1 \geq \lambda_{(t,t)}$ and $x_2 \geq \lambda_{(s,s)}$. Furthermore, $k - k_C = k_1 + k_2 = x_1 + x_2 + 2x_0$.

By Lemma 3.23,

$$\begin{aligned}&\lambda_{(s,s)}\ln(\lambda/\lambda_{(s,s)})+\lambda_{(t,t)}\ln(\lambda/\lambda_{(t,t)})+\lambda_{(\perp,\perp)}\ln(\lambda/\lambda_{(\perp,\perp)})\\&\quad=-\left(\lambda_{(s,s)}\ln(\lambda_{(s,s)}/\lambda_=)+\lambda_{(t,t)}\ln(\lambda_{(t,t)}/\lambda_=)+\lambda_{(\perp,\perp)}\ln(\lambda_{(\perp,\perp)}/\lambda_=)\right)+\lambda_=\ln(\lambda/\lambda_=)\\&\quad\le-\left(\lambda_{(s,s)}\ln(x_2/x)+\lambda_{(t,t)}\ln(x_1/x)+\lambda_{(\perp,\perp)}\ln(2x_0/x)\right)+\lambda_=\ln(\lambda/\lambda_=).\end{aligned}$$

As $\lambda_{(s,s)}\le x_2$, $\lambda_{(t,t)}\le x_1$, and $\lambda_{(\perp,\perp)}=x_0$, we have

$$\begin{aligned}&\lambda_{(s,s)}\ln(\lambda/\lambda_{(s,s)})+\lambda_{(t,t)}\ln(\lambda/\lambda_{(t,t)})+\lambda_{(\perp,\perp)}\ln(\lambda/\lambda_{(\perp,\perp)})\\&\quad\le x_1\ln(x/x_1)+x_2\ln(x/x_2)+2x_0\ln(x/(2x_0))+\lambda_=\ln(\lambda/\lambda_=).\end{aligned}\tag{4}$$

We also need the following observation:

Claim 1. *It holds that*

$$k_1-\lambda_{G_1}(s,t)+k_2-\lambda_{G_2}(s,t)\le k-\lambda_G(s,t)+\lambda_{(\perp,\perp)}.$$

Proof. From Proposition 3.21(4.), we infer that

$$|C|-k_C-\lambda_{(\perp,\perp)}\le\lambda_{(s,s)}+\lambda_{(t,t)}.$$

Since in G_1, an endpoint of every edge $e\in C$ with $\varphi_Z(e)=(t,t)$ is connected to t with $k+1$ edges, we have

$$\lambda_{G_1}(s,t)\ge\lambda_{(t,t)}.$$

Symmetrically,

$$\lambda_{G_2}(s,t)\ge\lambda_{(s,s)}.$$

As $k_1+k_2+k_C=k$ and $\lambda_G(s,t)=|C|$, the claim follows. □

To wrap up the analysis, we need the following property of the $z\mapsto z\ln z$ function (for completeness, we provide a proof in Appendix A):

Claim 2. *Let $f(z)=z\ln z$ for $z>0$ and $f(0)=0$. For every constant $C_1>0$ there exists a constant $C_2>0$ such that for every $x_1,x_2,x_0\ge 0$ it holds that*

$$C_2f(x_1+x_2+2x_0)+f(x_1)+f(x_2)+f(2x_0)\ge f(x_1+x_2+2x_0)+C_2f(x_1+x_0)+C_2f(x_2+x_0)+C_1x_0.$$

Claim 2 for $C_1=c_2$ implies an existence of $C_2>0$ (depending on c_2) such that

$$x_1\ln(x/x_1)+x_2\ln(x/x_2)+2x_0\ln(x/(2x_0))+c_2x_0\le C_2\left(x\ln x-k_1\ln k_1-k_2\ln k_2\right).\tag{5}$$

Using the definition of $g(\cdot,\cdot)$, the fact that $\lambda_{(\perp,\perp)}=x_0$ and Claim 1, we obtain that

$$g(\lambda_G(s,t),k)\ge g(\lambda_{G_1}(s,t),k_1)+g(\lambda_{G_2}(s,t),k_2)+c_1\left(k\ln k-k_1\ln k_1-k_2\ln k_2\right)+c_2(1+\ln k)-c_2x_0.\tag{6}$$

Thus, we bound the negated exponent of the probability bound of (2) as follows:

$$\begin{aligned}&13\ln k+\sum_{\gamma\in\Gamma}\lambda_\gamma\ln(\lambda/\lambda_\gamma)+g(\lambda_{G_1}(s,t),k_1)+g(\lambda_{G_2}(s,t),k_2) && \text{by (3) and (4)}\\&\le 13\ln k+x_1\ln(x/x_1)+x_2\ln(x/x_2)+2x_0\ln(x/(2x_0))+5k_C\ln k && \text{by (6)}\\&\quad+g(\lambda_{G_1}(s,t),k_1)+g(\lambda_{G_2}(s,t),k_2)\\&\le 13\ln k+x_1\ln(x/x_1)+x_2\ln(x/x_2)+2x_0\ln(x/(2x_0))+5k_C\ln k && \text{by (5),}\\&\quad+g(\lambda_G(s,t),k)+c_2x_0-c_2(1+\ln k) && c_2\ge 16,\ c_1\ge C_2+5,\\&\quad+c_1(x_1+x_0)\ln(x_1+x_0)+c_1(x_2+x_0)\ln(x_2+x_0)-c_1k\ln k && k=x+k_C\\&\le g(\lambda_G(s,t),k)-3\ln k-\ln 32\end{aligned}$$

This finishes the proof of the lemma. □

3.4 Efficient implementation and final proof

Finally, we show that the algorithms can be implemented to run in time $k^{O(1)}O(m)$, i.e., linear time up to factors of k. We first show how to efficiently decompose G into bundles.

Lemma 3.24. *Let $G = (V, E)$ be an undirected graph, $s, t \in V$, and let $\lambda^* \in \mathbb{Z}$ be given. In time $O(\lambda^* m)$ we can either show that $\lambda_G(s,t) > \lambda^*$ or compute a max (s,t)-flow, blocks, and bundles in G.*

Proof. Since all edge capacities are unit we can compute a packing $\widehat{\mathcal{P}}$ of up to $\lambda^* + 1$ (s,t)-paths in time $O(\lambda^* m)$ using Ford-Fulkerson, and if $|\widehat{\mathcal{P}}| = \lambda^* + 1$ then we are done. Otherwise, assume that $\widehat{\mathcal{P}}$ is a max-flow of value $|\widehat{\mathcal{P}}| = \lambda$, and let G' be the residual flow graph for $\widehat{\mathcal{P}}$ on G. We show how to decompose G into blocks and bundles. Let the *mass* of a vertex set S be $\sum_{v \in S} d(v)$, where $d(v)$ is the degree of the vertex v.

First, observe that the closest min-cut C_0 can be found using a simple reachability query in the residual flow graph. Specifically, the first block V_0 is precisely the set of vertices reachable from s in G'. Hence V_0 can be computed in time linear in its mass and $C_0 = \delta(V_0)$. The sets $V_1, \ldots, V_{p+1}$ can be computed as follows. Let $i \in [p]$ and let $V' = V_0 \cup \ldots \cup V_{i-1}$. Assume that all sets $V_{i'}$, $i' < i$ have been computed, in total time linear in the mass of V'. Hence the cut $C_{i-1} = \delta(V')$ is known as well. Contract V' into a single vertex s' and reorient the arcs of C_{i-1} out from s'. Then V_i is precisely the set of vertices reachable from s', and can be computed in time linear in its mass. Hence we can decompose G into blocks.

To further group the blocks into bundles, we only need to be able to test connectivity. Recall that the first bundle is just $W_0 = V_0$. Assume that we are computing the bundle starting with block V_a. Label the flow-paths $\widehat{\mathcal{P}} = \{P_1, \ldots, P_\lambda\}$, and initialize a partition Q of $[\lambda]$ corresponding to the endpoints of C_{a-1} in V_a (i.e., for every vertex $v \in \mathrm{left}(W_a)$ there is a part $B \in Q$ where $i \in B$ if and only if the edge $E(P_i) \cap C_{a-1}$ is incident with v). In time linear in the mass of V_a, we can compute the connected components of V_a, and the corresponding partition Q' of $\mathrm{right}(W_a)$. Then, as long as the current sequence of blocks is not yet connected (i.e., as long as $Q' \neq \{[\lambda]\}$), repeat the process for every block $a' \geq a$: Let $H = G[V_{a'}]$; for every block $B \in Q$, add a vertex s_B to H, connected to the endpoints of P_i in $V_{a'}$ for every $i \in B$; and compute the connected components of H and the corresponding partition Q'' of $\mathrm{right}(W_{a'})$. Clearly this takes linear time and allows us to detect the first block V_{b+1} such that $V_a \cup \ldots \cup V_b$ is connected. Then the next bundle contains blocks V_a through V_{b-1}. □

We can now prove Theorem 3.7. We refrain from optimizing the exponent of k in the running time, since every plausible application of the theorem will have an overhead of $2^{O(k \log k)}$ separate applications anyway.

Proof of Theorem 3.7. To bound the running time, we make two notes. First, in every call to one of the algorithms Sample, Short-separation or Short-separation-single, recursive calls are only made on disjoint vertex sets of the respective graph G (excepting special vertices s, t). Furthermore, on each call into Sample we either have a decreased value of k or an increased value of $\lambda_G(s,t)$, and there are only $O(k^2)$ possible combined values of (k, λ). Thus every vertex of G except s, t is processed in at most a polynomial number of process calls.

Second, we note that the density of the graph $G + A$ does not increase too much beyond the density of G. Specifically, it is easy to verify that for every vertex $v \in V(G)$, at most $k^{O(1)}$ new edges are added incident with v. Hence it suffices that the local work in each procedure is linear-time in the size of the graph it is called on. For this, the only part that needs care is the computation of bundles, Lemma 3.24. Every other step is immediate. Hence the running time is bounded by some $k^{O(1)}O(m)$.

The rest of the statement — namely, the fact that in the output $(A, \widehat{\mathcal{P}})$ we have $\lambda_{G+A}(s,t) \geq \lambda^*$, $\widehat{\mathcal{P}}$ is an (s,t)-flow of size λ^*, and that with probability $2^{-O(k \log k)}$ the pair $(A, \widehat{\mathcal{P}})$ is compatible with Z, for any eligible (s,t)-cut Z of size at most k — now follows via joined induction from Lemma 3.18, Lemma 3.20 and Lemma 3.22. ☐

ACKNOWLEDGMENTS

This research is a part of a project that have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme Grant Agreement 714704 (M. Pilipczuk). Eun Jung Kim is supported by the grant from French National Research Agency under JCJC program (ASSK: ANR-18-CE40-0025-01).

This project was initiated during two invited research stays of Stefan Kratsch and Magnus Wahlström at Université Paris Dauphine.

REFERENCES

[1] Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. 2018. Multicut Is FPT. *SIAM J. Comput.* 47, 1 (2018), 166–207. https://doi.org/10.1137/140961808

[2] Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michal Pilipczuk. 2016. Designing FPT Algorithms for Cut Problems Using Randomized Contractions. *SIAM J. Comput.* 45, 4 (2016), 1171–1229. https://doi.org/10.1137/15M1032077

[3] Rajesh Chitnis, László Egri, and Dániel Marx. 2017. List H-Coloring a Graph by Removing Few Vertices. *Algorithmica* 78, 1 (2017), 110–146. https://doi.org/10.1007/s00453-016-0139-6

[4] Marek Cygan, Pawel Komosa, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, Saket Saurabh, and Magnus Wahlström. 2021. Randomized Contractions Meet Lean Decompositions. *ACM Trans. Algorithms* 17, 1 (2021), 6:1–6:30. https://doi.org/10.1145/3426738

[5] Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. 2019. Minimum Bisection Is Fixed-Parameter Tractable. *SIAM J. Comput.* 48, 2 (2019), 417–450.

[6] Reinhard Diestel. 2012. *Graph Theory, 4th Edition.* Graduate texts in mathematics, Vol. 173. Springer.

[7] Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. 2021. Flow-augmentation I: Directed graphs. *CoRR* abs/2111.03450 (2021). arXiv:2111.03450 https://arxiv.org/abs/2111.03450

[8] Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. 2021. Solving hard cut problems via flow-augmentation. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, Dániel Marx (Ed.). SIAM, 149–168. https://doi.org/10.1137/1.9781611976465.11

[9] Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. 2022. Directed flow-augmentation. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, Stefano Leonardi and Anupam Gupta (Eds.). ACM, 938–947. https://doi.org/10.1145/3519935.3520018

[10] Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. 2022. Flow-augmentation III: Complexity dichotomy for Boolean CSPs parameterized by the number of unsatisfied constraints. *CoRR* abs/2207.07422 (2022). https://doi.org/10.48550/arXiv.2207.07422 arXiv:2207.07422

[11] Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. 2023. Flow-augmentation III: Complexity dichotomy for Boolean CSPs parameterized by the number of unsatisfied constraints. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, Nikhil Bansal and Viswanath Nagarajan (Eds.). SIAM, 3218–3228. https://doi.org/10.1137/1.9781611977554.CH122

[12] Stefan Kratsch, Shaohua Li, Dániel Marx, Marcin Pilipczuk, and Magnus Wahlström. 2020. Multi-budgeted Directed Cuts. *Algorithmica* 82, 8 (2020), 2135–2155. https://doi.org/10.1007/s00453-019-00609-1

[13] Dániel Marx and Igor Razgon. 2014. Fixed-Parameter Tractability of Multicut Parameterized by the Size of the Cutset. *SIAM J. Comput.* 43, 2 (2014), 355–388. https://doi.org/10.1137/110855247

[14] Christos H. Papadimitriou and Mihalis Yannakakis. 2001. Multiobjective Query Optimization. In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 21-23, 2001, Santa Barbara, California, USA*, Peter Buneman (Ed.). ACM. https://doi.org/10.1145/375551.375560

A PROOF OF CLAIM 2

The following proof of the following lemma, being almost a restatement of Claim 2, is due to Piotr Nayar. We thank Piotr for allowing us to include here the proof. (Note that in Claim 2 we allow x_1, x_2, or x_0 to be equal to 0 with the convention $0 \ln 0 = 0$. Claim 2 follows from the lemma below as $\lim_{z \to 0^+} z \ln z = 0$.)

Lemma A.1. *For every $c_1 \geq 0$ there exist $c_2 \geq 0$ such that for all $x_1, x_2, x_0 > 0$ we have*

$$\begin{aligned} &c_2(x_1 + x_2 + 2x_0) \ln(x_1 + x_2 + 2x_0) + x_1 \ln(x_1) + x_2 \ln(x_2) + 2x_0 \ln(x_0) \\ &\quad \geq (x_1 + x_2 + 2x_0) \ln(x_1 + x_2 + 2x_0) + c_2(x_1 + x_0) \ln(x_1 + x_0) + c_2(x_2 + x_0) \ln(x_2 + x_0) + c_1 x_0. \end{aligned}$$

Proof. The inequality is homogeneous under $(x_1, x_2, x_0) \longrightarrow (\alpha x_1, \alpha x_2, \alpha x_0)$. We can therefore assume that $x_1 + x_2 + 2x_0 = 1$. We then introduce $\lambda \in (0, 1)$ such that $x_1 + x_0 = \lambda$ and $x_2 + x_0 = 1 - \lambda$. The inequality is invariant under $(x_1, x_2) \longrightarrow (x_2, x_1)$, so we can assume that $\lambda \in (0, 1/2]$. Our goal is now to prove that

$$x_1 \ln(x_1) + x_2 \ln(x_2) + 2x_0 \ln(x_0) - c_1 x_0 \geq +c_2 \left[\lambda \ln \lambda + (1 - \lambda) \ln(1 - \lambda)\right].$$

The function $z \mapsto z \ln z$ is decreasing on $(1, 1/e)$ and increasing on $(1/e, 1)$. We consider two cases.

Case 1. $\lambda \in ((e - 1)/(2e), 1/2]$. In this case the left hand side if bounded from below by $-\frac{4}{e \ln 2} - c_1$ and $\lambda \log \lambda + (1 - \lambda) \log(1 - \lambda)$ is bounded from above by some negative constant, so there is nothing to prove.

Case 2. $\lambda \leq \frac{e-1}{2e} < \frac{1}{e}$. In this case if $0 < x \leq \lambda$, then $x \log x \geq \lambda \log \lambda$ and thus $x_1 \log(x_1) + 2x_0 \log(x_0) \geq 3\lambda \log \lambda$. Moreover, $x_2 \geq 1 - 2\lambda$ and therefore $x_2 \log(x_2) \geq (1 - 2\lambda) \log(1 - 2\lambda)$, since $1 - 2\lambda \geq \frac{1}{e}$. After applying theses bounds we have to show that

$$3\lambda \log \lambda + (1 - 2\lambda) \log(1 - 2\lambda) - c_1 \lambda \geq c_2 \left[\lambda \log \lambda + (1 - \lambda) \log(1 - \lambda)\right].$$

In other words, we want to show that there exists c_2 such that

$$\frac{3\lambda \log \lambda + (1 - 2\lambda) \log(1 - 2\lambda) - c_1 \lambda}{\lambda \log \lambda + (1 - \lambda) \log(1 - \lambda)} \leq c_2, \qquad 0 < \lambda \leq \frac{e - 1}{2e}.$$

This can be verified by checking that the limit $\lambda \longrightarrow 0^+$ is finite and thus it is enough to take c_2 to be the supremum of the left hand side. □

Received ???; revised ???; accepted ???