Finding Small Dijoins in Transitive Closure Time

Chaitanya Nalam
University of Michigan, Ann Arbor, MI, USA

Thatchaphol Saranurak
University of Michigan, Ann Arbor, MI, USA

Abstract

We present a faster algorithm for finding a *minimum dijoin*, a smallest set of edges whose contraction makes a directed graph strongly connected. This problem has been studied since the 1960s [Seshu and Reed 1961] and is dual to finding a maximum sized family of disjoint *dicuts* [Lucchesi and Younger 1978].

Given a directed graph G with n vertices and m edges whose minimum dijoin has size d, our algorithm outputs both a minimum dijoin and a maximum sized family of disjoint dicuts in $O(\mathrm{TC} \cdot d)$ time, where $\mathrm{TC} = \min(mn, n^\omega)$ is the time to compute the transitive closure. This improves upon the state of the art of [Gabow 1993], which requires $O(\mathrm{TC} \cdot \min(m^{1/2}, n^{2/3}))$ time when $d = o(\min(m^{1/2}, n^{2/3}))$. Our result extends to finding a minimum *weighted* dijoin. We achieve this by observing that Frank's algorithm [Frank 1981] can be sped up when warm-started with a 2-approximation solution, which we observed can be computed in near-linear time.

2012 ACM Subject Classification Theory of computation → Network flows

Keywords and phrases Graph algorithms, Dijoin, Submodular flow

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.46

Funding *Thatchaphol Saranurak*: Supported by NSF Grant CCF-2238138.

1 Introduction

The primary goal of this work is to give an algorithm with an improved running time for the minimum (weighted) dijoin problem. We define the problem formally.

1.1 Problem definition

In a directed graph $G = (V, E)$ with n vertices and m edges, a *dijoin* is a subset of edges $B \subset E$ whose contraction makes the graph strongly connected. The *minimum dijoin* problem seeks a dijoin of minimum possible size.

The dual of the minimum dijoin problem is to find a *maximum disjoint dicut family*, defined as follows. For any vertex sets $S, T \subseteq V$, let $E(S, T)$ denote the set of edges $(u, v) \in E$ where $u \in S$ and $v \in T$. A *directed cut*, or simply *dicut*, is a subset of edges $D \subset E$ such that there exists a set $S \subset V$ for which $E(S, V \setminus S) = \emptyset$ and $E(V \setminus S, S) = D$. In the maximum disjoint dicut family problem, we want to find a maximum collection of dicuts with pairwise disjoint edge sets.

The weighted variants of these problems involve an edge weight function $w : E \mapsto \mathbb{Z}^+$. The minimum weighted dijoin requires finding a dijoin that minimizes the total weight of its edges. Its dual deals with finding a maximum-sized *w-independent dicut family*, a family of dicuts such that each edge e appears in at most $w(e)$ dicuts within the family.

1.2 Previous work and technical challenges

Enhancing connectivity in directed graphs under minimal edge additions/contractions is a fundamental theme in graph theory. A central problem in this domain is the minimum dijoin problem, first posed by John Runyon (as cited in [23]). Younger [25] and Robertson

(unpublished) independently conjectured that the minimum size of a dijoin equals the maximum size of a disjoint dicut family. This fascinating min-max statement was first proved for bipartite graphs by [21] and then proved for general graphs by Lucchesi and Younger in their seminal work [20]. This theorem is a canonical example of problems captured by the submodular flow framework [2], a vast generalization of both minimum-cost flow and matroid intersection. It has played a central role in the development of submodular flow algorithms [26, 22, 5, 8, 9, 10, 14, 3, 7, 18].

Below, we discuss the algorithmic aspect of these problems. All prior results extend naturally to weighted variants. Lucchesi and Younger [19], Karzanov [17], and Frank [4] independently gave the first polynomial-time algorithms for the minimum dijoin problem. Among them, the algorithm by Frank [4] runs in $O(mn^3)$ time. Frank's algorithm maintains a feasible dijoin and a disjoint dicut family and iteratively improves both until their sizes become equal, at which point both are optimal by the min-max theorem. He provided an $O(mn^2)$-time subroutine for improving the solution pair, and showed that the number of iterations is at most the size of the initial feasible dijoin. This quantity is bounded by $n-1$, as edges of any spanning tree form a feasible dijoin, leading to the $O(mn^3)$ time claimed above.

Let $\mathrm{TC} = O(\min(mn, n^\omega))$ denote the time to compute the transitive closure of a directed graph with n vertices and m edges, where ω is the matrix multiplication exponent. Gabow [13, 12] improved the running time of the iterative sub-routine to TC. He [14] further reduced the number of iterations to $O(\min(m^{1/2}, n^{2/3}))$ leading to the state-of-the-art running time of $O(\mathrm{TC} \cdot \min(m^{1/2}, n^{2/3}))$. The algorithm by Gabow [12] for the iterative sub-routine is applicable to a class of more general problems. Subsequent work by [24, 16] gave simpler $O(mn)$-time algorithms for the iterative sub-routine specifically tailored to the dijoin problem.

All of these approaches rely on computing *minimal tight sets* to improve the pair of solutions. Shepherd and Vetta [24] gave a reduction showing that the minimal tight set computation is as hard as computing the transitive closure. Hence, TC effectively becomes the best time bound one can hope for within this framework. This leads us to the following question.

Can we compute a minimum dijoin and a maximum disjoint dicut family in $O(\mathrm{TC})$ time?

Although the dijoin problem is a special case of submodular flow, recent developments in submodular flow algorithms have not yielded improved bounds. In particular, the algorithm by Lee, Sidford and Wang [18] applies only when the input submodular function is defined over a complete set family $\mathcal{F} = 2^V$, whereas the dijoin problem requires a function defined on a more general crossing family where, in particular, $\mathcal{F} \subsetneq 2^V$.

1.3 Our Contributions

Improved algorithm for unweighted dijoin. We give an algorithm with running time $O(\mathrm{TC} \cdot d)$ for solving the minimum dijoin problem, where d is the size of the minimum dijoin and maximum disjoint dicut family. Formally,

▶ **Theorem 1.** *There exists an algorithm that, given an unweighted directed graph $G = (V, E)$, computes a minimum dijoin and a maximum disjoint dicut family in $O(d \cdot \min(mn, n^\omega))$ time.*

Within the algorithmic framework that relies on computing minimal tight sets, our algorithm achieves the optimal $O(\mathrm{TC})$ time when $d = O(1)$. When $d = o(\min(m^{1/2}, n^{2/3}))$, our algorithm outperforms the fastest algorithm by Gabow [14].

We also extend this result to the weighted case, giving an algorithm to find a minimum weighted dijoin. Formally,

▶ **Corollary 2.** *There exists an algorithm that, given a weighted directed graph $G = (V, E, w)$ where $w : E \mapsto \mathbb{Z}^+$, computes a minimum weighted dijoin in $O(d_{uw} \cdot \min(mn, n^\omega))$ time, where d_{uw} represents the minimum cardinality of all feasible dijoins.*

1.4 Organization

Section 2 presents the two key ingredients needed to obtain an algorithm for the minimum weighted dijoin problem, and then Section 3 combines them, proving Theorem 1 and Corollary 2. We conclude this section by introducing the notation used throughout the rest of the paper.

1.5 Preliminaries

For a directed edge (u, v), we refer to u as the *tail* and v as the *head*. For any two subsets of vertices $S, T \subset V$, let $E(S, T)$ denote the set of directed edges whose head is in T and tail is in S. We write $\delta^-_G(S)$ to denote the set of incoming edges to S, i.e., $E(V \setminus S, S)$, and $\delta^+_G(S)$ for the outgoing edges from S, i.e., $E(S, V \setminus S)$ in the graph G. We omit the subscripts if it is clear from the context. We refer to a set $S \subset V$ with $\delta^+(S) = \emptyset$ and $\delta^-(S) \neq \emptyset$ as the *kernel* of the dicut $\delta^-(S)$.

We begin with some simple structural observations about the dijoin problem. Without loss of generality, we may assume the input graph is weakly connected, i.e., the underlying undirected graph obtained by ignoring edge directions is connected, since disconnected graphs have no feasible dijoin.

We may also assume that G is acyclic. If G contains a directed cycle C, then any vertex reachable to/from a vertex in C can reach or be reached from all vertices in C. Therefore, contracting C preserves reachability and does not affect the minimum dijoin. Repeating this process, we reduce G to a DAG without changing the dijoin size.

This also ensures that G does not contain both an edge and its reverse, as they form a directed cycle of length two.

A directed graph is strongly connected if and only if it contains no dicut. Since contracting an edge in a dicut removes it, a dijoin can be equivalently defined as a subset of edges intersecting every dicut in the graph.

We use G^R to denote the graph obtained by reversing all arcs of G. For a weighted graph $G = (V, E, w)$, $G^R = (V, E^R, w^R)$ where E^R is the set of reverse arcs of E and $w^R : E^R \mapsto \mathbb{Z}^+$ is defined by $w^R(a) = w(a^R)$. The following proposition formalizes the symmetry of the dijoin problem under graph reversal.

▶ **Proposition 3.** *Let $G = (V, E, w)$ be a weighted directed graph, and let $G^R = (V, E^R, w^R)$ be its reverse. Then a subset $B \subseteq E$ is a feasible dijoin in G if and only if $B^R \subseteq E^R$ is a feasible dijoin in G^R, and the total weights of the two solutions are equal.*

Proof. Since contraction and reversal commute, we have $(G^R/B^R)^R = G/B$ for any edge set $B \subseteq E$. Moreover, strong connectivity is preserved under reversal: G is strongly connected if and only if G^R is. Thus, B is a dijoin in G if and only if B^R is a dijoin in G^R. The weight equality follows from the definition of w^R. ◀

We formally define the min-cost s-arborescence problem for future reference.

▶ **Definition 4** (s-arborescence). *Given a graph $G = (V, E, w)$ and a vertex $s \in V$, an* s-arborescence *is a spanning tree of G rooted at s, with no incoming edges into s, where every vertex $v \neq s$ has exactly one incoming edge. The cost of the arborescence is defined as the total weight of all edges of the arborescence.*

▶ **Problem 5** (Min-cost s-arborescence). *Given a directed graph $G = (V, E, w)$ and $s \in V$, find an s-arborescence of minimum cost, if one exists.*

2 Technical overview

In this section, we state the two key ingredients needed for proving Theorem 1: Frank's primal-dual algorithm from [4, 12] for computing the optimal weighted dijoin, and a 2-approximation algorithm for the same problem from [6, 1].

First ingredient

We begin by recalling Frank's primal-dual framework for the minimum weighted dijoin problem, where the primal corresponds to the dijoin problem and the dual to the w-independent dicut family. The algorithm maintains a potential function $p : V \mapsto \mathbb{Z}^+$ as a proxy for the dual solution. (The construction of the corresponding dicut family is deferred.) Frank gives sufficient conditions on a dijoin-potential pair (B, p) that ensure the optimality of both solutions:

Optimality criteria:

$$\forall e = (u, v) \in B, \qquad p(u) - p(v) = w(uv) \tag{1}$$

$$\forall e = (u, v) \in E \setminus B, \qquad p(u) - p(v) \leq w(uv) \tag{2}$$

$$\forall u \in T_B(v), \qquad p(u) \leq p(v) \tag{3}$$

Here, $T_B(v)$ denotes the intersection of all kernels containing v whose corresponding dicuts intersect B in exactly one edge. These are referred to as *minimal tight sets* in Frank's work.

Let B_0 be any spanning tree and p_0 the zero function. Frank starts with (B_0, p_0) as the initial primal-dual pair. Note that (B_0, p_0) already satisfies Equations (2) and (3) and violates only Equation (1). He then describes a subroutine that updates the given (B, p) to a new pair (B', p') that still satisfies Equations (2) and (3) and a strictly smaller number of edges violate Equation (1). We define it below. Let violate(B, p) denote the set of edges that violate Equation (1) with respect to the solution pair (B, p).

▶ **Lemma 6** (Algorithm 4.1 from [4]). *There exists a subroutine, ImprovePair, which takes a directed graph $G = (V, E, w)$, a primal-dual pair (B, p) satisfying Equations (2) and (3), along with an edge $e \in B$ that currently violates Equation (1). It returns a new pair (B', p') such that:*

- *Equations (2) and (3) still hold for the pair (B', p'),*
- violate$(B', p') \subseteq$ violate$(B, p) \setminus \{e\}$

The subroutine runs in $O(n^2 + f(n))$ time, where $f(n)$ denotes the time required to compute $T_B(v)$ for all $v \in V$.

The proof of correctness for this lemma is given in Section 4 of [4], and the running-time analysis in Section 5. To compute $T_B(v)$ for all vertices in Lemma 6, Gabow [12] showed that this can be done in $O(\text{TC})$ time.

▶ **Lemma 7** (Theorem 12.1 from [12]). *Given $G = (V, E)$ and a dijoin B, the sets $T_B(v)$ for all $v \in V$ can be computed in $O(\min(mn, n^\omega))$ time.*

As discussed in the introduction, transitive closure can be reduced to computing minimal tight sets. Hence, this lemma achieves the best possible running time for this task.

Second ingredient

As noted in Lemma 6, the number of initially violated edges determines the number of ImprovePair iterations and hence the overall running time.

While [4, 14] use a spanning tree as the trivial starting solution, we observe that a better dijoin can be obtained in linear time whose size is at most twice that of the optimum. This yields Theorem 1 and Corollary 2.

Bang-Jensen et al. [1] give a 2-approximation algorithm for a generalization of the minimum dijoin problem. We note that their algorithm for the dijoin problem can be implemented in near-linear time and outline the details below.

For this purpose, we define the *rooted* variant of the dijoin problem as follows.

▶ **Problem 8** (Minimum s-Dijoin). *Given a directed graph $G = (V, E, w)$ and a vertex $s \in V$, an s-dijoin $B \subseteq E$ is a set of edges such that, after contracting them, s can reach all vertices. The minimum s-dijoin problem is to find an s-dijoin with minimum total weight.*

Frank [6] observes that a minimum s-dijoin can be computed in near-linear time by reducing to the min-cost arborescence problem. Note that adding a reverse edge is equivalent to contracting an edge in terms of preserving reachability. Hence an s-dijoin can be equivalently seen as the set of reverse edges which need to be added so that every vertex is reachable from s. Given these observations, the reduction is simple and proceeds as follows.

Given a graph $G = (V, E, w)$, construct the graph $G' = G^0 \cup G^R$, where G^0 is the same graph as G with edge weights set to zero; refer Section 1.5 for the definition of G^R. Compute a min-weight s-out-arborescence T. Observe that the set of reverse edges in T corresponds to an s-dijoin (with respect to the equivalent definition of adding reverse edges) in G and is the minimum-weight set with that property. As the minimum-weight s-arborescence can be found in $O(m + n \log n)$, we have the following theorem.

▶ **Theorem 9.** *There exists an algorithm that takes a weighted directed graph $G = (V, E, w)$ and a vertex $s \in V$ as input and finds a minimum s-dijoin in $O(m + n \log n)$ time.*

It is worthwhile to note that even for the s-dijoin problem, computing minimal tight sets is known to require at least transitive-closure time [24], presenting a fundamental bottleneck for any approach that relies on such computations. Theorem 9 circumvents this barrier by avoiding the computation of minimal tight sets entirely. An intriguing open question is whether a similar approach can be developed for the general minimum dijoin problem, potentially breaking the $O(\mathrm{TC})$-time barrier.

We observe that the dual of the s-dijoin – similarly defined as a maximum w-independent s-dicut family – can also be solved in $O(m + n \log n)$ time. This is discussed in detail in Appendix A, as it is not directly relevant to the current work.

Given Theorem 9, the key insight in [1] is that combining a minimum s-dijoin in G with the reverse of a minimum s-dijoin in G^R results in a 2-approximate dijoin of G. The algorithm is shown below. Let sDijoin denote the algorithm mentioned in Theorem 9.

Algorithm 1 APPROXDIJOIN.

Input : Graph $G = (V, E, w)$
Output : 2-approximate dijoin
Construct $G^R = (V, E^R, w^R)$;
Let s be any vertex in V;
$B_1 = \text{SDIJOIN}(G, s)$;
$B_2 = \text{SDIJOIN}(G^R, s)$;
return $B_1 \cup B_2^R$;

▶ **Theorem 10** ([1]). *There exists an algorithm that, given a weighted directed graph $G = (V, E, w)$, outputs a 2-approximate dijoin in $O(m + n \log n)$ time.*

We now state an immediate corollary of Theorem 10, needed in the next section.

▶ **Corollary 11.** *For any unweighted directed graph, a feasible dijoin of size at most $2d$ (where d is the size of an optimal dijoin) can be found in $O(m + n \log n)$ time.*

3 $O(mnd)$-Algorithm for Minimum Dijoin

This section combines the two ingredients from the technical overview to derive our algorithm and prove Theorem 1 and Corollary 2. Lemma 12 shows how IMPROVEPAIR yields a primal-dual pair (B^*, p^*) satisfying the optimality conditions, and Lemma 13 shows how to construct the corresponding optimal disjoint dicut family. Together, these results establish Theorem 1, which is followed by the proof of Corollary 2.

We will now prove Lemma 12.

▶ **Lemma 12.** *There exists an algorithm that, given an unweighted directed graph $G = (V, E)$, computes a primal-dual pair (B^*, p^*) satisfying Equations (1)–(3) in $O(d \min(mn, n^\omega))$ time.*

Proof. Let B_0 be a dijoin of size at most $2d$ computed using Corollary 11, and let $p_0 \equiv 0$ be the initial potential function. The pair (B_0, p_0) satisfies Equations (2) and (3), but may violate Equation (1) for up to $|B_0|$ many edges.

Each invocation of IMPROVEPAIR fixes at least one violation without introducing new ones, so at most $|B_0| \leq 2d$ invocations are needed to output a pair, (B^*, p^*), that satisfies the optimality criteria. By Lemma 6 and Lemma 7, each invocation is executed in $O(n^2 + \min(mn, n^\omega))$ time. Thus, the total time is

$$O(m + n \log n) + O(n^2 + \min(mn, n^\omega)) \cdot 2d = O(d \min(mn, n^\omega)).$$ ◀

We now describe how to construct a disjoint dicut family from the optimal solution pair (B^*, p^*) in the following lemma.

▶ **Lemma 13.** *There exists an algorithm that, given an unweighted directed graph $G = (V, E)$ and an optimal solution pair (B^*, p^*), computes a maximum disjoint dicut family in $O(n^2 d + \min(mn, n^\omega))$ time, where d is the number of dicuts in the optimal family.*

This lemma provides the second part of our promised output: a maximum disjoint dicut family. We present a modification to the algorithm by Frank (specifically, the version described at the end of Section 5 in [4]), altering the representation of the dicut family it outputs. This modification is necessary because Frank's original algorithm requires $O(n^4)$ time to construct the optimal dicut family from an optimal potential function, whereas we aim to compute the optimal dual solution within $O(d \cdot \min(mn, n^\omega))$ time.

If D denotes the edge set of a dicut in the family, we output the complement of the corresponding kernel. For any graph G, let $CC(G)$ denote the weakly connected component partition of the vertex set of graph G. The algorithm is as follows:

Algorithm 2 DisjointDicutFamily from [4].

Input : Graph $G = (V, E)$, optimal solution pair (B, p)
Output : Maximum disjoint dicut family

1 Compute $T_B(v)$ for all $v \in V$;
2 Let $p_1, \cdots, p_l$ be the distinct potential values in ascending order;
3 For $i \in [l]$, let $P_i = \{u \mid p(u) = p_i\}$;
4 Initialize $\mathcal{F} = \emptyset$;
5 **for** $i \leftarrow 1$ **to** $l - 1$ **do**
6 Define $V_i = \bigcup_{j=1}^{i} P_j$;
7 Define $G_i = (V_i, E_i)$ where $E_i = \{(x, y) \mid x, y \in V_i, y \in T_B(x)\}$;
8 Let $\mathcal{C} = CC(G_i)$;
9 **for** $C \in \mathcal{C}$ **do**
10 $\mathcal{F} = \mathcal{F} \cup CC(G[V \setminus C])$ where $G[S]$ denotes the induced subgraph on $S \subseteq V$;
11 **return** $\mathcal{F}$;

We will now prove Lemma 13.

Proof of Lemma 13. Given the unweighted directed graph $G = (V, E)$ and the pair of optimal primal dual solutions (B^*, p^*) as inputs to Algorithm 2. It outputs a disjoint dicut family in which each edge occurs in at most one dicut which follows from the optimality condition Equation (1) as explained in [4]. The optimality of this family follows as the size of the family is equal to the cost of the optimal dijoin, as proved in [4].

We now analyze the running time of the algorithm when the graph is unweighted. It takes $O(\min(mn, n^\omega))$ time to compute $T_B(v)$ for all $v \in V$ in line 1 using Lemma 7. Lines 2 and 3 take $O(n \log n)$ to sort and compute the partitions P_i. Defining G_i in line 7 and computing $CC(G_i)$ take at most $O(n^2)$ as there can be $O(n^2)$ edges in the worst case. Note that each iteration of the outer loop adds at least one dicut to $\mathcal{F}$. As the size of the optimal disjoint dicut family is d, $l \leq d + 1$. Hence, it takes $O(n^2 d)$ time overall for line 7 over the course of the algorithm.

For any set $S \subset V$, $G[V \setminus S]$ contains at most m edges. Thus, it takes $O(m)$ time to compute $CC(G[V \setminus C])$ for each component C of G_i. Since each induced subgraph has at least one component, it adds at least one dicut to $\mathcal{F}$. As there can be at most d such computations in total, it takes $O(md)$ time in total during the algorithm for line 10.

Thus, the overall running time is $O(\min(mn, n^\omega) + n \log n + n^2 d + md) = O(n^2 d + \min(mn, n^\omega))$. ◀

We now present the proof of the main result.

Proof of Theorem 1. Given an unweighted directed graph $G = (V, E)$, we first compute the pair of optimal primal-dual solutions (B^*, p^*) using the algorithm from Lemma 12, where B^* is the optimal dijoin. Then, we apply the algorithm from Lemma 13 with inputs G and (B^*, p^*) to obtain the optimal disjoint dicut family.

Computing the optimal primal-dual pair takes $O(d \cdot \min(mn, n^\omega))$ time, and constructing the dicut family takes $O(n^2 d + \min(mn, n^\omega))$ time. The total time remains $O(d \cdot \min(mn, n^\omega))$. ◀

We will now prove Corollary 2 using Theorem 1.

Proof of Corollary 2. Given a weighted graph $G = (V, E, w)$ with $w : E \mapsto \mathbb{Z}^+$, consider the underlying unweighted graph G_{uw} and let d, d_{uw} be the cardinalities of the optimal weighted dijoin and the optimal unweighted dijoin, respectively. We know that $d_{uw} \leq d$ as any weighted dijoin ignoring weights is a feasible dijoin with the same cardinality.

Let B_{uw} be the feasible dijoin obtained by giving G_{uw} as input to Corollary 11 and it guarantees that $|B_{uw}| \leq 2d_{uw}$.

Hence, starting with the solution pair (B_{uw}, p) where $p \equiv 0$ as the potential function, at most $2d$ edges violate Equation (1). From the guarantees of Lemma 6, together with Lemma 7, we know that each iteration takes $O(\min(mn, n^\omega))$ time and reduces one violated edge, so the final pair (B^*, p^*) after at most $|B_{uw}| \leq 2d_{uw}$ iterations, satisfies the optimality criteria which implies optimality; in particular, B^* is an optimal weighted dijoin. The total time taken is at most $O(d_{uw} \cdot \min(mn, n^\omega))$. ◀

References

1 Jørgen Bang-Jensen, Florian Hörsch, and Matthias Kriesell. Complexity of (arc)-connectivity problems involving arc-reversals or deorientations. *Theor. Comput. Sci.*, 973:114097, 2023. doi:10.1016/j.tcs.2023.114097.

2 Jack Edmonds and Rick Giles. A min-max relation for submodular functions on graphs. In P.L. Hammer, E.L. Johnson, B.H. Korte, and G.L. Nemhauser, editors, *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 185–204. Elsevier, 1977. doi:10.1016/S0167-5060(08)70734-9.

3 Lisa Fleischer and Satoru Iwata. Improved algorithms for submodular function minimization and submodular flow. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 107–116. ACM, 2000. doi:10.1145/335305.335318.

4 András Frank. How to make a digraph strongly connected. *Comb.*, 1(2):145–153, 1981. doi:10.1007/BF02579270.

5 András Frank. Finding feasible vectors of edmonds-giles polyhedra. *Journal of Combinatorial Theory, Series B*, 36(3):221–239, 1984. doi:10.1016/0095-8956(84)90029-7.

6 András Frank. Connections in combinatorial optimization. *Discrete Appl. Math.*, page 1875, 2012. doi:10.1016/j.dam.2011.09.003.

7 András Frank and Zoltán Miklós. Simple push-relabel algorithms for matroids and submodular flows. *Japan Journal of Industrial and Applied Mathematics*, 29(3):419–439, 2012. doi:10.1007/s13160-012-0076-y.

8 András Frank and Éva Tardos. Generalized polymatroids and submodular flows. *Mathematical Programming*, 42(1):489–563, 1988. doi:10.1007/BF01589418.

9 András Frank and Éva Tardos. An application of submodular flows. *Linear Algebra and its Applications*, 114-115:329–348, 1989. doi:10.1016/0024-3795(89)90469-2.

10 Satoru Fujishige. Submodular functions and optimization. In *Submodular Functions and Optimization*, volume 58 of *Annals of Discrete Mathematics*, page 1. Elsevier, 1990.

11 D. R. Fulkerson. Packing rooted directed cuts in a weighted directed graph. *Math. Program.*, 6(1), 1974. doi:10.1007/BF01580218.

12 H. N. Gabow. Centroids, representations, and submodular flows. *Journal of Algorithms*, 18(3):586–628, 1995. doi:10.1006/jagm.1995.1022.

13 Harold N. Gabow. A representation for crossing set families with applications to submodular flow problems. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 202–211. ACM/SIAM, 1993. URL: http://dl.acm.org/citation.cfm?id=313559.313753.

14 H.N. Gabow. A framework for cost-scaling algorithms for submodular flow problems. In *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, pages 449–458, 1993. doi:10.1109/SFCS.1993.366842.

15 Anupam Gupta. Lecture 2: Arborescences, 2020. Lecture Notes, CMU Course 15-850: Advanced Algorithms.

16 Satoru Iwata and Yusuke Kobayashi. An algorithm for minimum cost arc-connectivity orientations. *Algorithmica*, 56(4):437–447, 2010. doi:10.1007/s00453-008-9179-x.

17 A. V. Karzanov. On the minimal number of arcs of a digraph meeting all its directed cutsets. *Graph Theory Newsletter*, 8(4), 1979. Abstract.

18 Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. *CoRR*, abs/1508.04874, 2015. arXiv:1508.04874.

19 C. L. Lucchesi. *A Minimax Equality for Directed Graphs*. Ph.d. thesis, University of Waterloo, 1976.

20 C. L. Lucchesi and D. H. Younger. A minimax theorem for directed graphs. *Journal of the London Mathematical Society*, s2-17(3):369–374, 1978. doi:10.1112/jlms/s2-17.3.369.

21 I. P. McWhirter and D. H. Younger. Strong covering of a bipartite graph. *Journal of the London Mathematical Society*, s2-3(1):86–90, 1971. doi:10.1112/jlms/s2-3.1.86.

22 A. Schrijver. *Min-Max Results in Combinatorial Optimization*, pages 439–500. Springer Berlin Heidelberg, 1983. doi:10.1007/978-3-642-68874-4_18.

23 S. Seshu and M.B. Reed. *Linear Graphs and Electrical Networks*. Addison-Wesley series in behavioral science: Quantitative methods. Addison-Wesley Publishing Company, 1961. URL: https://books.google.com/books?id=mMk-AAAAIAAJ.

24 F.B. Shepherd and A. Vetta. Visualizing, finding and packing dijoins. In David Avis, Alain Hertz, and Odile Marcotte, editors, *Graph Theory and Combinatorial Optimization*, pages 219–254. Springer-Verlag, 2005. doi:10.1007/0-387-25592-3_8.

25 D. H. Younger. *Feedback in a Directed Graph*. Ph.d. thesis, Columbia University, 1963.

26 U. Zimmermann. Minimization on submodular flows. *Discrete Applied Mathematics*, 1982. doi:10.1016/0166-218X(82)90053-1.

A Optimal dual of s-dijoin

In this section, we show how to solve the dual problem of s-dijoin. We begin with some definitions.

▶ **Definition 14** (s-dicut). *A dicut D of the graph G is an* s-dicut *for some vertex $s \in V$ if the kernel corresponding to the dicut contains s.*

The dual problem follows.

▶ **Problem 15** (Maximum w-independent s-dicut family). *Given a directed graph $G = (V, E, w)$ and a vertex $s \in V$, find a family of s-dicuts with maximum possible size such that, each edge $e \in E$ appears in at most $w(e)$ s-dicuts within the family.*

We give an algorithm that solves this problem in $O(m + n \log n)$ time.

▶ **Theorem 16** (Informal). *There exists an algorithm that computes optimal w-independent s-dicut family in $O(m + n \log n)$ time.*

There is a slight catch in the informal theorem above. The total size of the optimal s-dicut family can be as large as $\Omega(n^2)$ and since we only have $O(m + n \log n)$ time, we have to be focusing on how we represent the output.

To represent a dicut, we can either list its edges or describe the vertex set defining the cut. In this case, we use the vertex-set representation. One can show that the complement of the kernels corresponding to the optimal family of s-dicuts forms a *laminar family.* A family of vertex sets is laminar if any two sets in the family are either disjoint or one contains the other. So, we can represent them using a tree and we output this tree representation of an optimal w-independent s-dicut family in $O(m + n \log n)$ time.

To achieve this we follow a similar strategy as before reducing Problem 15 to the dual of the min-cost s-arborescence problem. The key technical claim is that the algorithm for the dual of the min-cost s-arborescence problem, when run on the graph $G^0 \cup G^R$, produces a w-independent s-dicut family.

We begin by introducing the necessary terminology and then present the dual of the minimum cost s-arborescence.

A *cut* in a directed graph is defined as the set of edges C for which there exists a set $S \subset V$ such that $\delta^+(S) = C$. An *s-cut* is a cut where the corresponding vertex set S contains s. Note that the definitions of s-cut and s-dicut do not correlate in the directionality.

The dual of the min-cost s-arborescence is stated as follows:

▶ **Problem 17** (Maximum w-independent s-cut family)**.** *Given a directed graph $G = (V, E, w)$ and a vertex $s \in V$, find a family of s-cuts of maximum possible size such that each edge is contained in at most $w(e)$ many s-cuts in the family.*

The minimax result relating Problem 5 and Problem 17 is due to Fulkerson, who also provided the first algorithm for constructing an optimal dual solution to the minimum-cost arborescence problem.

▶ **Theorem 18** (Fulkerson branching theorem [11])**.** *The value of the minimum-cost s-arborescence equals the maximum possible size of a w-independent s-cut family.*

We refer to the lecture notes by Gupta [15], who presents an exposition showing how the algorithm for the minimum-cost arborescence implicitly solves the dual problem. He also provides a procedure to recover the dual solution by constructing the corresponding tree representation, thereby effectively proving the following theorem.

▶ **Theorem 19.** *There exists an algorithm that, given a directed graph $G = (V, E, w)$ and $s \in V$, computes a tree representation, $T_{\mathcal{F}}$, of the maximum-sized w-independent s-dicut family $\mathcal{F}$ in $O(m + n \log n)$ time.*

We use Theorem 19 to find maximum w-independent s-dicut family and prove the following theorem.

▶ **Theorem 20.** *There exists an algorithm that takes a directed graph $G = (V, E, w)$, $s \in V$ and outputs the tree representation $T_{\mathcal{F}}$ of the optimal w-independent s-dicut family $\mathcal{F}$ in $O(m + n \log n)$ time.*

Proof. Given a directed graph $G = (V, E, w)$, construct the graph $G' = G^0 \cup G^R = (V, A', w')$ which has $O(m)$ edges and n vertices. Give G' and $s \in V$ as inputs to the algorithm from Theorem 19 which outputs a tree representation $T_{\mathcal{F}}$ of the optimal w'-independent s-cut family $\mathcal{F}$ in $O(m + n \log n)$ time.

We prove that $\mathcal{F}$ is also an optimal w-independent s-dicut family by first proving that every s-cut $S \in \mathcal{F}$ in G' is a s-dicut in the original graph G, proving feasibility, and that the size of $\mathcal{F}$ equals the cost of an optimal s-dijoin, proving optimality.

Consider any $S \in \mathcal{F}$. Since S is an s-cut in G', the set $\delta^+_{G'}(S)$ cannot contain zero-weight edges e (from G^0) because these edges have weight $w(e) = 0$ and hence cannot appear in any cut of the w'-independent family. Thus, all edges in the s-cut belong to G^R, so

$$\delta^+_{G'}(S) = \delta^+_{G^R}(S) = \delta^-_G(S).$$

Similarly, $\delta^-_{G'}(S)$ cannot contain any edges of G^R because that would force $\delta^+_{G'}(S)$ to include a zero-weight edge, which is impossible. Hence,

$$\delta^+_G(S) = \delta^-_{G^R}(S) = \emptyset.$$

Therefore, S is a s-dicut in G. As each edge e in the s-cut is from G^R and occurs at most $w(e^R)$ times, the corresponding edge e^R in G occurs in at most $w(e^R)$ many s-dicuts making $\mathcal{F}$ a w-independent s-dicut family.

From Theorem 18, the size of the maximum w-independent s-cut family equals the cost of the minimum cost s-arborescence on G'. The former quantity equals the size of a w-independent s-dicut family as shown above and the latter equals the cost of an optimal s-dijoin by Theorem 9 proving optimality. ◀