Directed Flow-Augmentation

Eun Jung Kim†
eun-jung.kim@dauphine.fr
Université Paris-Dauphine, PSL Research University,
CNRS, UMR 7243, LAMSADE, 75016, Paris
Paris, France

Marcin Pilipczuk‡
m.pilipczuk@mimuw.edu.pl
University of Warsaw
Warsaw, Poland

Stefan Kratsch
kratsch@informatik.hu-berlin.de
Humboldt-Universität zu Berlin
Berlin, Germany

Magnus Wahlström
Magnus.Wahlstrom@rhul.ac.uk
Royal Holloway, University of London, TW20 0EX
London, UK

ABSTRACT

We show a flow-augmentation algorithm in directed graphs: There exists a randomized polynomial-time algorithm that, given a directed graph $G$, two integers $s, t \in V(G)$, and an integer $k$, adds (randomly) to $G$ a number of arcs such that for every minimal $st$-cut $Z$ in $G$ of size at most $k$, with probability $2^{-\text{poly}(k)}$ the set $Z$ becomes a minimum $st$-cut in the resulting graph.

The directed flow-augmentation tool allows us to prove (randomized) fixed-parameter tractability of a number of problems parameterized by the cardinality of the deletion set, whose parameterized complexity status was repeatedly posed as open problems:

1. **Chain SAT**, defined by Chitnis, Egri, and Marx [ESA’13, Algorithmica’17],
2. a number of weighted variants of classic directed cut problems, such as **Weighted-St-Cut**, **Weighted Directed Feedback Vertex Set**, or **Weighted Almost 2-SAT**.

By proving that Chain SAT is (randomized) FPT, we confirm a conjecture of Chitnis, Egri, and Marx that, for any graph $H$, if the List $H$-COLORING problem is polynomial-time solvable, then the corresponding vertex-deletion problem is fixed-parameter tractable (with the remark that our algorithms are randomized).

CCS CONCEPTS

• Theory of computation → Fixed parameter tractability.

KEYWORDS

directed graphs, fixed-parameter tractability, flow-augmentation, Chain SAT

†Supported by the grant from French National Research Agency under JCJC program (ASSK-ANR-18-CE40-0025-01).
‡This research is a part of a project that has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme Grant Agreement 714794.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

STOC ’22, June 20–24, 2022, Rome, Italy
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9264-8/22/06...
https://doi.org/10.1145/3519935.3520018

1 INTRODUCTION

The study of graph separation problems has been one of the more vivid areas of parameterized complexity in the recent 10–15 years. The term “graph separation problems” captures a number of classical graph problems where, given a (undirected or directed) graph $G$ with a budget $k$ (and possibly some annotations, such as terminal vertices), one aims at obtaining some separation via at most $k$ edge or vertex deletions. For example, the classic $st$-Cut problem asks to delete at most $k$ edges so that there is no $s$-to-$t$ path in the resulting graph and the Feedback Vertex Set asks to remove at most $k$ vertices so that the resulting graph does not contain any cycles (i.e., is a forest in the undirected setting or a DAG in the directed setting). In all these problems, the cardinality of the deletion set is a natural parameter to study.

In 2004, Marx introduced the notion of important separators [15, 16] that turned out to be the key to fixed-parameter tractability of Multiway Cut and Directed Feedback Vertex Set [1], among many other examples. In subsequent years, the study of graph separation problems resulted in a rich toolbox of algorithmic techniques, such as shadow removal [19], treewidth reduction [17], or randomized contractions [2, 7, 8]. At SODA’21, the current authors added one more item to this list: a flow-augmentation technique in undirected graphs [10]. As a result of these developments, most open problems regarding parameterized complexity of graph separation problems in undirected graphs has been resolved. Our work [10] shows that this statement can be formalized in some sense: every problem that can be formalized as a Min UNSAT problem for some Boolean CSP language $\Gamma$ that cannot express implication, is either $\text{W}[1]$-hard or FPT by one of the aforementioned techniques.

For directed graphs, the chartered landscape is much less complete. The notion of important separators and related technique of shadow removal generalizes to directed graphs, leading to fixed-parameter tractability of Directed Feedback Vertex Set [1], Directed Multiway Cut [6], and Directed Subset Feedback Vertex Set [4]. A number of problems whose undirected counterparts are FPT turned out to be $\text{W}[1]$-hard in the directed setting,
including Directed Multicut [6, 20] or Directed Odd Cycle Transversal [14].

However, these results are far from satisfactorily charting the parameterized complexity of directed graph separation problems. Arguably, we seem to lack algorithmic techniques. Most notably, the powerful treewidth reduction theorem [17], stating that (in undirected graphs) all separations of size at most \( k \) between two fixed terminals live in a part of the graph with treewidth bounded by \( O(k) \), seems not to have any meaningful counterpart in directed graphs. As a result, essentially all known FPT algorithms for graph separation problems in directed graphs rely in some part on important separators, which is a greedy argument bounding the number of cuts between two terminals of bounded size that have inclusion-wise maximal set of vertices reachable from one of the terminal. This “greedy” part of this argument breaks down whenever a problem at hand has some weights or annotations.

On the other hand, despite efforts in the last years, we were not able to prove lower bounds for FPT algorithms for many directed graph separation problems. This suggests that maybe there are still more algorithmic techniques to be explored, leading to more positive tractability results.

In this work, we provide such a technique: we show that flow-augmentation, introduced by current authors in [10] for undirected graphs, generalizes to directed graphs.

**Theorem 1.1.** There exists a randomized polynomial-time algorithm that, given a directed graph \( G \), two vertices \( s, t \in V(G) \), and an integer \( k \), outputs a set \( A \subseteq V(G) \times V(G) \) such that the following holds: for every minimal \( st \)-cut \( Z \subseteq E(G) \) of size at most \( k \), with probability \( Z \leq O(k^k \log k) \) remains an \( st \)-cut in \( G + A \) and, furthermore, \( Z \) is a minimum \( st \)-cut in \( G \).

Here, a set \( Z \subseteq E(G) \) is an \( st \)-cut if there is no path from \( s \) to \( t \) in \( G - Z \); it is minimal if no proper subset of \( Z \) is an \( st \)-cut and minimum if it is of minimum possible cardinality. By \( G + A \) we mean the graph \( G \) with all elements of \( A \) added as infinity-capacity arcs.

The proof of Theorem 1.1 is sketched in Section 2. Full proof can be found in the arXiv version [9]. While the proof borrows a general outline and course of action from its undirected counterpart [10], at lower level most details are different, as directed graphs pose different challenges. The main differences are discussed in Section 2.

**Applications.** To illustrate the applicability of the directed flow-augmentation, let us first consider the Weighted \( st \)-Cut problem. Here, we are given a directed graph \( G \) with two terminals \( s, t \in V(G) \), a weight function \( \omega : E(G) \to \mathbb{Z}_+ \), and two integers \( k, W \in \mathbb{Z}_+ \), and we ask for a \( st \)-cut \( Z \) of cardinality at most \( k \) and total weight at most \( W \). This problem is known to be NP-hard and FPT when parameterized by \( k + W \) [11].

By using directed flow-augmentation, we can ensure that the sought solution \( Z \) is actually a minimum \( st \)-cut (i.e., of minimum cardinality). Then, a solution can easily be found in polynomial-time: take \( M := 1 + \sum_{e \in E(G)} \omega(e) \), set the capacity of every edge \( e \) as \( \omega(e) + M \), and find an \( st \)-cut of minimum capacity. By repeating the algorithm \( 2^O(k^k \log k) \) times to ensure constant success probability, we obtain the following.

**Theorem 1.2.** Weighted \( st \)-Cut can be solved in randomized time \( 2^O(k^k \log k) nO(1) \).

The above approach turns out to be more general. An instance of Bundled Cut consists of a directed graph \( G \), vertices \( s, t \in V(G) \), a nonnegative integer \( k \), and a family \( B \) of pairwise disjoint subsets of \( E(G) \). The elements of \( B \) are henceforth called bundles. A cut in a Bundled Cut instance \( I = (G, s, t, k, B) \) is an \( st \)-Cut in \( Z \subseteq \bigcup B \). The Bundled Cut problem asks for an \( st \)-cut that intersects at most \( k \) bundles, that is, \( |B \in B \mid Z \cap B \neq \emptyset | \leq k \). One can define a weighted variant of the Bundled Cut problem where every bundle \( B \in B \) is equipped with a weight \( \omega(B) \in \mathbb{Z}_+ \), we are given also a weight bound \( W \), and we ask for an \( st \)-Cut that intersects at most \( k \) bundles whose total weight is at most \( W \).

If all bundles are singletons, then Bundled Cut just asks for an \( st \)-cut of size at most \( k \) (with the edges of \( E(G) \setminus \bigcup B \) being undeletable), so it is polynomial-time solvable. It is known that, when restricted to bundles of size 2, Bundled Cut is \( W[1] \)-hard when parameterized by \( k \) [18]. To get tractability, we define the following restriction called (Weighted) Bundled Cut with Order where we require the following: for every \( B \in B \) one can order the arcs of \( B \) as \( e_1^B, \ldots, e_{|B|}^B \) such that for every \( 1 \leq i < j \leq |B| \) there exists a path \( P_{i,j}^B \) in \( G \) that goes from one endpoint of \( e_i^B \) to one endpoint of \( e_j^B \) and uses only undeletable arcs (i.e., from \( E(G) \setminus \bigcup B \)) and arcs of \( B \).

Using directed flow-augmentation, we show the following.

**Theorem 1.3.** Weighted Bundled Cut with Order is randomized FPT when parameterized by \( k \) and maximum size of a bundle.

For an integer \( \ell \geq 1 \), the \( \ell \)-Chain SAT problem is the Bundled Cut problem, where every bundle is a path of length at most \( \ell \). At ESA’13, Chitnis, Egri, and Marx [3, 5] defined \( \ell \)-Chain SAT and showed that fixed-parameter tractability of \( \ell \)-Chain SAT (for every fixed \( \ell \geq 1 \)) is equivalent to the following conjecture.

**Conjecture 1.4 (Conjecture 1.1 of [3]).** For every graph \( H \), if List \( H \)-Coloring problem is polynomial-time solvable, then the vertex-deletion variant (delete at most \( k \) vertices from the input graph to obtain a yes-instance to List \( H \)-Coloring) is fixed-parameter tractable when parameterized by \( k \).

Clearly, a \( \ell \)-Chain SAT instance is a Bundled Cut with Order instance with maximum bundle size at most \( \ell \). Hence, Theorem 1.3 implies the following.

**Corollary 1.5.** \( \ell \)-Chain SAT is (randomized) FPT when parameterized by \( \ell \) and \( k \), even in the weighted setting. Consequently, Conjecture 1.4 is confirmed (with randomized FPT algorithms).

By standard reductions, a Directed Feedback Vertex Set instance can also be represented as a Bundled Cut with Order instance (with maximum size of a bundle and budget bounded linearly in the parameter of the input instance). Furthermore, in case of a weighted variant\(^1\) the reduction preserves weights. Hence, we obtain the following.

\(^1\)In Weighted Directed Feedback Vertex Set, the input graph is equipped with vertex weights being positive integers and one asks for a solution of cardinality at most \( k \) and total weight bounded by a threshold given on input.
Corollary 1.6. Weighted Directed Feedback Vertex Set, parameterized by the cardinality of the deletion set, is (randomized) FPT.

The question of parameterized complexity of Weighted Directed Feedback Vertex Set (which we answer affirmatively in Corollary 1.6) has been asked e.g. at Recent Advances in Parameterized Complexity school in December 2017 [22] and in [13].

We also take the applicability of directed flow-augmentation one step further and prove fixed-parameter tractability of Weighted Almost 2-SAT. Here, we are given a 2-CNF formula \( \phi \), where every clause \( C \) has weight \( \omega(C) \), and positive integers \( k \) and \( W \); the goal is to delete at most \( k \) clauses of total weight at most \( W \) to get a satisfiable instance.

Theorem 1.7. Weighted Almost 2-SAT parameterized by \( k \) is (randomized) FPT.

The FPT algorithm for (unweighted) Almost 2-SAT problem was one of the first applications of important separators and one of the first bridges between graph separation problems and (boolean) constraint satisfaction problems [21].

Compared to the previous applications, applying directed flow-augmentation to Weighted Almost 2-SAT in Theorem 1.7 is much more technically advanced. By known reductions [12], (Weighted) Almost 2-SAT is FPT-equivalent to (Weighted) Digraph Pair Cut. In this problem, we are given a directed graph \( G \), a vertex \( s \in V(G) \), and a number of pairs \( T \subseteq (V(G))^2 \). The goal is to delete at most \( k \) edges of \( G \) such that for every pair \( xy \in T \), either \( x \) or \( y \) is not reachable from \( s \). (In the weighted variant, edges have positive integer weights and we also require that the total weight of deleted edges is not larger than a given threshold.)

In the process of solving a (Weighted) Digraph Pair Cut instance, an algorithm makes some decisions about some pairs \( xy \in T \) whether \( x \) or \( y \) will be not reachable from \( s \) in the final graph. The vertices guessed to be unreachable can be merged into one sink \( t \); the solution \( Z \) needs to be then an \( st \)-cut, but may be required to be more than a minimal \( st \)-cut to satisfy other pairs of \( T \). On the other hand, in an inclusion-wise minimal solution \( Z \) every arc \((u, v)\) serves some purpose: \( u \) is reachable from \( s \) in \( G - Z \) but \( v \) is not. Clearly, every minimal \( st \)-cut is a star \( st \)-cut, but the reverse implication is far from true.

To apply directed flow-augmentation to Weighted Almost 2-SAT, we enhanced the procedure of Theorem 1.1 to star-cuts: the procedure returns a set \( A \subseteq V(G) \times V(G) \) and a maximum \( st \)-flow \( \mathcal{P} \) in \( G + A \) such that for every \( star-st-cut Z \) of size at most \( k \), with probability \( 1 - O(k \log k) \) the set \( Z \) remains a \( star-st-cut \) in \( G + A \) and, furthermore, the edges of \( Z \cap E(\mathcal{P}) \) is exactly a minimal \( st \)-cut in \( G + A \). That is, the arcs of \( A \) make \( Z \) contain a minimal \( st \)-cut and, furthermore, the flow \( \mathcal{P} \) avoids edges of \( Z \) that are not part of the said minimal \( st \)-cut.

Future work. We believe the new technique introduced in this paper opens new research directions. We would like to point out to three of them.

In the Min UNSAT(\( \Gamma \)) problem, we are given an instance of a Constraint Satisfaction Problem over language \( \Gamma \) and an integer \( k \) to delete at most \( k \) clauses to get a satisfiable instance. Here, we restrict only to binary alphabet. Note that if \( \Gamma \) allows unary clauses and equalities, Min UNSAT(\( \Gamma \)) becomes the question of minimum (undirected) \( st \)-cut and when \( \Gamma \) allows unary clauses and implications, Min UNSAT(\( \Gamma \)) becomes the question of minimum (directed) \( st \)-cut.

Recall that the undirected flow-augmentation algorithm [10] led to full classification of the Min UNSAT(\( \Gamma \)) for Boolean alphabet into FPT problems and two [1]-hard problems for languages \( \Gamma \) that are unable to express implication (and thus directed cuts). Up to this work, the main roadblock into generalizing this result to all Boolean languages was the \( \ell \)-CHAIN SAT problem (which is easily expressible in the CSP world). Now this roadblock is removed and exploring this classification is a promising research direction.

Moreover, a similar generalization to the “star \( st \)-cut” was also present in the undirected case and was pivotal to fixed-parameter tractability of Generalized Coupled MinCut problem [10]. We believe the exemplary case of Theorem 1.7 shows that directed flow-augmentation has potential to be a handy tool in proving fixed-parameter tractability of further Boolean Min UNSAT CSP problems.

Corollary 1.6 provides an alternative algorithm for the classic Directed Feedback Vertex Set algorithm, albeit with worse time complexity. We remark that up to now only one approach to fixed-parameter tractability of Directed Feedback Vertex Set was known, namely the one using important separators [1]. A second newly opened research direction is: have we learned anything new about Directed Feedback Vertex Set? To be more precise, let us recall two other important open problems about Directed Feedback Vertex Set: is it solvable in time \( 2^{O(k)} \cdot n^{O(1)} \) and does it admit a polynomial kernel?

For the third direction, we mention that it is possible that directed flow-augmentation can help with resolving the question of the parameterized complexity of the Directed Multicut problem with three terminal pairs, parameterized by the solution size. Recall that this problem is FPT for two terminal pairs [6] and \( W[1] \)-hard for four [20].

2 OVERVIEW

In this section we sketch the proof of Theorem 1.1.

Note that the actual formal statement in the full version [9] is a bit more general: it handles a bit wider family of cuts than only minimal cuts (called star \( st \)-cuts) and provides also a handy object called witnessing flow. While these extensions are critical for the Weighted Almost 2-SAT algorithm, already the case of minimal \( st \)-cuts encompasses main technical ideas and thus this overview focuses only on this case. The full version of the theorem, and associated definitions, are presented at the end of this section.

Basic notation. Before we start, let us agree on basic nomenclature. All our graphs can be multigraphs. Edges may have capacities 1 or \( +\infty \). The input graph has all edge capacities equal to 1. Since we never consider flows of value greater than \( k \), a \( +\infty \)-capacity edge is equivalent to \( (k + 1) \) copies of an edge of capacity 1. If \( G \) is a graph and \( A \subseteq V(G) \times V(G) \), then \( G + A \) is the graph \( G \) with every arc of \( A \) added with capacity \( +\infty \).
For vertices $s, t \in V(G)$, an st-flow is a collection $P$ of paths from $s$ to $t$ such that no edge of capacity 1 lies on more than one path of $P$. The value of the flow is the number of paths. By $\lambda_G(s, t)$ we denote the maximum possible value of an st-flow; note that it may happen that $\lambda_G(s, t) = +\infty$. An st-flow is a maximum st-flow or st-maxflow if its value is $\lambda_G(s, t)$. By convention, if $\lambda_G(s, t) = +\infty$, then any st-flow that contains a path with all edges of capacity $+\infty$ is considered an st-maxflow.

A set $Z \subseteq E(G)$ is an st-cut if it contains no edge of capacity $+\infty$ and there is no path from $s$ to $t$ in $G - Z$. An st-cut $Z$ is minimal if no proper subset of $Z$ is an st-cut and minimum (or st-mincut) if it has minimum possible cardinality. By Menger’s theorem, if $\lambda_G(s, t) = +\infty$, then any st-flow contains a path with all edges of capacity $+\infty$.

For an st-cut $Z$, the s-side of $Z$ is the set of vertices reachable from $s$ in $G - Z$ and the t-side of $Z$ is the complement of the s-side. Note that this is not symmetric as we do not mandate that $t$ is reachable from all elements of the t-side in $G - Z$. If we have two ( inclusion-wise) minimal st-cuts $C_1$ and $C_2$ such that both endpoints of every edge of $C_1$ lie in the s-side of $C_2$ (and thus both endpoints of every edge of $C_2$ lie in the t-side of $C_1$), then a vertex is between $C_1$ and $C_2$ if it is in the t-side of $C_1$ and s-side of $C_2$.

One flow-augmentation step. If we do not pay particular attention to the exact bound on the probability of a success and we are happy with any $1/\ell(k)$ bound for a computable function $\ell$, it suffices to prove the following:

**Theorem 2.1.** There exists a computable function $\ell : \mathbb{N} \to \mathbb{Z}_+$ and a polynomial-time algorithm that, given a directed graph $G$, two vertices $s, t \in V(G)$, and an integer $k$, outputs a set $A \subseteq V(G) \times V(G)$ such that $\lambda_{G+A}(s, t) > \lambda_G(s, t)$ and for every minimal st-cut $Z$ of size at most $k$ that is not a minimum st-cut, $Z$ remains an st-cut in $G + A$ with probability at least $1/\ell(k)$.

Indeed, an algorithm for Theorem 1.1 may first randomly guess the size of the minimal st-cut $Z$ it cares about and then repeat the algorithm of Theorem 2.1, adding the output set $A$ to the graph $G$, as long as $\lambda_G(s, t)$ is smaller than the guessed size of $Z$. Thus, in the remainder of this section we sketch the proof of Theorem 2.1.

For the proof of Theorem 2.1, we fix one minimal st-cut $Z$ of size at most $k$ that is not a minimum st-cut and we bound from below the probability that the algorithm never adds an arc from the s-side to the t-side of $Z$.

The algorithm for Theorem 2.1 is recursive. We start with computing an st-maxflow $P$ in $G$; the goal is to push somehow an extra unit of flow from $s$ to $t$ by adding arcs so that no added arc violates the cut $Z$. The algorithm tries to isolate smaller and smaller parts of the graph through which it tries to push an extra unit of flow, guessing some properties of the cut $Z$ along the way to guide the recursion. To obtain the desired success probability, we need to very restrictively guess properties of the cut $Z$.

Natural candidates as separations between distinct areas for recursive calls are minimum st-cuts. This was also the case for the undirected counterpart of [10]; the algorithm there constructed a maximal sequence of noncrossing and pairwise disjoint minimum st-cuts, partitioned them into minimal subsegments where the graph in-between was connected, and recursed on them.

In the undirected case, the edges of a minimum st-cut $C$ is the entire “interface” between the s-side and t-side of $C$. In the directed case, there can be an unbounded number of arcs with tails in the t-side of $C$ and heads in the s-side of $C$. This makes the interaction between the s-side of $C$ and t-side of $C$ much more complicated and harder to grasp in recursive calls.

One of the main ideas of the proof of Theorem 2.1 — and one of the main differences between it and its undirected counterpart [10] — is a very careful choice of the minimum st-cuts we want to separate along. Let us now introduce it in detail.

Reachability patterns, leaders, and mincut sequences. An instance is a tuple $I = (G, s, t, k)$ where $G$ is a directed graph, $s, t \in V(G)$, and $k$ is a nonnegative integer. An instance with a flow is a pair $(I, P)$ where $I = (G, s, t, k)$ is an instance and $P = \{P_1, P_2, \ldots, P_j\}$ is an st-flow. In an instance with a maximum flow we additionally require $\lambda = \lambda_G(s, t)$, that is, $P$ is a maximum st-flow. With an instance with a flow $(I, P)$ we associate the residual graph $G^P$.

For an instance with a maximum flow $(I = (G, s, t, k), P = \{P_1, P_2, \ldots, P_j\})$, a reachability pattern is a directed graph $H$ with vertex set $V(H) = \{\lambda\}$, that is, each vertex $v \in V(H)$ corresponds to a path $P_i \in P$, that contains a self-loop at every vertex. The pattern associated with $(I, P)$ is a graph $H$ with vertex set $V(H) = \{\lambda\}$ and $(i, j) \in E(H)$ if and only if there exists $v \in V(P_i) \setminus \{s, t\}$ and $u \in V(P_j) \setminus \{s, t\}$ such that there is a $v$ to $u$ path in $G^P$. Note that a pattern associated with $(I, P)$ is a reachability pattern if and only if every path $P_i$ has at least one internal vertex.

For a vertex $v \in V(G)$, the set $RReach(v)$ is the set of vertices reachable from $v$ in $G^P$. For a vertex $v \in V(G)$ and an index $i \in [\lambda]$, the last vertex on $P_i$ residually reachable from $v$, denoted $LastReach(v, P_i)$, is the last (closest to $t$) vertex $u$ on $P_i$ that is reachable from $v$ in $G^P$. Note that if $u$ is reachable from $v$ in $G^P$, then all vertices preceding $u$ on $P_i$ are also reachable from $v$ in $G^P$, that is, the set of vertices of $P_i$ reachable from $v$ in $G^P$ form a prefix of $P_i$. Furthermore, note that if $v_1$ and $v_2$ are on $P_i$ and $v_1$ is earlier than $v_2$ on $P_i$, then $LastReach(v_1, P_i)$ is not later than $LastReach(v_2, P_i)$ on $P_i$ (they may be equal).

Let $C$ be an st-mincut in $I$ and let $H$ be a reachability pattern. For $i \in [\lambda]$, the leader of the path $P_i$ after $C$ (with respect to the pattern $H$), denoted $leader_H(C, i)$, is the first (closest to $s$) vertex $v$ on $P_i$ such that for every $(i, j) \in E(H)$ the vertex $LastReach(v, P_j)$ is after $C$ on $P_j$. A few remarks are in place. First, the notion of the leader is well-defined for any $H$ as the vertex $t$ is always a feasible candidate. Second, since a reachability pattern is required to contain a self-loop at every vertex, for every $i \in [\lambda]$ there is at least one edge $(i, j) \in E(H)$ to consider. Third, as $C$ is oriented from the s-side to the t-side in $G^P$, $leader_H(C, i)$ is after $C$ on $P_i$. A few remarks are in place. First, the notion of the leader is well-defined for any $H$ as the vertex $t$ is always a feasible candidate. Second, since a reachability pattern is required to contain a self-loop at every vertex, for every $i \in [\lambda]$ there is at least one edge $(i, j) \in E(H)$ to consider. Third, as $C$ is oriented from the s-side to the t-side in $G^P$, $leader_H(C, i)$ is after $C$ on $P_i$.
Directed Flow-Augmentation

Directed Flow-Augmentation STOC ’22, June 20–24, 2022, Rome, Italy

Let \( (G, s, t, k) \) be an instance with maximum flow, let \( H \) be the reachability pattern associated with it, let \( C \) be an st-mincut and let \( C' \) be an st-mincut whose endpoints are entirely in the t-side of \( C \). Let \( G' \) be a graph constructed from \( G \) by contracting the s-side of \( C \) onto \( s \) and t-side of \( C' \) onto \( t \) and let \( P' \) be an st-flow in \( G' \) consisting of the subpaths of paths of \( P \) between the edge of \( C \) and the edge of \( C' \). Suppose \( P' \) is an st-maxflow in \( G' \). If \( C' \) is \( H \)-subsequent to \( C \), then the pattern associated with \( (G', s, t, k, P') \) is still \( H \). Moreover, the st-mincut \( H \)-subsequent to \( C \) is an st-mincut closest to \( C \) with this property.

For a reachability pattern \( H \), an \( H \)-sequence of mincuts is a sequence \( C_1, C_2, \ldots, C_t \) of mincuts defined as follows. \( C_1 \) is the st-mincut closest to \( s \) and for \( a > 1 \) the mincut \( C_a \) is the mincut \( H \)-subsequent to \( C_{a-1} \), as long as it is defined. Observe that for every \( 1 \leq a < b \leq t \) and \( i \in [\lambda] \), we have that the edge of \( C_a \) on \( P_i \) lies strictly before the edge of \( C_b \) on \( P_i \). That is, the s-side of \( C_a \) is contained in the s-side of \( C_b \), and, furthermore, on every path \( P_i \) the s-side of \( C_a \) is a strict subset of the s-side of \( C_b \).

Preprocessing and the base case of the recursion. The input to a recursive call is an instance with a flow \( (I = (G, s, t, k), P = \{P_1, \ldots, P_j\}) \) and the goal is to return a set \( A \subseteq V(G) \times V(G) \) such that \( \lambda < \lambda_{G+A}(s, t) \) and for every minimal st-cut \( Z \) with \( \lambda < |Z| \leq k \) the set \( Z \) remains an st-cut in \( G + A \) with good enough probability. In the initial, root set we call \( P \) to be any st-maxflow.

A recursive call first performs a few preprocessing steps. If \( \lambda < \lambda_{G}(s, t) \), it is safe to return \( A = \emptyset \). If \( \lambda_{G}(s, t) = 0 \) or \( \lambda_{G}(s, t) \geq k \), then there is no minimal st-cut of size at most \( k \) that is not a minimum st-cut. If this is the case, we can return \( A = \{(s, t)\} \). Henceforth we assume \( 0 < \lambda_{G}(s, t) < k \) and that \( P \) is an st-maxflow.

A relatively standard preprocessing step (that thus we do not describe here) can ensure that both \( \delta^+(s) \) and \( \delta^-(t) \) (the set of edges with tail in \( s \) and the set of edges with head in \( t \), respectively) are disjoint and are both st-mincuts. In particular, this implies that the pattern \( H \) associated with \( I \) and \( P \) is a reachability pattern.

In the recursive steps, we will aim at either decreasing \( 2k - \lambda \) or keeping \( k \) and \( \lambda \) intact, but decreasing the number of edges in the pattern \( H \) associated with \( (I, P) \). Intuitively, less edges in \( E(H) \) means that the graph \( G \) is somewhat simpler.

In the base case of the recursion, \( H \) consists of \( \lambda \) isolated vertices, with a loop on every vertex of \( V(H) = [\lambda] \). In other words, \( G^P \) contains no path from a vertex of \( V(P_i) \) \( \{s, t\} \) to a vertex of \( V(P_j) \) \( \{s, t\} \) for any distinct \( i, j \in [\lambda] \), or equivalently \( G \) \( \{s, t\} \) contains no path from a vertex of \( V(P_i) \) to a vertex of \( V(P_j) \). In this case, the paths \( P_i \) are somewhat independent in \( G \) and it is relatively easy to show the following. (Here, an edge \( e \in E(G) \) is bottleneck if there exists a minimum st-cut \( C \) with \( e \in C \).

Lemma 2.3. Let \((I, P)\) be an instance with maximum flow with proper boundaries and let \( H \) be its reachability pattern. If \( |V(H)| = |E(H)| \), then for every minimal st-cut \( Z \) that is not a minimum cut, there exists \( i \in [\lambda] \) such that no edge of \( Z \) \( E(P_i) \) is a bottleneck edge.

Lemma 2.3 allows the following step: sample \( i \in [\lambda] \) at random and return \( A \) consisting of duplicates of all bottleneck edges of \( P_i \). Lemma 2.3 ensures that with probability at least \( \lambda^{-1} \geq k^{-1} \), a minimal st-cut that is not a minimum st-cut remains an st-cut in \( G + A \), while \( \lambda_{G+A}(s, t) > \lambda_{G}(s, t) \) follows from the fact that we duplicated all bottleneck edges on one flow path.

If \(|E(H)| > |V(H)|\), we aim at decomposing the graph further and make some recursive calls. We compute the \( H \)-sequence of mincuts \( C_1, C_2, \ldots, C_t \). Note that we have \( C_1 = \delta^+(s) \) and it is easy to see that \( C_2 \) is defined, and thus \( t \geq 2 \). We set a threshold \( \delta_{\text{std}} \) depending on \( k \) (in the actual proof we have \( \delta_{\text{std}} \) being polynomially bounded in \( k \)) and split into two cases: \( t \leq \delta_{\text{std}} \) or \( t > \delta_{\text{std}} \).

Small \( t \) case. In this case, we can afford guessing (by random coin flip) for each \( i \in [t] \) and \( e \in C_i \); whether the endpoints of \( e = (u, v) \) are in the s-side or t-side of \( Z \).

There are a few simple cases. If for some \( e = (u, v) \) we guessed that \( u \) is in the s-side of \( Z \) and \( v \) is in the t-side, then \( e \in Z \); we can recurse on \( G' := G - e \) and \( P' := P \setminus \{P_i\} \) with parameter \( k - 1 \), where \( P \in P \) is a flow path containing \( e \), and obtain a set \( A' \), and return \( A := A' \cup \{(s, u), (v, t)\} \). If for some \( e = (u, v) \) we guessed that \( u \) is in the t-side of \( Z \) and \( v \) is in the s-side of \( Z \), then we can return \( A := \{(s, v), (u, t)\} \) as \( s - v - u - t \) is an augmenting path in \( G + A \) with respect to \( P' \), so \( \lambda_{G+A}(s, t) > \lambda \). Finally, if for every \( e \in C_t \), both endpoints of \( e \) are in the s-side of \( Z \), then we can recurse on \( G' \) being the graph \( G \) with the s-side of \( C_t \) contracted onto \( s \); the fact that \( C_{t+1} \) is undefined implies that in the recursive call the associated pattern is a proper subgraph of \( H \).

In the main case, let \( A_0 \) consist of edges \( (s, v) \) for any endpoint \( v \) of an edge \( e \in C_i \), \( i \in [t] \) that is guessed to be in the s-side of \( Z \) and edges \( (u, t) \) for any endpoint \( u \) of an edge \( e \in C_i \), \( i \in [t] \) that is guessed to be in the t-side of \( Z \). If the guess is correct, \( Z \) remains an st-cut in \( G + A_0 \). If \( \lambda_{G+A_0}(s, t) > \lambda \), then we can return \( A_0 \), so assume otherwise.

Let \( C \) be the minimum st-cut in \( G + A_0 \) that is closest to \( t \). For every endpoint of an edge of \( C \), we guess whether it is in the s-side or t-side of \( Z \). We again have a few simple cases. If for some \( e = (u, v) \in C \), \( u \) is in the s-side of \( Z \) and \( v \) is in the t-side of \( Z \), we have \( e \in Z \) and we recurse on \( G - e \) as before. If a head of an edge of \( C \) is in the s-side of \( Z \), then we return \( A := A_0 \cup \{(s, v)\} \); \( \lambda_{G+A_0}(s, t) > \lambda \), follows from the fact that \( C \) is the closest to \( t \) minimum st-cut in \( G + A_0 \).

We are left with the most interesting case where \( C \) is completely in the t-side of \( Z \). We define \( a \in [t] \) to be the maximum index such that all endpoints of \( C_a \) are in the s-side of \( Z \). This is well-defined as \( C_1 = \delta^+(s) \) is in the s-side of \( Z \) (as we are not in any of the simple cases). Also we have \( a < t \) and \( C_{a+1} \) is defined (again, we are not in any of the simple cases) and also \( C \) lies entirely in the t-side of \( C_a \).

Let \( G' \) be the graph \( G \) with the s-side of \( C_a \) contracted onto \( s \) and
the $t$-side of $C$ contracted onto $t$. Let $\mathcal{P}'$ be the flow $\mathcal{P}$ projected onto $G'$ (i.e., we shorten all paths to start from the edges of $C_a$ and end with the edges of $C$).

The crucial observation is that the pattern associated with $G'$ and $\mathcal{P}'$  is a proper subgraph of $H$. Indeed, by the choice of $a$ (and exclusion of the simple cases), there is at least one tail $u$ of an edge of $C_{a+1}$ that is in the $t$-side of $Z$; this vertex $u$ lies in the $t$-side of $C_a$ and the $s$-side of $C_{a+1}$. Moreover, $u$ is in the $t$-side of $C$ as the edge $(u,t)$ has been added to $A_0$ and $C$ is an st-mincut in $G + A_0$. Since $C_{a+1}$ is the mincut $H$-subsequent to $C_a$, by Lemma 2.2 the subgraph between $C_a$ and $C_{a+1}$ maintains all reachability (in the residual graph) described by $H$, but $C_{a+1}$ is chosen to be the closest-to-$C_a$ minimum st-cut that satisfies Lemma 2.2. Since $u$ is in the $t$-side of $C$ but $s$-side of $C_{a+1}$, we can infer that for at least one edge of $H$, the corresponding reachability is not present in $G'$ and $\mathcal{P}'$. We recurse on $G'$ and $\mathcal{P}'$; this decrease in $|E(H)|$ is a progress in the recursive step.

We remark here that the above progress in the recursion is the main reason to introduce the notions of reachability patterns and sequences of mincuts. It seems to us very delicate; we were not able to reproduce a similar measure of progress with different ways of defining sequences of cuts $C_1, \ldots, C_j$.

Large $t$ case. The progress in the previous case is possible partially due to the fact that $t$ is bounded in $k$ and we could afford guessing the $s$-side/$t$-side assignment of all endpoints of all cuts $C_i$. If $t$ is large (unbounded in $k$), we need to resort to a color-coding step to split the long sequence of $C_i$’s into smaller chunks that are eligible for the small $t$ case.

Let $Z_{t^k}$ be the set of edges of paths of $\mathcal{P}$ whose tail is in the $t$-side of $Z$ but head is in the $s$-side of $Z$. On each flow path $P_i$, we have $|E(P_i) \cap Z| = |E(P_i) \cap Z_{t^k}| + 1$ and thus $|Z_{t^k}| \leq k - \lambda$. An index $a \in [t]$ is touched if there is an endpoint of an arc of $Z \cup Z_{t^k}$ that is in the $t$-side of $C_a$ and $s$-side of $C_{a+1}$. We have at most $2|Z \cup Z_{t^k}| \leq 4k - 2\lambda$ touched indices.

Recall that the construction of the cuts $C_1, C_2, \ldots, C_j$ ensures that, for every $a \in [t - 1]$, between $C_{a}$ and $C_{a+1}$ one can find a path from $P_0$ to $P_j$ in the residual graph for every $(i, j) \in E(H)$. On the other hand, $G - \{s,t\}$ features no path in the residual graph from $P_0$ to $P_j$ if $(i,j) \notin E(H)$. In some sense, it means that the connectivity between paths $P_i$ that is present in the entire graph $G$ is repeatedly realized between each two consecutive cuts $C_a$.

The fact that we are operating with the residual graph was essential for the small $t$ case. Here, we need to depart from the residual graph and observe the following: for every $(i, j) \in E(H)$ and $1 \leq a \leq \ell - \lambda$, there is a path in $G$ from $P_1$ to $P_j$ that starts between $C_a$ and $C_{a+1}$, ends between $C_{a+\lambda-1}$ and $C_{a+\lambda}$, and is contained between $C_a$ and $C_{a+\lambda}$.

This has a number of consequences. First, if $\ell^{|BR|}$ is large enough, then $H$ must be transitive.

Second, for every $a \in [t]$ let $L_a \subseteq [\ell]$ be the set of indices $i$ such that both endpoints of the unique edge of $E(P_i) \cap C_a$ are in the $s$-side of $Z$. Observe that if for some $1 \leq a \leq \ell - \lambda$, all indices $a \leq b \leq a + \lambda$ are untouched, then the set $L_{a+\lambda}$ is downward-closed: there is no arc $(i,j) \in E(H)$ such that $i \in L_{a+\lambda}$ but $j \not\in L_{a+\lambda}$.

Third, observe that if $L \subseteq [\ell]$ is downward-closed, then for every $i \in L$ and $j \not\in L$, the graph $G - \{s,t\}$ contains no path from $P_i$ to $P_j$. Hence, if we denote by $cl(L_a)$ be the minimal superset of $L_a$ that is downward-closed, we have that $L_a \subseteq cl(L_a)$ whenever $1 \leq a \leq b \leq \ell$.

In particular, if $L_a$ is downward-closed, we have the following.

1. $L_b \subseteq L_a$ for $b \geq a$.
2. For every $i \in L_a$, the entire prefix of $P_i$ up to the edge of $C_a$ is in the $s$-side of $Z$, as every path from a vertex $s$ on such prefix to $t$ needs to intersect $C_a \cup \bigcup_{j \in L_a} E(P_j)$ (and $Z$ is a minimal st-cut).
3. Symmetrically for every $i \not\in L_a$, the entire suffix of $P_i$ from the edge of $C_a$ is in the $t$-side of $Z$.

4. There is no edge $e = (u,v) \in Z$ with $u$ in the $t$-side of $C_a$ and $v$ in the $s$-side of $C_a$. Indeed, if $e = (u,v)$ were such an edge, then by minimality of $Z$ the graph $G - Z$ would need to feature a path from $s$ to $v$ and from $u$ to $t$; the first path needs to traverse an edge of $C_a \cup E(P_i)$ for $i \in L_a$ and the second path needs to traverse an edge of $C_a \cup E(P_j)$ for $j \not\in L_a$, hence in between we have a path from $P_i$ to $P_j$ in $G - \{s,t\}$.

Let $T$ be the set of touched indices. We say that two indices $a < b$ are close if $b - a \leq \lambda$. The set $T$ partitions into blocks: maximal sets of indices such that two consecutive indices are close to each other. Let $B_1, \ldots, B_r$ be the blocks and for $1 \leq a \leq \lambda$, let $a_0$ and $b_0$ be the minimum and maximum index of the block $B_a$. We have $b_0 - a_0 = O(k\lambda)$ while indices $a_\lambda - \lambda, a_\lambda - \lambda + 1, \ldots, a_\lambda - 1$ and $b_\lambda + 1, \ldots, b_\lambda + \lambda$ are untouched (we ignore here boundary cases when $a_\lambda \leq \lambda$ or $b_\lambda + \lambda > \ell$ which are easy to adjust to). In particular, $L_{a_\lambda - 1}$ and $L_{b_\lambda + \lambda}$ are downward-closed. Denote $L_{a_\lambda} = L_{a_\lambda - 1}, L_{a_\lambda} = L_{b_\lambda + \lambda}$, and $D^\ell = L_{a_\lambda} \setminus L_{a_\lambda}'$.

An important observation from the downward-closedness of $L_{a_\lambda}'$ and $L_{a_\lambda}''$ is the following: for every edge $e \in Z$, there is a block $B_a$ such that both endpoints of $e$ lie between $C_{a_\lambda}$ and $C_{b_\lambda + 1}$. Hence, $Z$ partitions into $Z_1 \cup Z_2 \cup \ldots \cup Z_{\ell}$ where $Z_{a_\lambda}$ is the set of edges of $Z$ with both endpoints between $C_{a_\lambda}$ and $C_{b_\lambda + 1}$.

For every $a \in [t]$, define a graph $G_a$ as follows: contract the $s$-side of $C_{a_\lambda - 1}$ to $s$, contract the $t$-side of $C_{b_\lambda + 1}$ onto $t$, and from the edges incident with $s$ or $t$ leave only the ones on paths $P_i$ for $i \in D_a$. The flow paths $P_i$ for $i \in D_a$ naturally project to a flow $\mathcal{P}_a$ of size $|D^\ell|$ in $G_a$. The important observation again from the downward-closedness of $L_{a_\lambda}'$ and $L_{a_\lambda}''$ is as follows: in $G_a$, $Z_{a_\lambda}$ is a minimal st-cut.

This in particular implies that $D^\ell \neq \emptyset$ and thus

$$|\lambda| = L_{a_\lambda}' \geq L_{a_\lambda} \geq L_{a_\lambda}'' \geq L_{a_\lambda}''' = \ldots = L_{a_\lambda}''' \geq L_{a_\lambda}''' = 0.$$ 

We would like to recurse on instances $I_a = (G_a, \mathcal{P}_a, k_a)$ where $k_a = |Z_{a_\lambda}|$, but there is a significant problem: we do not know the blocks $B_a$. However, following a similar step in the undirected algorithm of [10], we can deal with it with color-coding.

Sample $\Gamma \subseteq [t]$, aiming at follows: for every $a$ that is close to a touched index, we want $a \in \Gamma$ if and only if $a$ is touched. If we put every $a \in [t]$ into $\Gamma$ with probability $0.5$ independently of other indices, the success probability is $2^{-O(k\lambda)}$. Then, the indices of $\Gamma$ partition into $\Gamma$-blocks: one $\Gamma$-block is a maximal subset of $\Gamma$ in which two consecutive indices are close. If the guess is successful, every block is a $\Gamma$-block, but there may be numerous $\Gamma$-blocks that are in fact wholly untouched.
We can guess $r \leq 4k - 2\lambda$, sets $L_\alpha^+, L_\alpha^-$ as well as sizes $k_\alpha = |Z_\alpha|$ for $\alpha \in \{r\}$. For every $\Gamma$-block, we guess its “type” in $\{r\}$, aiming that a true block $B_\alpha$ has type $\alpha$. For a $\Gamma$-block $B$ of type $\alpha(B)$, we construct from it an instance $(G_\alpha, s, t, k_\alpha(B))$ with flow $P_\alpha$ in the same way as we constructed $G_\alpha(B)$ and $P_\alpha(B)$ for $B_\alpha$. If $k_\alpha(B) > |D_\alpha(B)|$ (i.e., $Z_\alpha(B)$ is not a minimum cut in $G_\alpha(B)$) we recurse on it, obtaining a set $A_B$ (we denote $A_B = \emptyset$ for cases $k_\alpha(B) = D_\alpha(B)$).

The final set $A$ consists of:

1. for every $\Gamma$-block $B$ with minimum index $a$ and maximum index $b$, the following edges:
   (a) for every $(u, v) \in A_B$:
   - if $u \neq s$ and $v \neq t$, just the arc $(u, v)$ itself;
   - if $u = s$ and $v \neq t$, all arcs $(u', v')$ where $u'$ ranges over all tails of edges of $C_{t-1}$ on paths $P_i$ for $i \in D_\alpha;
   - if u \neq s$ and $v = t$, all arcs $(u, v')$ where $v'$ ranges over all tails of edges of $C_{t-1}$ on paths $P_i$ for $i \in D_\alpha;
   - if u = s$ and $v = t$, all arcs $(u', v')$ where $v'$ ranges over all edges of $C_{t-1}$ on paths $P_i$ for $i \in D_\alpha$ and $v'$ ranges over all tails of edges of $C_{b+1}$ on paths $P_i$ for $i \in D_\alpha$;
   (b) for every $i \in \{1\} \setminus D_\alpha$ and arc from the tail of the edge of $C_{a-1} \cap E(P_i)$ to the tail of the edge of $C_{b+1} \cap E(P_i)$;
2. for every $1 \leq c \leq t$ for which there is no $\Gamma$-block $B$ with minimum index $a$ and maximum index $b$ and $a-1 \leq c \leq b+\lambda$:
   - for every $i \in \{1\}$, an edge from the tail of the edge of $C_{t} \cap E(P_i)$ to the tail of the edge of $C_{a} \cap E(P_i)$;
3. for every $1 \leq c \leq t$ for which there is no $\Gamma$-block $B$ with minimum index $a$ and maximum index $b$ and $a-1 \leq c \leq b+\lambda$:
   - for every $1 \leq \alpha \leq r$, for every $i, j \in D_\alpha$, an edge from the tail of the edge of $C_{t} \cap E(P_j)$ to the tail of the edge of $C_{t} \cap E(P_j)$.

To see that $\lambda_{g+\lambda}(s, t) > \lambda$, notice first that as $Z$ is not a minimum $st$-cut, there exists an index $\alpha$ where $k_\alpha = |Z_\alpha| > D_\alpha$. We push an extra unit of flow along flow paths $P_i, i \in D_\alpha$ as follows.

- For every $\Gamma$-block $B$ of type $\alpha$ (with minimum index $a$ and maximum index $b$), we push $|D_\alpha| + 1$ units of flow from the tails of edges of $C_{a-1}$ on paths $P_i$ for $i \in D_\alpha$ to the tails of edges of $C_{b+1}$ on paths $P_i$ for $i \in D_\alpha$ using edges of $A_\alpha$ (Point 1a) and inductive assumption from the recursive call.
- For every $\Gamma$-block $B$ of type $\alpha'$, we push $|D_\alpha| + 1$ units of flow along edges added in Point 1b for paths $P_i, i \in D_\alpha$.
- For every $1 \leq c \leq t$ for which there is no $\Gamma$-block $B$ with minimum index $a$ and maximum index $b$ and $a-1 \leq c \leq b+\lambda$, we push $|D_\alpha| + 1$ units of flow along edges added in Point 2 for paths $P_i, i \in D_\alpha$.
- We use edges added in Point 3 to reshuffle the flow on paths $P_i, i \in D_\alpha$ between the steps above, if needed.

Let us now briefly analyse the progress in the above recursion. If $r > 1$, then all recursive calls treat strictly smaller value of $k$. For $r = 1$, we have $D_1 = \lambda$ and possibly $k_\alpha = k$. However, then the recursive calls fall into the “small $\Gamma$ case” as the cuts $C_{\alpha'}$ inside recursive call remain an $H$-sequence of mincuts. This is the desired progress and it finishes the overview of the proof of Theorem 2.1.

## 2.1 Star cuts

For more advanced applications of the framework, such as the algorithm for WEIGHTED ALMOST 2-SAT, we need the full version of Theorem 2.1, which we present here. The proofs are deferred to the full version [9].

An $st$-cut $Z$ is a star $st$-cut if for every $(u, v) \in Z$, in the graph $G - Z$ there is a path from $s$ to $u$ but there is no path from $s$ to $v$. Note that every minimal $st$-cut is a star $st$-cut, but the implication in the other direction does not hold in general. For a star $st$-cut $Z$ in $G$, by $\text{core}_{G}(Z) \subseteq Z$ we denote the set of arcs $(u, v) \in Z$ such that there exists a path from $v$ to $t$ in $G - Z$. We drop the subscript if the graph $G$ is clear from the context. We observe the following.

**Lemma 2.4.** If $Z$ is a star $st$-cut in a graph $G$, then core$(Z)$ is a minimal $st$-cut.

We say that a set of arcs $A \subseteq V(G) \times V(G)$ is compatible with a star $st$-cut $Z$ if the following holds: for every $(u, v) \in V(G)$, there is a path from $s$ to $v$ in $G - Z$ if and only if there is a path from $s$ to $v$ in $(G + A) - Z$. Equivalently: the $s$-sides and $t$-sides of $Z$ are equal in $G$ and $G + A$, or no arc of $A$ has its tail in the $s$-side of $Z$ in $G$ and its head in the $t$-side of $Z$ in $G$.

An immediate yet important observation is as follows.

**Lemma 2.5.** If $Z$ is a star $st$-cut in $G$ and $A \subseteq V(G) \times V(G)$ is compatible with $Z$, then $Z$ is a star $st$-cut in $G + A$ as well.

We remark that albeit in the setting of Lemma 2.5 the cut $Z$ may add some new reachability towards $t$.

Assume $Z$ is a star $st$-cut in $G$ such that $\text{core}(Z)$ is an $st$-mincut. An $st$-maxflow $P$ is a witnessing flow if $E(P) \cap Z = \text{core}(Z)$, that is, $P$ contains one edge of core$(Z)$ on each flow path and no other edge of $Z$. A witnessing flow may not exist in general, even if core$(Z)$ is an $st$-mincut. In contrast, on undirected graphs an arbitrary $st$-maxflow is a witnessing flow for the counterpart of core$(Z)$ when the latter is an $st$-mincut [10]. However, our flow-augmentation procedure will ensure that not only core$(Z)$ becomes an $st$-mincut in the augmented graph, but also a flow is returned that is a witnessing flow in the augmented graph. Formally, for a star $st$-cut $Z$ in $G$, $A \subseteq V(G) \times V(G)$, and an $st$-maxflow $\tilde{P}$ in $G + A$, we say that $(A, \tilde{P})$ is compatible with $Z$ if $A$ is compatible with $Z$, $\text{core}_{G+A}(Z)$ is an $st$-mincut in $G + A$, and $\tilde{P}$ is a witnessing flow for $Z$ in $G + A$. Our flow-augmenting procedure will return a pair $(A, \tilde{P})$ that is compatible with a fixed star $st$-cut $Z$ with good probability.

**Theorem 2.6.** There exists a polynomial-time algorithm that, given an instance $I = (G, s, t, k)$, returns a set $A \subseteq V(G) \times V(G)$ and an $st$-maxflow $\tilde{P}$ in $G + A$ such that for every star $st$-cut $Z$ of size at most $k$, with probability $2^{-O(k \log k)}$ the pair $(A, \tilde{P})$ is compatible with $Z$.

## 3 APPLICATIONS: WEIGHTED $st$-CUT, WEIGHTED DFVS, WEIGHTED CHAIN SAT

**3.1 Tractable case of Weighted Bundled Cut**

An instance of Bundled Cut consists of a directed graph $G$, vertices $s, t \in V(G)$, a nonnegative integer $k$, and a family $\mathcal{B}$ of pairwise
disjoint subsets of $E(G)$. An element $B \in \mathcal{B}$ is called a bundle. An edge that is part of a bundle is deletable, otherwise it is undeletable. A cut in a Bundled Cut instance $I = (G, s, t, k, \mathcal{B})$ is an st-cut $Z$ that does not contain any undeletable edge, that is, $Z \subseteq \mathcal{B}$. A cut $Z$ touches a bundle $B \in \mathcal{B}$ if $Z \cap B \neq \emptyset$. The cost of a cut $Z$ is the number of bundles it touches. A cut $Z$ is a solution if its cost is at most $k$. The Bundled Cut problem asks if there exists a solution to the input instance.

An instance of Weighted Bundled Cut consists of a Bundled Cut instance $I = (G, s, t, k, \mathcal{B})$ and additionally a weight function $\omega : \mathcal{B} \to \mathbb{Z}_+$, and an integer weight $W$. The weight of a cut $Z$ in $I$ is the total weight of all bundles it touches. A solution $Z$ to $I$ is a solution to the Weighted Bundled Cut instance $(I, \omega, W)$ if additionally the weight of $Z$ is at most $W$. The Weighted Bundled Cut problem asks if there is a solution to the input instance. Note that any Bundled Cut instance can be treated as a Weighted Bundled Cut instance by setting $\omega$ uniformly equal $1$ and $W = k$.

When parameterized by $k$, it is well known that Bundled Cut is $W[1]$-hard even if all bundles are of size $2$ [18]. To get tractability, we define the following restriction called Weighted Bundled Cut with Order where we require the following: in the input instance $((G, s, t, k, \mathcal{B}), \omega, W)$, for every $B \in \mathcal{B}$ one can order the arcs of $B$ as $e_{i}^{B}, \ldots, e_{m}^{B}$ such that for every $1 \leq i < j \leq |B|$ there exists a path $P_{ij}^{B}$ in $G$ that goes from one endpoint of $e_{i}^{B}$ to one endpoint of $e_{j}^{B}$ and uses only undeletable arcs and arcs of $B$.

In this subsection we prove the following theorem.

**Theorem 3.1.** **Weighted Bundled Cut with Order** is randomized FPT when parameterized by $k$ and maximum size of a bundle.

The main workhorses behind the proof of Theorem 3.1 are the flow-augmentation routine and the following lemma.

**Lemma 3.2.** **Assume we are given a Weighted Bundled Cut with Order instance $I = ((G, s, t, k, \mathcal{B}), \omega, W)$ Then one can in time FPT with parameters $\lambda_{\mathcal{C}}(s, t)$, maximum size of a bundle, and $k$ check if there is a solution to $I$ that is an st-mincut, where for $\lambda_{\mathcal{C}}(s, t)$ we treat every deletable edge as of capacity $1$ and every undeletable edge as of capacity $+\infty$.**

**Proof of Theorem 3.1 using Lemma 3.2.** Let $b$ be the maximum size of a bundle. We will describe a randomized algorithm that runs in time FPT in $b$ and $k$ and, if given a yes-instance, answers yes with probability $2^{-O((bk)^{4}\log(bk))}$, and never answers yes to a no-instance. Repeating the algorithm $2^{O((bk)^{4}\log(bk))}$ times gives the desired algorithm.

Consider a solution $Z$ to the input Weighted Bundled Cut with Order instance $I = ((G, s, t, k, \mathcal{B}), \omega, W)$. Without loss of generality, we can assume that $Z$ is a minimal st-cut. Note that $|Z| \leq bk$. Apply flow-augmentation (Theorem 1.1) to $G$ and parameter $bk$, obtaining a set $A$, and add all arcs of $A$ to $G$ as undeletable arcs. Note that adding arcs $A$ to $G$ does not break the requirements for a Weighted Bundled Cut with Order instance. With probability $2^{-O((bk)^{4}\log(bk))}$ the set $Z$ is still a solution in the final instance and is an st-mincut. Run the algorithm of Lemma 3.2, returning its answer.

Thus, we are left with proving Lemma 3.2.

**Proof of Lemma 3.2.** For every bundle $B \in \mathcal{B}$, fix an order $B = (e_{1}^{B}, \ldots, e_{|B|}^{B})$ as in the definition of a Weighted Bundled Cut with Order instance. Let $b$ be the maximum size of a bundle.

Compute a maximum st-flow $P = (P_{1}, P_{2}, \ldots, P_{b})$ with $\lambda := \lambda_{\mathcal{C}}(s, t)$. Fix a hypothetical solution $Z$; note that $Z$ contains exactly one deletable edge on every path $P_{i}$ denote by $f_{i}$ the unique edge of $E(P_{i}) \cap \mathcal{B}$. Branch, guessing the following information about $Z$. First, guess the number $\delta \leq k$ of bundles touched by $Z$. Let those bundles be $F_{1}, F_{2}, \ldots, F_{\delta} \in \mathcal{B}$. For every $i \in [\delta]$, guess the indices $a(i) \in [k]$ and $b(i) \in [b]$ such that $f_{i} = e_{a(i)}^{F_{b(i)}}$. There are $2^{O(\lambda \log(bk))}$ options.

We perform a simple sanity check we consider only options where the mapping $i \mapsto (a(i), b(i))$ is injective.

We also guess a partition of $B$ into sets $B_{1}, \ldots, B_{k}$, aiming at $F_{j} \in B_{j}$. By standard objects for derandomizing color-coding, this can be turned into a branch into $2^{O(\lambda \log(bk))}$ options.

For every $1 \leq j \leq k$ and every $B \in B_{j}$ we make a sanity check: we expect that, for every $i \in [\delta]$ such that $a(i) = j$, the edge $e_{\beta(i)}^{B}$ actually lies on $P_{i}$. If this is not the case, we remove $B$ from $B_{j}$ and $\mathcal{B}$ (making its edges undeletable). Clearly, $F_{j}$ remains in $B_{j}$ in the branch where the guesses are correct.

We now make the following (main) filtering step. Iterate over all $1 \leq j \leq k$ and indices $1 \leq i_{1}, i_{2} \leq \lambda$ such that $a(i_{1}) = a(i_{2}) = j$, but $\beta(i_{1}) < \beta(i_{2})$. Consider $B \in B_{j}$ such that there exists another $B' \in B_{j}$ such that $e_{\beta(i_{1})}^{B'}$ is before $e_{\beta(i_{2})}^{B}$ on $P_{i}$, but $e_{\beta(i_{2})}^{B'}$ is after $e_{\beta(i_{1})}^{B}$ on $P_{i}$. Assume $B = F_{j}$. Then $B' \cap Z = \emptyset$. By the property of the Weighted Bundled Cut with Order instance, $G$ contains a path $Q$ from an endpoint of $e_{\beta(i_{1})}^{B'}$ to an endpoint of $e_{\beta(i_{2})}^{B}$, that uses only undeletable edges and edges of $B'$. Hence, $Q$ is disjoint with $Z$. However, as $e_{\beta(i_{1})}^{B} \in Z$ lies after $e_{\beta(i_{1})}^{B'}$ on $P_{i}$, while $e_{\beta(i_{2})}^{B} \in Z$ lies before $e_{\beta(i_{2})}^{B'}$ on $P_{i}$, $Q$ goes from the s-side of $Z$ to the t-side of $Z$, a contradiction. Hence, $B \neq F_{j}$ and we can remove such $B$ from $B_{j}$ and $\mathcal{B}$.

After we perform the filtering step exhaustively, for every $1 \leq j \leq k$ we can order $B_{j}$ as $B_{j_{1}}, \ldots, B_{j_{|B_{j}|}}$ such that for every $1 \leq \xi < \nu \leq |B_{j}|$ and for every $i \in [\lambda]$, if $a(i) = j$, then $B_{j_{\nu}}$ is before $e_{\beta(i)}^{B_{j_{\xi}}}$ on $P_{i}$. Let $A_{j}$ be such that $F_{j} = B_{j_{|B_{j}|}}$.

In the end, we also check if still $\lambda = \lambda_{\mathcal{C}}(s, t)$. If this is not the case, we terminate the current branch.

An edge $e \in E(G)$ is vulnerable if $e = e_{a(i)}^{B}$ for some $i \in [\lambda]$ and $B \in B_{a(i)}$. Note that all edges of $Z$ are vulnerable, all vulnerable edges are deletable, and vulnerable edges appear only on paths of $P$.

We now construct an auxiliary weighted directed graph $H$ as follows. Start with $H$ consisting of two vertices $s$ and $t$. For every $1 \leq i \leq k$, add a path $P_{ij}^{B}$ from $s$ to $t$ with $[B_{j}]$ edges; denote the $a$-th edge as $e_{ij}^{a}$ and set its weight as $\omega(e_{ij}^{a}) = \omega(B_{j_{a}})$. Furthermore, for every $1 \leq i_{1}, i_{2} \leq \lambda$, denote $j_{1} = a(i_{1}), j_{2} = a(i_{2})$, for every $1 \leq a_{1} \leq |B_{j_{1}}|$ and $1 \leq a_{2} \leq |B_{j_{2}}|$, for every endpoint $u_{1}$ of $e_{\beta(i)}^{B_{j_{1}}}$, for every endpoint $u_{2}$ of $e_{\beta(i)}^{B_{j_{2}}}$, if $G$ contains a path from $u_{1}$ to $u_{2}$ consisting only of nonvulnerable edges, then add to $H$ an edge of weight $+\infty$ from the corresponding endpoint of $e_{ij}^{a}$ (i.e., tail if
and only if \( u_1 \) is a tail of \( e^{B_{\beta(1)}}_i \) to the corresponding endpoint of \( e_{j_2,a_2} \) (i.e., tail if and only if \( u_2 \) is a tail of \( e^{B_{\beta(2)}}_i \)).

Observe that \( Z' := \{ e_{j,a} \mid 1 \leq j \leq k \} \) is an st-cut in \( H \). Indeed, if \( H \) would contain an arc \((v,u)\) with \( v \) before \( e_{j,a} \) and \( u \) after \( e_{j',a'} \) for some \( j, j' \in [k] \), then this arc was added to \( H \) because of some path between the corresponding endpoints in \( G \) and such a path would lead from the \( s \)-side to \( t \)-side of \( Z \). Also, in the other direction, observe that if \( Y' = \{ e_{j,a} \mid 1 \leq j \leq k \} \) is an st-mincut in \( H \), then

\[
Y = \left\{ e^{B_{\beta(i)}}_i \mid i \in [\lambda] \right\}
\]

is a solution to \( I \) of the same weight.

Hence, it suffices to find in \( H \) an st-cut of cardinality \( \kappa \) and minimum possible weight. Since \( \kappa \leq \lambda_H(s,t) \), this can be done in polynomial time.

\( \hfill \square \)

### 3.2 Applications

In all three discussed problems, we are given both a bound \( k \) on the cardinality of the solution and a bound \( W \) on the total weight of the solution. The parameter is \( k \).

In \textsc{Weighted Cut}, the solution is an st-cut. In \textsc{Weighted DFAS}, the solution is a feedback arc set in a directed graph. Note that by standard reduction, this is equivalent to \textsc{Weighted DFVS}. If \textsc{Weighted b-Chain SAT}, the problem is in fact a \textsc{Weighted Bundled Cut} where every bundle has a path of length at most \( b \).

Note that \textsc{Weighted Cut} is \textsc{Weighted Bundled Cut} with bundles of size 1 and no undeletable edges while \textsc{Weighted b-Chain SAT} is \textsc{Weighted Bundled Cut} with bundles of size at most \( b \). Furthermore, in both problems the assumption for \textsc{Weighted Bundled Cut with Order} is satisfied (in the latter case, order the edges along the path). We infer the following.

**Theorem 3.3.** \textsc{Weighted Cut} is randomized FPT when parameterized by \( k \).

**Theorem 3.4.** \textsc{Weighted b-Chain SAT} is randomized FPT when parameterized by \( k \) and \( b \).

By standard approach, \textsc{Weighted DFAS} can be solved using a subroutine for \textsc{Weighted Skew Multicut}. Here, we are given a directed graph \( G \), a tuple \((s_i, t_i)_{i=1}^b\) of terminal pairs, a weight function \( \omega : E(G) \rightarrow \mathbb{Z}_+ \), and integers \( k, W \). The goal is to find a set \( Z \subseteq E(G) \) of cardinality at most \( k \), weight at most \( W \), and such that there is no path from \( s_i \) to \( t_j \) in \( G - Z \) for any \( 1 \leq i \leq j \leq b \). We observe the following reduction.

**Lemma 3.5.** Given a \textsc{Weighted Skew Multicut} instance \( I = (G, (s_i, t_i)_{i=1}^b, \omega, k, W) \), one can in polynomial time construct an equivalent \textsc{Weighted Bundled Cut} with Order instance \( I' = (G', B, \omega, k, W) \) with the same \( k \) and \( W \) and bundles of size \( b \) each.

**Proof.** To construct the graph \( G' \), we start with \( b \) disjoint copies \( G^1, \ldots, G^b \) of the graph \( G \). By \( e^1, e^2 \), etc., we denote the copy of vertex \( v \) or edge \( e \) in the copy \( G^i \). We set \( B = \{ e^i \mid i \in [b] \} \) for \( e \in E(G) \) and \( B = \{ B_e \mid e \in E(G) \} \), that is, all \( b \) copies of one edge of \( G \) form a bundle. We set weights of bundles as \( \omega'(B_e) = \omega(e) \).

There will be no more bundles, so all arcs introduced later to \( G' \) are undeletable.

For every \( 1 \leq i < j \leq b \) and \( v \in V(G) \), we add to \( G' \) an arc \((e^i, e^j, v)\). These arcs make the instance satisfy the "order" property with the natural order \((e^1, e^2, \ldots, e^b)\) for the bundle \( B_e \). Furthermore, we add to \( G' \) vertices \( s \) and \( t \) and arcs \((s, t') \) and \((t, t') \) for every \( i \in [b] \). This finishes the description of the instance \( I' = (G', B, \omega', k, W) \).

It is straightforward to observe that if \( Z \) is a solution to the instance \( I \), then \( \bigcup_{e \in Z} B_e \) is a solution to \( I' \) of the same weight and touching \( |Z| \) bundles. In the other direction, note that if \( Z' \) is a solution to \( I' \) then \( Z = \{ e \in E(G) \mid Z' \cap B_e \neq \emptyset \} \) is a solution to \( I \).

We deduce the following.

**Theorem 3.6.** \textsc{Weighted DFAS} and \textsc{Weighted DFVS} are randomized FPT when parameterized by \( k \).

### 4 WEIGHTED ALMOST 2-SAT

The final application of the flow-augmentation framework is to the problem \textsc{Weighted Almost 2-SAT}. Let us recall the problem definition. The input consists of a 2-CNF formula \( \phi \), viewed as a set of 2-clauses; a weight function \( \omega : \phi \rightarrow \mathbb{Z}_+ \), and integers \( k, W \in \mathbb{Z}_+ \).

The objective is to find a clause deletion set \( Z \subseteq \phi \) such that \( \phi - Z \) is satisfiable, \( |Z| \leq k \), and \( \omega(Z) = \sum_{e \in Z} \omega(e) \leq W \). We refer to a clause set \( Z \subseteq \phi \) such that \( \phi - Z \) is satisfiable as a solution. Our goal is thus to find such a solution \( Z \), with \( |Z| \leq k \) and \( \omega(Z) \leq W \), in time FPT parameterized by \( k \). We show the following.

**Theorem 4.1.** \textsc{Weighted Almost 2-SAT} is randomized FPT parameterized by \( k \), with a running time of \( 2^{O(k)} n^{O(1)} \).

As in the \textsc{Almost 2-SAT}-algorithm of Kratsch and Wahlström [12], we solve the problem by reduction to the auxiliary problem \textsc{DiGraph Pair Cut}. Let us define the weighted version of this. An instance \( I = (G, s, t, \omega, T, k, W) \) of \textsc{Weighted DiGraph Pair Cut} consists of a digraph \( G \) with special vertices \( s \) and \( t \), a weight function \( \omega : E(G) \rightarrow \mathbb{Z}_+ \), and integers \( k, W \in \mathbb{Z}_+ \). Refer to a set of edges \( Z \subseteq E(G) \) as a solution if it is an st-cut and for every pair \( p \in T \), at most one endpoint of \( p \) is reachable from \( s \) in \( G - Z \). The goal is then to find a solution \( Z \subseteq E(G) \) such that \( |Z| \leq k \) and \( \omega(Z) \leq W \). Note that the addition of a sink vertex \( t \) differs from the original formulation of \textsc{DiGraph Pair Cut} [12]; we add such a vertex \( t \) in order to make use of the flow-augmentation technique. We show that \textsc{Weighted Almost 2-SAT} FPT-reduces to \textsc{Weighted DiGraph Pair Cut}.

**Lemma 4.2.** Let \( I = (\phi, \omega, k, W) \) be an instance of \textsc{Weighted Almost 2-SAT}. In time \( 2^{O(k)} \), we can produce a list of instances \( I' = (G, s, t, \omega', T', k', W') \) of \textsc{Weighted DiGraph Pair Cut} such that \( I \) is a yes-instance if and only if at least one instance \( I' \) is a yes-instance, where \( k' = O(k) \) for each produced instance.

Therefore, it suffices to provide an FPT-algorithm for \textsc{Weighted DiGraph Pair Cut}. Let \( I = (G, s, t, \omega, T, k, W) \) be an instance of \textsc{Weighted DiGraph Pair Cut} and let \( Z \) be a minimal solution to \( I \). Then \( Z \) must be a star st-cut; i.e., for every \((u, v) \in Z \), \( u \) is reachable from \( s \) in \( G - Z \) but \( v \) is not. Therefore, the tool of flow
augmentation for star $st$-cuts (Theorem 2.6) is applicable to the problem, and our algorithm is based around this method. We defer to the full version [9] for the proof.

REFERENCES


