On the Power of Tree-Depth for Fully Polynomial FPT Algorithms

Yoichi Iwata
National Institute of Informatics, Tokyo, Japan
yiwata@nii.ac.jp

Tomoaki Ogasawara
The University of Tokyo, Tokyo, Japan
t.ogasawara@is.s.u-tokyo.ac.jp

Naoto Ohsaka
The University of Tokyo, Tokyo, Japan
ohsaka@is.s.u-tokyo.ac.jp

Abstract

There are many classical problems in P whose time complexities have not been improved over the past decades. Recent studies of "Hardness in P" have revealed that, for several of such problems, the current fastest algorithm is the best possible under some complexity assumptions. To bypass this difficulty, the concept of "FPT inside P" has been introduced. For a problem with the current best time complexity $O(n^c)$, the goal is to design an algorithm running in $k^{O(1)}n^{c'}$ time for a parameter k and a constant $c' < c$.

In this paper, we investigate the complexity of graph problems in P parameterized by *tree-depth*, a graph parameter related to tree-width. We show that a simple divide-and-conquer method can solve many graph problems, including Weighted Matching, Negative Cycle Detection, Minimum Weight Cycle, Replacement Paths, and 2-hop Cover, in $O(\mathrm{td}\cdot m)$ time or $O(\mathrm{td}\cdot(m + n\log n))$ time, where td is the tree-depth of the input graph. Because any graph of tree-width tw has tree-depth at most $(\mathrm{tw}+1)\log_2 n$, our algorithms also run in $O(\mathrm{tw}\cdot m\log n)$ time or $O(\mathrm{tw}\cdot(m + n\log n)\log n)$ time. These results match or improve the previous best algorithms parameterized by tree-width. Especially, we solve an open problem of fully polynomial FPT algorithm for Weighted Matching parameterized by tree-width posed by Fomin et al. (SODA 2017).

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Fully Polynomial FPT Algorithm, Tree-Depth, Divide-and-Conquer

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.41

Funding Yoichi Iwata was supported by JSPS KAKENHI Grant Number JP17K12643. Naoto Ohsaka was supported by JSPS KAKENHI Grant Number JP16J09440.

Acknowledgements We would like to thank Hiroshi Imai for pointing out to us the reference [21].

1 Introduction

There are many classical problems in P whose time complexities have not been improved over the past decades. For some of such problems, recent studies of "Hardness in P" have provided evidence of why obtaining faster algorithms is difficult. For instance, Vassilevska Williams and Williams [33] and Abboud, Grandoni and Vassilevska Williams [1] showed that many problems including Minimum Weight Cycle, Replacement Paths, and Radius

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 41; pp. 41:1–41:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

are equivalent to All Pair Shortest Paths (APSP) under subcubic reductions; that is, if one of them admits a subcubic-time algorithm, then all of them do.

One of the approaches to bypass this difficulty is to analyze the running time by introducing another measure, called a *parameter*, in addition to the input size. In the theory of parameterized complexity, a problem with a parameter k is called *fixed parameter tractable (FPT)* if it can be solved in $f(k) \cdot |I|^{O(1)}$ time for some function $f(k)$ that does not depend on the input size $|I|$. While the main aim of this theory is to provide fine-grained analysis of NP-hard problems, it is also useful for problems in P. For instance, a simple dynamic programming can solve Maximum Matching in $O(3^{\mathrm{tw}}m)$ time, where m is the number of edges and tw is a famous graph parameter called *tree-width* which intuitively measures how much a graph looks like a tree (see Section 2 for the definition). Therefore, it runs in linear time for any graph of constant tree-width, which is faster than the current best $O(\sqrt{n}m)$ time for the general case [5, 31, 15].

When working on NP-hard problems, we can only expect superpolynomial (or usually exponential) function $f(k)$ in the running time of FPT algorithms. On the other hand, for problems in P, $k^{O(1)}|I|^{O(1)}$-time FPT algorithms might be possible. Such algorithms are called *(fully) polynomial FPT algorithms*, introduced by Giannopoulou, Mertzios and Niedermeier [16]. For instance, Fomin, Lokshtanov, Pilipczuk, Saurabh and Wrochna [11] obtained an $O(\mathrm{tw}^4 \cdot n \log^2 n)$-time (randomized) algorithm for Maximum Matching. In contrast to the $O(3^{\mathrm{tw}}m)$-time dynamic programming, this algorithm is faster than the current best general-case algorithm already for graphs of $\mathrm{tw} = O(n^{\frac{1}{8}-\epsilon})$. In general, for a problem with the current best time complexity $O(n^c)$, the goal is to design an algorithm running in $O(k^d n^{c'})$ time for some small constants d and $c' < c$. Such an algorithm is faster than the current best general-case algorithm already for inputs of $k = O(n^{(c-c')/d-\epsilon})$. On the negative side, Abboud, Vassilevska Williams and Wang [2] showed that Diameter and Radius do not admit $2^{o(\mathrm{tw})}n^{2-\epsilon}$-time algorithms under some plausible assumptions. In this paper, we give new or improved fully polynomial FPT algorithms for several classical graph problems. Especially, we solve an open problem for Weighted Matching posed by Fomin et al. [11].

Our approach. Before describing our results, we first give a short review of existing work on fully polynomial FPT algorithms parameterized by tree-width and explain our approach. There are roughly three types of approaches in the literature. The first approach is to use a polynomial-time dynamic programming on a tree-decomposition, which has been mainly used for problems related to shortest paths [7, 27, 4, 32]. The second approach is to use an $O(\mathrm{tw}^3 \cdot n)$-time Gaussian elimination of matrices of small tree-width developed by Fomin et al. [11]. The above-mentioned $O(\mathrm{tw}^4 \cdot n \log^2 n)$-time algorithm for Maximum Matching was obtained by this approach. The third approach is to apply a divide-and-conquer method exploiting the existence of small *balanced separators*. This approach was first used for planar graphs by Lipton and Tarjan [21]. Using the existence of $O(\sqrt{n})$-size balanced separators, they obtained an $O(n^{1.5})$-time algorithm for Maximum Matching and an $O(n^{1.5} \log n)$-time algorithm for Weighted Matching for planar graphs. For graphs of bounded tree-width, Akiba, Iwata and Yoshida [3] obtained an $O(\mathrm{tw} \cdot (m + n \log n) \log n)$-time algorithm for 2-hop Cover, which is a problem of constructing a distance oracle, and Fomin et al. [11] obtained an $O(\mathrm{tw} \cdot m \log n)$-time[1] algorithm for Vertex-disjoint $s-t$

[1] While the running time shown in [11] is $O(\mathrm{tw}^2 \cdot n \log n)$, we can easily see that it also runs in $O(\mathrm{tw} \cdot m \log n)$ time. Because $m = O(\mathrm{tw} \cdot n)$ holds for any graphs of tree-width tw, the latter is never worse than the former. Note that $\mathrm{tw} \cdot n$ in the running time of other algorithms cannot be replaced by m in general;

Table 1 Comparison of previous results and our results. n and m denote the number of vertices and edges, w denotes the width of the given tree-decomposition, and d denotes the depth of the given elimination forest. The factor d in our results can be replaced by $w \cdot \log n$.

Problem	Previous result	Our result
Maximum Matching	$O(w^4 n \log^2 n)$ [11]	$O(dm)$
Weighted Matching	Open problem [11]	$O(d(m + n \log n))$
Negative Cycle Detection	$O(w^2 n)$ [27]	$O(d(m + n \log n))$
Minimum Weight Cycle	—	$O(d(m + n \log n))$
Replacement Paths	—	$O(d(m + n \log n))$
2-hop Cover	$O(w(m + n \log n) \log n)$ [3]	$O(d(m + n \log n))$

Paths. We obtain fully polynomial FPT algorithms for a wide range of problems by using this approach. Our key observation is that, when using the divide-and-conquer approach, another graph parameter called *tree-depth* is more powerful than the tree-width.

A graph G of tree-width tw admits a set S of $\mathrm{tw} + 1$ vertices, called a balanced separator, such that each connected component of $G - S$ contains at most $\frac{n}{2}$ vertices. In both of the above-mentioned divide-and-conquer algorithms for graphs of bounded tree-width, after the algorithm recursively computes a solution for each connected component of $G - S$, it constructs a solution for G in $O(\mathrm{tw} \cdot (m + n \log n))$ time or $O(\mathrm{tw} \cdot m)$ time, respectively. Because the depth of the recursive calls is bounded by $O(\log n)$, the total running time becomes $O(\mathrm{tw} \cdot (m + n \log n) \log n)$ or $O(\mathrm{tw} \cdot m \log n)$, respectively.

Here, we observe that, by using tree-depth, this kind of divide-and-conquer algorithm can be simplified and the analysis can be improved. Tree-depth is a graph parameter which has been studied under various names [29, 20, 6, 25]. A graph has tree-depth td if and only if there exists an *elimination forest* of depth td. See Section 2 for the precise definition of the tree-depth and the elimination forest. An important property of tree-depth is that any connected graph G of tree-depth td can be divided into connected components of tree-depth at most $\mathrm{td} - 1$ by removing a single vertex r. Therefore, if there exists an $O(m)$-time or $O(m + n \log n)$-time *incremental algorithm*, which constructs a solution for G from a solution for $G - r$, we can solve the problem in $O(\mathrm{td} \cdot m)$ time or $O(\mathrm{td} \cdot (m + n \log n))$ time, respectively. Now, the only thing to do is to develop such an incremental algorithm for each problem. We present a detailed discussion of this framework in Section 3. Because any graph of tree-width tw has tree-depth at most $(\mathrm{tw} + 1) \log_2 n$ [24], the running time can also be bounded by $O(\mathrm{tw} \cdot m \log n)$ or $O(\mathrm{tw} \cdot (m + n \log n) \log n)$. Therefore, our analysis using tree-depth is never worse than the existing results directly using tree-width. On the other hand, there are infinitely many graphs whose tree-depth has asymptotically the same bound as tree-width. For instance, if every N-vertex subgraph admits a balanced separator of size $O(N^\alpha)$ for some constant $\alpha > 0$ (e.g., $\alpha = \frac{1}{2}$ for H-minor free graphs), both tree-width and tree-depth are $O(n^\alpha)$. Hence, for such graphs, the time complexity using tree-depth is truly better than that using tree-width.

Our results. Table 1 shows our results and the comparison to the existing results on fully polynomial FPT algorithms parameterized by tree-width. The formal definition of each problem is given in Section 4. Because obtaining an elimination forest of the lowest depth

e.g., we cannot bound the running time of the Gaussian elimination by $O(\mathrm{tw}^2 \cdot m)$, where m is the number of non-zero elements.

is NP-hard, we assume that an elimination forest is given as an input and the parameter for our results is the depth d of the given elimination forest. Similarly, for the existing results, the parameter is the width w of the given tree-decomposition. Note that, because a tree-decomposition of width w can be converted into an elimination forest of depth $O(w \cdot \log n)$ in linear time [29], we can always replace the factor d in our running time by $w \cdot \log n$. This also means that we can use arbitrary approximation algorithms or heuristics for constructing tree-decompositions for obtaining an elimination forest.

The first polynomial-time algorithms for Maximum Matching and Weighted Matching were obtained by Edmonds [10], and the current fastest algorithms run in $O(\sqrt{n}m)$ time [5, 31, 15] and $O(n(m + n \log n))$ time [5], respectively. Fomin et al. [11] obtained the $O(w^4 n \log^2 n)$-time randomized algorithm for Maximum Matching by using an algebraic method and the fast computation of Gaussian elimination. They left as an open problem whether a similar running time is possible for Weighted Matching. The general-case algorithms for these problems compute a maximum matching by iteratively finding an *augmenting path*, and therefore, they are already incremental. Thus, we can easily obtain an $O(dm)$-time algorithm for Maximum Matching and an $O(d(m + n \log n))$-time algorithm for Weighted Matching. Note that the divide-and-conquer algorithms for planar matching by Lipton and Tarjan [21] also use this augmenting-path approach, and our result can be seen as extension to bounded tree-depth graphs. Our algorithm for Maximum Matching is always faster than the one by Fomin et al. because we have $m = O(kn)$ for any graph of tree-width or tree-depth k and is faster than the general-case algorithm already when $d = O(n^{\frac{1}{2}-\epsilon})$. Our algorithm for Weighted Matching settles the open problem and is faster than the general-case algorithm already when $d = O(n^{1-\epsilon})$.

The current fastest algorithm for Negative Cycle Detection is the classical $O(nm)$-time Bellman-Ford algorithm. Planken et al. [27] obtained an $O(w^2 n)$-time algorithm by using a Floyd-Warshall-like dynamic programming. In this paper, we give an $O(d(m+n \log n))$-time algorithm. While the algorithm by Planken et al. is faster than the general-case algorithm only when $w = O(m^{\frac{1}{2}-\epsilon})$, our algorithm achieves a faster running time already when $d = O(n^{1-\epsilon})$.

Both Minimum Weight Cycle (or Girth) and Replacement Paths are subcubic-equivalent to APSP [33]. A naive algorithm can solve both problems in $O(n^3)$ time or $O(n(m+ n \log n))$ time. For Minimum Weight Cycle of directed graphs, an improved $O(nm)$-time algorithm was recently obtained by Orlin and Sedeño-Noda [26]. For Replacement Paths, Malik et al. [22] obtained an $O(m+n \log n)$-time algorithm for undirected graphs, and Roditty and Zwick [28] obtained an $O(\sqrt{n}m \cdot \text{polylog}\, n)$-time algorithm for unweighted graphs. For the general case, Gotthilf and Lewenstein [17] obtained an $O(n(m + n \log \log n))$-time algorithm, and there exists an $\Omega(\sqrt{n}m)$-time lower bound in the path-comparison model [19] (whenever $m = O(n\sqrt{n})$) [18]. In this paper, we give an $O(d(m + n \log n))$-time algorithm for each of these problems, which is faster than the general-case algorithm already when $d = O(n^{1-\epsilon})$. This result shows the following contrast to the known result of "Hardness in P": Radius is also subcubic-equivalent to APSP [1] but it cannot be solved in a similar running time under some plausible assumptions [2].

2-hop cover [8] is a data structure for answering distance queries in an efficient manner. Akiba et al. [3] obtained an $O(w(m + n \log n) \log n)$-time algorithm for constructing a 2-hop cover answering each distance query in $O(w \log n)$ time. In this paper, we give an $O(d(m + n \log n))$-time algorithm for constructing a 2-hop cover answering each distance query in $O(d)$ time.

Related work. Coudert, Ducoffe and Popa [9] have developed fully polynomial FPT algorithms using several other graph parameters including clique-width. In contrast to the tree-depth, their parameters are not polynomially bounded by tree-width, and therefore, their results do not imply fully polynomial FPT algorithms parameterized by tree-width. Mertzios, Nichterlein and Niedermeier [23] have obtained an $O(m + k^{1.5})$-time algorithm for Maximum Matching parameterized by feedback edge number k ($= m - n + 1$ when the graph is connected) by giving a linear-time kernel.

2 Preliminaries

Let $G = (V, E)$ be a directed or undirected graph, where V is a set of vertices of G and E is a set of edges of G. When the graph is clear from the context, we use n to denote the number of vertices and m to denote the number of edges. All the graphs in this paper are simple (i.e., they have no multiple edges nor self-loops). Let $S \subseteq V$ be a subset of vertices. We denote by $E[S]$ the set of edges whose endpoints are both in S and denote by $G[S]$ the subgraph induced by S (i.e., $G[S] = (S, E[S])$).

A *tree decomposition* of a graph $G = (V, E)$ is a pair (T, B) of a tree $T = (X, F)$ and a collection of *bags* $\{B_x \subseteq V \mid x \in X\}$ satisfying the following two conditions.

- For each edge $uv \in E$, there exists some $x \in X$ such that $\{u, v\} \subseteq B_x$.
- For each vertex $v \in V$, the set $\{x \in X \mid v \in B_x\}$ induces a connected subtree in T.

The *width* of (T, B) is the maximum of $|B_x| - 1$ and the *tree-width* $\mathrm{tw}(G)$ of G is the minimum width among all possible tree decompositions.

An *elimination forest* T of a graph $G = (V, E)$ is a rooted forest on the same vertex set V such that, for every edge $uv \in E$, one of u and v is an ancestor of the other. The *depth* of T is the maximum number of vertices on a path from a root to a leaf in T. The *tree-depth* $\mathrm{td}(G)$ of a graph G is the minimum depth among all possible elimination forests. Tree-width and tree-depth are strongly related as the following lemma shows.

▶ **Lemma 1** ([24, 29]). *For any graph G, the following holds.*

$$\mathrm{tw}(G) + 1 \leq \mathrm{td}(G) \leq (\mathrm{tw}(G) + 1) \log_2 n.$$

Moreover, given a tree decomposition of width k, we can construct an elimination forest of depth $O(k \log n)$ in linear time.

3 Divide-and-conquer framework

In this section, we propose a divide-and-conquer framework that can be applicable to a wide range of problems parameterized by tree-depth.

▶ **Theorem 2.** *Let $G = (V, E)$ be a graph and let f be a function defined on subsets of V. Suppose that $f(\emptyset)$ can be computed in constant time and we have the following two algorithms* Increment *and* Union *with time complexity $T(n, m)(= \Omega(n + m))$.*

- Increment$(X, f(X), x) \mapsto f(X \cup \{x\})$. *Given a set $X \subseteq V$, its value $f(X)$, and a vertex $x \notin X$, this algorithm computes the value $f(X \cup \{x\})$ in $T(|X \cup \{x\}|, |E[X \cup \{x\}]|)$ time.*
- Union$((X_1, f(X_1)), \ldots, (X_c, f(X_c))) \mapsto f(\bigcup_i X_i)$. *Given disjoint sets $X_1, \ldots, X_c \subseteq V$ such that G has no edges between X_i and X_j for any $i \neq j$, and their values $f(X_1), \ldots, f(X_c)$, this algorithm computes the value $f(\bigcup_i X_i)$ in $T(|\bigcup_i X_i|, |E[\bigcup_i X]|)$ time.*

Then, for a given elimination forest of G of depth k, we can compute the value $f(V)$ in $O(k \cdot T(n, m))$ time.

Algorithm 1 Algorithm for computing $f(V)$.

```
1: procedure Compute(S, T_S) ↦ f(S)        ▷ T_S is an elimination forest of G[S].
2:     if S = ∅ then return f(∅)
3:     T_1, ..., T_c ← the connected trees of T_S
4:     X_1, ..., X_c ← the sets of vertices of T_1, ..., T_c
5:     for i ∈ {1, ..., c} do
6:         x_i ← the root of T_i
7:         f_i ← Increment(X_i \ {x_i}, Compute(X_i \ {x_i}, T_i − x_i), x_i)
8:     return Union((X_1, f_1), ..., (X_c, f_c))
```

Proof. Algorithm 1 describes our divide-and-conquer algorithm. We prove that for any set S and any elimination forest T_S of $G[S]$ of depth k_S, $\textsc{Compute}(S, T_S)$ correctly computes the value $f(S)$ in $(2k_S + 1) \cdot T(|S|, |E[S]|)$ time by induction on the size of S.

The claim trivially holds when $S = \emptyset$. For a set $S \neq \emptyset$, let $T_1, \ldots, T_c$ be the connected trees of T_S ($c = 1$ if T_S is connected). For each i, let X_i be the set of vertices of T_i. From the definition of the elimination forest, G has no edges between X_i and X_j for any $i \neq j$. For each i, we compute the value $f(X_i)$ as follows. Let x_i be the root of T_i. By removing x_i from T_i, we obtain an elimination forest of $G[X_i \setminus \{x_i\}]$ of depth at most $k_S - 1$. Therefore, by the induction hypothesis, we can correctly compute the value $f(X_i \setminus \{x_i\})$ in $(2k_S - 1) \cdot T(|X_i|, |E[X_i]|)$ time. Then, by applying $\textsc{Increment}(X_i \setminus \{x_i\}, f(X_i \setminus \{x_i\}), x_i)$, we obtain the value $f(X_i)$ in $2k_S \cdot T(|X_i|, |E[X_i]|)$ time. Because $|S| = \sum_i |X_i|$ and $|E[S]| = \sum_i |E[X_i]|$ hold, the total running time of these computations is $2k_S \cdot \sum_i T(|X_i|, |E[X_i]|) \leq 2k_S \cdot T(|S|, |E[S]|)$. Finally, by applying the algorithm Union, we obtain the value $f(S)$ in $(2k_S + 1) \cdot T(|S|, |E[S]|)$ time. ◀

Note that the algorithm Union is trivial in most applications. We have only one non-trivial case in Section 4.5 in this paper. From the relation between tree-depth and tree-width (Lemma 1), we obtain the following corollary.

▶ **Corollary 3.** *Under the same assumption as in Theorem 2, for a given tree decomposition of G of width k, we can compute the value $f(V)$ in $O(k \cdot T(n, m) \log n)$ time.*

4 Applications

4.1 Maximum matching

For an undirected graph $G = (V, E)$, a *matching* M of G is a subset of E such that no edges in M share a vertex. In this section, we prove the following theorem.

▶ **Theorem 4.** *Given an undirected graph and its elimination forest of depth k, we can compute a maximum-size matching in $O(km)$ time.*

As mentioned in the introduction, we use the augmenting-path approach, which is also used for planar matching [21]. Let M be a matching. A vertex not incident to M is called *exposed*. An *M-alternating path* is a (simple) path whose edges are alternately out of and in M. An M-alternating path connecting two different exposed vertices is called an *M-augmenting path*. If there exists an M-augmenting path P, by taking the symmetric difference $M \Delta E(P)$, where $E(P)$ is the set of edges in P, we can construct a matching of size $|M| + 1$. In fact, M is the maximum-size matching if and only if there exist no M-augmenting paths. Edmonds [10] developed the first polynomial-time algorithm for computing an M-augmenting path by introducing the notion of blossom, and an $O(m)$-time algorithm was given by Gabow and Tarjan [14].

▶ **Lemma 5** ([14]). *Given an undirected graph and its matching M, we can either compute a matching of size $|M|+1$ or correctly conclude that M is a maximum-size matching in $O(m)$ time.*

For $S \subseteq V$, we define $f(S)$ as a function that returns a maximum-size matching of $G[S]$. We now give Increment and Union.

Increment$(X, f(X), x)$. Because the size of the maximum matching of $G[X \cup \{x\}]$ is at most the size of the maximum matching of $G[X]$ plus one, we can compute a maximum matching of $G[X \cup \{x\}]$ in $O(|E[X \cup \{x\}]|)$ time by a single application of Lemma 5.

Union$((X_1, f(X_1)), \ldots, (X_c, f(X_c)))$. Because there exist no edges between X_i and X_j for any $i \neq j$, we can construct a maximum matching of $G[\bigcup_i X_i]$ just by taking the union of $f(X_i)$.

Proof of Theorem 4. The algorithm Increment$(X, f(X), x)$ correctly computes $f(X \cup \{x\})$ in $O(|E[X \cup \{x\}]|)$ time and the algorithm Union$((X_1, f(X_1)), \ldots, (X_c, f(X_c)))$ correctly computes $f(\bigcup_i X_i)$ in $O(|\bigcup_i X_i|)$ time. Therefore, from Theorem 2, we can compute a maximum-size matching of G in $O(km)$ time. ◀

4.2 Weighted matching

Let $G = (V, E)$ be an undirected graph with an edge-weight function $w : E \to \mathbb{R}$. A weight of a matching M, denoted by $w(M)$, is simply defined as the total weight of edges in M. A matching M of G is called *perfect* if G has no exposed vertices (or equivalently $|M| = \frac{n}{2}$). A perfect matching is called a *maximum-weight perfect matching* if it has the maximum weight among all perfect matchings of G. We can easily see that other variants of weighted matching problems can be reduced to the problem of finding a maximum-weight perfect matching even when parameterized by tree-depth. In this section, we prove the following theorem.

▶ **Theorem 6.** *Given an edge-weighted undirected graph admitting at least one perfect matching and its elimination forest of depth k, we can compute a maximum-weight perfect matching in $O(k(m + n \log n))$ time.*

In our algorithm, we use an $O(n(m + n \log n))$-time primal-dual algorithm by Gabow [12]. In this primal-dual algorithm, we keep a pair of a matching M and dual variables (Ω, y, z), where Ω is a laminar[2] collection of odd-size subsets of V and y and z are functions $y : V \to \mathbb{R}$ and $z : \Omega \to \mathbb{R}_{\geq 0}$, satisfying the following conditions:

$$\widehat{yz}(uv) := y(u) + y(v) + \sum_{B \in \Omega : u, v \in B} z(B) \geq w(uv) \qquad \text{for every } uv \in E, \tag{1}$$

$$\widehat{yz}(uv) = w(uv) \qquad \text{for every } uv \in M, \tag{2}$$

$$|\{uv \in M \mid u, v \in B\}| = \left\lfloor \frac{|B|}{2} \right\rfloor \qquad \text{for every } B \in \Omega. \tag{3}$$

From the duality theory (see e.g. [13]), a perfect matching M is a maximum-weight perfect matching if and only if there exist dual variables (Ω, y, z) satisfying the above conditions. Gabow [12] obtained the $O(n(m + n \log n))$-time algorithm by iteratively applying the following lemma.

[2] A collection Ω of subsets of a ground set V is called *laminar* if for any $X, Y \in \Omega$, one of $X \cap Y = \emptyset$, $X \subseteq Y$, or $X \subseteq Y$ holds. When Ω is laminar, we have $|\Omega| = O(|V|)$.

▶ **Lemma 7** ([12]). *Given an edge-weighted undirected graph and a pair of a matching M and dual variables (Ω, y, z) satisfying the conditions (1)–(3), we can either compute a pair of a matching M' of cardinality $|M|+1$ and dual variables (Ω', y', z') satisfying the conditions (1)–(3) or correctly conclude that M is a maximum-size matching[3] in $O(m + n \log n)$ time.*

For $S \subseteq V$, we define $f(S)$ as a function that returns a pair of a maximum-size matching M_S of $G[S]$ and dual variables (Ω_S, y_S, z_S) satisfying the conditions (1)–(3). We now give Increment and Union.

Increment$(X, f(X), x)$. Let W be a value satisfying $W + y_X(v) \geq w(xv)$ for every $xv \in E[X \cup \{x\}]$. Let $y : X \cup \{x\} \to \mathbb{R}$ be a function defined as $y(x) := W$ and $y(v) := y_X(v)$ for $v \in X$. In the subgraph $G[X \cup \{x\}]$, a pair of the matching M_X and dual variables (Ω_X, y, z_X) satisfies the conditions (1)–(3). Therefore, we can apply Lemma 7. If M_X is a maximum-size matching of $G[X \cup \{x\}]$, we return M_X and (Ω_X, y, z_X). Otherwise, we obtain a matching M' of size $|M_X|+1$ and dual variables (Ω', y', z') satisfying the conditions (1)–(3). Because the cardinality of maximum-size matching of $G[X \cup \{x\}]$ is at most the cardinality of maximum-size matching of $G[X]$ plus one, the obtained M' is a maximum-size matching of $G[X \cup \{x\}]$. Therefore, we can return M' and (Ω', y', z').

Union$((X_1, f(X_1)), \ldots, (X_c, f(X_c)))$. Because there exist no edges between X_i and X_j for any $i \neq j$, we can simply return a pair of a maximum-size matching obtained by taking the union $\bigcup_i M_{X_i}$ and dual variables (Ω, y, z) such that $\Omega := \bigcup_i \Omega_{X_i}$, $y(v) := y_{X_i}(v)$ for $v \in X_i$, and $z(B) = z_{X_i}(B)$ for $B \in \Omega_{X_i}$.

Proof of Theorem 6. The algorithm Increment$(X, f(X), x)$ runs in $O(|E[X \cup \{x\}]| + |X| \log |X|)$ time and the algorithm Union$((X_1, f(X_1)), \ldots, (X_c, f(X_c)))$ runs in $O(|\bigcup X_i|)$ time. Therefore, from Theorem 2, we can compute $f(V)$ in $O(k(m + n \log n))$ time. From the duality theory, the perfect matching obtained by computing $f(V)$ is a maximum-weight perfect matching of G. ◀

4.3 Negative cycle detection and potentials

Let $G = (V, E)$ be a directed graph with an edge-weight function $w : E \to \mathbb{R}$. For a function $p : V \to \mathbb{R}$, we define an edge-weight function w_p as $w_p(uv) := w(uv) + p(u) - p(v)$. If w_p becomes non-negative for all edges, p is called a *potential on G*.

▶ **Lemma 8** ([30]). *There exists a potential on G if and only if G has no negative cycles.*

In this section, we prove the following theorem.

▶ **Theorem 9.** *Given an edge-weighted directed graph and its elimination forest of depth k, we can compute either a potential or a negative cycle in $O(k(m + n \log n))$ time.*

Suppose that we have a potential p. Because w_p is non-negative, we can compute a shortest-path tree rooted at a given vertex s under w_p in $O(m + n \log n)$ time with Dijkstra's algorithm. For any $s - t$ path, its length under w_p is exactly the length under w plus a constant $p(s) - p(t)$. Therefore, the obtained tree is also a shortest-path tree under w. Thus, we obtain the following corollary.

[3] Note that when M is not a perfect matching, this does not imply that M has the maximum weight among all the maximum-size matchings.

▶ **Corollary 10.** *Given an edge-weighted directed graph without negative cycles, a vertex s, and its elimination forest of depth k, we can compute a shortest-path tree rooted at s in $O(k(m + n \log n))$ time.*

For $S \subseteq V$, we define $f(S)$ as a function that returns either a potential $p_S : S \to \mathbb{R}$ on $G[S]$ or a negative cycle contained in $G[S]$. We now give Increment and Union.

Increment$(X, f(X), x)$. If $f(X)$ is a negative cycle, we return it. Otherwise, let $G' = (X \cup \{x\}, E')$ be the graph obtained from $G[X \cup \{x\}]$ by removing all the edges incoming to x. Let W be a value satisfying $w(xv) + W - p_X(v) \geq 0$ for every $xv \in E'$. Let $p' : X \cup \{x\} \to \mathbb{R}$ be a function defined as $p'(x) := W$ and $p'(v) := p_X(v)$ for $v \in X$. Because x has no incoming edges in G', p' is a potential on G'. Therefore, we can compute a shortest-path tree rooted at x under $w_{p'}$ in $O(|E[X]| + |X| \log |X|)$ time with Dijkstra's algorithm. Let R be the set of vertices reachable from x in G' and let $d : R \to \mathbb{R}$ be the shortest-path distance from x under $w_{p'}$. If there exists an edge $vx \in E[X \cup \{x\}]$ such that $v \in R$ and $d(v) + w_{p'}(vx) < 0$, $G[X \cup \{x\}]$ contains a negative cycle starting from x, going to v along the shortest-path tree, and coming back to x via the edge vx. Otherwise, let D be a value satisfying $w_{p'}(uv) + D - d(v) \geq 0$ for every $uv \in E[X \cup \{x\}]$ with $u \in X \setminus R$ and $v \in R$. Then, we return a function $p : X \cup \{x\} \to \mathbb{R}$ defined as $p(v) := p'(v) + d(v)$ if $v \in R$ and $p(v) := p'(v) + D$ if $v \in X \setminus R$.

▶ **Claim 1.** *p is a potential on $G[X \cup \{x\}]$.*

Proof. For every edge $uv \in E[X \cup \{x\}]$, we have

$$w_p(uv) = \begin{cases} w_{p'}(uv) + d(u) - d(v) \geq 0 & \text{if } u, v \in R, \\ w_{p'}(uv) + D - d(v) \geq 0 & \text{if } u \in X \setminus R, v \in R, \\ w_{p'}(uv) + D - D \geq 0 & \text{if } u \in X \setminus R, v \in X \setminus R. \end{cases}$$

Note that there are no edges from R to $X \setminus R$. ◀

Union$((X_1, f(X_1)), \ldots, (X_c, f(X_c)))$. If at least one of $f(X_i)$ is a negative cycle, we return it. Otherwise, we return a potential p defined as $p(v) := p_{X_i}(v)$ for $v \in X_i$.

Proof of Theorem 9. The algorithm Increment$(X, f(X), x)$ correctly computes $f(X \cup \{x\})$ in $O(|E[X]| + |X| \log |X|)$ time and the algorithm Union$((X_1, f(X_1)), \ldots, (X_c, f(X_c)))$ correctly computes $f(\bigcup_i X_i)$ in $O(|\bigcup_i X_i|)$ time. Thus, from Theorem 2, we can compute $f(V)$, i.e., either a potential on G or a negative cycle contained in G, in $O(k(m + n \log n))$ time. ◀

4.4 Minimum weight cycle

In this section, we prove the following theorem.

▶ **Theorem 11.** *Given a non-negative edge-weighted undirected or directed graph and its elimination forest of depth k, we can compute a minimum-weight cycle in $O(k(m + n \log n))$ time.*

Note that when the graph is undirected, a closed walk of length two using the same edge twice is not considered as a cycle. Therefore, we cannot simply reduce the undirected version into the directed version by replacing each undirected edge by two directed edges of both directions.

Let $G = (V, E)$ be the input graph with an edge-weight function $w : E \to \mathbb{R}_{\geq 0}$. For $S \subseteq V$, we define $f(S)$ as a function that returns a minimum-weight cycle of $G[S]$. We describe INCREMENT and UNION below.

Increment$(X, f(X), x)$. Because we have a minimum-weight cycle $f(X)$ of $G[X]$, we only need to find a minimum-weight cycle passing through x. First, we construct a shortest-path tree of $G[X \cup \{x\}]$ rooted at x and let $d : X \cup \{x\} \to \mathbb{R}$ be the shortest-path distance.

When the graph is undirected, we find an edge $uv \in E[X \cup \{x\}]$ not contained in the shortest-path tree minimizing $d(u) + w(uv) + d(v)$. If this weight is at least the weight of $f(X)$, we return $f(X)$. Otherwise, we return the cycle starting from x, going to u along the shortest-path tree, jumping to v through the edge uv, and coming back to x along the shortest-path tree. Note that this always forms a cycle because otherwise, it induces a cycle contained in $G[X]$ that has a smaller weight than $f(X)$, which is a contradiction.

We can prove the correctness of this algorithm as follows. Let W be the weight of the cycle obtained by the algorithm and let C be a cycle passing through x. Let $v_0 = x, v_1, \ldots, v_{\ell-1}, v_\ell = x$ the vertices on C in order. Because a tree contains no cycles, there exists an edge $v_i v_{i+1}$ not contained in the shortest-path tree. Therefore, the weight of C is $\sum_{j=0}^{i-1} w(v_j v_{j+1}) + w(v_i v_{i+1}) + \sum_{j=i+1}^{\ell-1} w(v_j v_{j+1}) \geq d(v_i) + w(v_i v_{i+1}) + d(v_{i+1}) \geq W$.

When the graph is directed, we find an edge $ux \in E[X \cup \{x\}]$ with the minimum $d(u) + w(ux)$. If this weight is at least the weight of $f(X)$, we return $f(X)$. Otherwise, we return the cycle starting from x, going to u along the shortest-path tree, and coming back to x through the edge ux.

Union$((X_1, f(X_1)), \ldots, (X_c, f(X_c)))$. We return a cycle of the minimum weight among $f(X_1), \ldots, f(X_c)$.

Proof of Theorem 11. The algorithm INCREMENT$(X, f(X), x)$ correctly computes $f(X \cup \{x\})$ in $O(|E[X]| + |X| \log |X|)$ time and the algorithm UNION$((X_1, f(X_1)), \ldots, (X_c, f(X_c))$ correctly computes $f(\bigcup_i X_i)$ in $O(|\bigcup_i X_i|)$ time. Therefore, from Theorem 2, we can compute a minimum-weight cycle in $O(k(m + n \log n))$ time. ◀

4.5 Replacement paths

Fix two vertices s and t. For an edge-weighed directed graph $G = (V, E)$ and an edge $e \in E$, we denote the length of the shortest $s - t$ path avoiding e by $r_G(e)$. In this section, we prove the following theorem.

▶ **Theorem 12.** *Given an edge-weighted directed graph $G = (V, E)$, a shortest $s - t$ path P, and its elimination forest of depth k, we can compute $r_G(e)$ for all edges e on P in $O(k(m + n \log n))$ time.*

Let $v_0(= s), v_1, \ldots, v_{\ell-1}, v_\ell(= t)$ be the vertices on the given shortest $s - t$ path P in order. For $i \in \{0, \ldots, \ell\}$, we denote the length of the prefix $v_0 v_1 \ldots v_i$ by $\mathrm{pref}(v_i)$ and the length of the suffix $v_i v_{i+1} \ldots v_\ell$ by $\mathrm{suf}(v_i)$. These can be precomputed in linear time.

For $S \subseteq V$, we define $G[S] \cup P$ as a graph consisting of vertices $S \cup \{v_0, \ldots, v_\ell\}$ and edges $E[S] \cup \{v_0 v_1, \ldots, v_{\ell-1} v_\ell\}$, and define $G[S] \setminus P$ as a graph consisting of vertices S and edges $E[S] \setminus \{v_0 v_1, \ldots, v_{\ell-1} v_\ell\}$. We denote the shortest-path length from u to v in $G[S] \setminus P$ by $d_S(u, v)$. For convenience, we define $d_S(u, v) = \infty$ when $u \notin S$ or $v \notin S$. We use the following lemma.

▶ **Lemma 13.** *For any $S \subseteq V$ and any $i \in \{0, \dots, \ell - 1\}$, $r_{G[S]\cup P}(v_i v_{i+1})$ is the minimum of $\mathrm{pref}(v_a) + d_S(v_a, v_b) + \mathrm{suf}(v_b)$ for $a \le i < b$.*

Proof. Any $s - t$ path avoiding $v_i v_{i+1}$ in $G[S] \cup P$ can be written as, for some $a \le i < b$, a concatenation of $s - v_a$ path Q_1, $v_a - v_b$ path Q_2 that is contained in $G[S] \setminus P$, and $v_b - t$ path Q_3. Because P is a shortest $s - t$ path in G, we can replace Q_1 by the prefix $v_0 \dots v_a$, Q_2 by the shortest $v_a - v_b$ path in $G[S] \setminus P$, and Q_3 by the suffix $v_b \dots v_\ell$ without increasing the length. Therefore, the lemma holds. ◀

We want to define $f(S)$ as a function that returns a list of $r_{G[S]\cup P}(v_i v_{i+1})$ for all $i \in \{0, \dots, \ell - 1\}$; however, we cannot do so because the length of this list is not bounded by $|S|$. Instead, we define $f(S)$ as a function that returns a list of $r_{G[S]\cup P}(v_i v_{i+1})$ for all i with $v_i \in S$. This succinct representation has enough information because, for any $v_i \notin S$, we have $r_{G[S]\cup P}(v_i v_{i+1}) = r_{G[S]\cup P}(v_{i-1} v_i)$ (or ∞ when $i = 0$). We describe Increment and Union below.

Increment$(X, f(X), x)$. By running Dijkstra's algorithm twice, we compute $d_{X\cup\{x\}}(x, v)$ and $d_{X\cup\{x\}}(v, x)$ for all $v \in X \cup \{x\}$ in $O(|E[X]| + |X| \log |X|)$ time. For $v_i \in X \cup \{x\}$, we define $L_i := \min_{a \le i, v_a \in X\cup\{x\}} (\mathrm{pref}(v_a) + d(v_a, x))$ and $R_i := \min_{b > i, v_b \in X\cup\{x\}} (d(x, v_b) + \mathrm{suf}(v_b))$. By a standard dynamic programming, we can compute L_i and R_i for all i with $v_i \in X \cup \{x\}$ in $O(|X|)$ time.

From Lemma 13, $r_{G[X\cup\{x\}]\cup P}(v_i v_{i+1}) = \mathrm{pref}(v_a) + d_{X\cup\{x\}}(v_a, v_b) + \mathrm{suf}(v_b)$ holds for some $a \le i < b$. If $d_{X\cup\{x\}}(v_a, v_b) = d_X(v_a, v_b)$ holds, we have $r_{G[X\cup\{x\}]\cup P}(v_i v_{i+1}) = r_{G[X]\cup P}(v_i v_{i+1})$, and otherwise, we have $d_{X\cup\{x\}}(v_a, v_b) = d_{X\cup\{x\}}(v_a, x) + d_{X\cup\{x\}}(x, v_b)$. Therefore, we can compute $r_{G[X\cup\{x\}]\cup P}(v_i v_{i+1})$ by taking the minimum of $r_{G[X]\cup P}(v_i v_{i+1})$ and $\min_{a \le i < b} (\mathrm{pref}(v_a) + d(v_a, x) + d(x, v_b) + \mathrm{suf}(v_b)) = L_i + R_i$.

Union$((X_1, f(X_1)), \dots, (X_c, f(X_c)))$. Let $X := \bigcup_i X_i$. Because there exist no edges between X_i and X_j for any $i \ne j$, we have $d_X(u, v) = \min_i d_{X_i}(u, v)$ for any $u, v \in X$. Therefore, from Lemma 13, we have $r_{G[X]\cup P}(v_i v_{i+1}) = \min_j r_{G[X_j]\cup P}(v_i v_{i+1})$. For efficiently computing $r_{G[X]\cup P}(v_i v_{i+1})$ for all i with $v_i \in X$, we do as follows in increasing order of i.

For each X_j, we maintain a value r_j so that $r_j = r_{G[X_j]\cup P}(v_i v_{i+1})$ always holds. Initially, these values are set to ∞. We use a heap for computing $\min_j r_j$ and updating r_j in $O(\log c)$ time. For processing i, we first update $r_j \leftarrow r_{G[X_j]\cup P}(v_i v_{i+1})$ for the set X_j containing v_i. We do not need to update $r_{j'}$ for any other set $X_{j'}$ because $r_{G[X_{j'}]\cup P}(v_i v_{i+1}) = r_{G[X_{j'}]\cup P}(v_{i-1} v_i)$ holds. Then, we compute $r_{G[X]\cup P}(v_i v_{i+1}) = \min_j r_j$.

Proof of Theorem 12. The algorithm Increment$(X, f(X), x)$ correctly computes $f(X \cup \{x\})$ in $O(|E[X]| + |X| \log |X|)$ time and the algorithm Union$((X_1, f(X_1)), \dots, (X_c, f(X_c)))$ correctly computes $f(\bigcup_i X_i)$ in $O(|\bigcup_i X_i| \log c) = O(|\bigcup_i X_i| \log |\bigcup_i X_i|)$ time. Therefore, from Theorem 2, we can compute $f(V)$, i.e., $r_{G\cup P}(e) = r_G(e)$ for all edges e on P, in $O(k(m + n \log n))$ time. ◀

4.6 2-hop cover

Let $G = (V, E)$ be a directed graph with an edge-weight function $w : E \to \mathbb{R}_{\ge 0}$. A *2-hop cover* of G is the following data structure (L^+, L^-) for efficiently answering distance queries. For each vertex $u \in V$, we assign a set $L^+(u)$ of pairs $(v, d^+_{uv}) \in V \times \mathbb{R}_{\ge 0}$ and a set $L^-(u)$ of pairs $(v, d^-_{vu}) \in V \times \mathbb{R}_{\ge 0}$. We require that, for every pair of vertices $s, t \in V$, the shortest-path distance from s to t is exactly the minimum of $d^+_{sh} + d^-_{ht}$ among all pairs $(h, d^+_{sh}) \in L^+(s)$

and $(h, d^-_{ht}) \in L^-(t)$. The *size* of the 2-hop cover is defined as $\sum_{u \in V} |L^+(u)| + |L^-(u)|$, and the *maximum label size* is defined as $\max_{u \in V} |L^+(u)| + |L^-(u)|$. Using a 2-hop cover of maximum label size T, we can answer a distance query in $O(T)$ time. In this section, we prove the following theorem.

▶ **Theorem 14.** *Given a non-negative edge-weighted directed graph and its elimination forest of depth k, we can construct a 2-hop cover of maximum label size $2k$ in $O(k(m + n \log n))$ time.*

For $S \subseteq V$, we define $f(S)$ as a function that returns a 2-hop cover of $G[S]$. We denote the shortest-path distance from s to t in $G[S]$ by $d_S(s, t)$. We denote the result of the distance query from s to t for $f(S)$ by $q_S(s, t)$. We now describe Increment and Union.

Increment$(X, f(X), x)$. Let (L^+, L^-) be the 2-hop cover of $G[X]$. By running Dijkstra's algorithm twice, we compute the shortest-path distances from x and to x in $G[X \cup \{x\}]$. Then, for each $u \in X \cup \{x\}$, we insert $(x, d_{X \cup \{x\}}(u, x))$ into $L^+(u)$ and $(x, d_{X \cup \{x\}}(x, u))$ into $L^-(u)$. Finally, we return the updated (L^+, L^-) as $f(X \cup \{x\})$.

▶ **Claim 2.** *$f(X \cup \{x\})$ is a 2-hop cover of $G[X \cup \{x\}]$.*

Proof. It suffices to show that $q_{X \cup \{x\}}(s, t) = d_{X \cup \{x\}}(s, t)$ holds for every $s, t \in X \cup \{x\}$. The claim clearly holds when $s = x$ or $t = x$. For $s, t \in X$, let $\delta := d_{X \cup \{x\}}(s, x) + d_{X \cup \{x\}}(x, t)$. Then, we have $d_{X \cup \{x\}}(s, t) = \min(d_X(s, t), \delta)$. From the construction of $f(X \cup \{x\})$, we have $q_{X \cup \{x\}} = \min(q_X(s, t), \delta) = \min(d_X(s, t), \delta)$. Therefore, the claim holds. ◀

Union$((X_1, f(X_1)), \ldots, (X_c, f(X_c)))$. Because there exist no paths connecting X_i and X_j for any $i \neq j$, we can construct a 2-hop cover of $G[\bigcup_i X_i]$ by simply concatenating the 2-hop covers $f(X_1), \ldots, f(X_c)$.

Proof of Theorem 14. The algorithm Increment$(X, f(X), x)$ correctly computes $f(X \cup \{x\})$ in $O(|E[X]| + |X| \log |X|)$ time and the algorithm Union$((X_1, f(X_1)), \ldots, (X_c, f(X_c))$ correctly computes $f(\bigcup_i X_i)$ in $O(|\bigcup_i X_i|)$ time. Therefore, from Theorem 2, we can compute a 2-hop cover in $O(k(m + n \log n))$ time. Let (L^+, L^-) be the 2-hop cover obtained by computing $f(V)$. For each element $(u, d^+_{uv}) \in L^+(u)$ or $(u, d^-_{vu}) \in L^-(u)$, v is located on the path from u to the root in the elimination forest. Therefore, we have $|L^+(u)| + |L^-(u)| \leq 2k$ for every vertex $u \in V$. ◀

5 Open problems

Is it possible to obtain a $\mathrm{tw}^{O(1)} m \operatorname{polylog}(n)$-time algorithm for the edge-disjoint maximum $s - t$ flow problem? Because it looks difficult to obtain a maximum flow for G from a maximum flow for $G - v$ in linear time, it will be difficult to apply our approach to this problem. Another open question is whether the running time for (unweighted) Maximum Matching is optimal. For this problem, as it can be solved in $O(\sqrt{n}m)$ time, our algorithm improves the general-case algorithm only when $\mathrm{td} = n^{\frac{1}{2} - \epsilon}$.

References

1 Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *SODA*, pages 1681–1697, 2015.

2 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and Fixed Parameter Subquadratic Algorithms for Radius and Diameter in Sparse Graphs. In *SODA*, pages 377–391, 2016.

3 Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *SIGMOD*, pages 349–360, 2013.

4 Takuya Akiba, Christian Sommer, and Ken-ichi Kawarabayashi. Shortest-path queries for complex networks: exploiting low tree-width outside the core. In *EDBT*, pages 144–155, 2012.

5 Norbert Blum. A new approach to maximum matching in general graphs. In *ICALP*, pages 586–597, 1990.

6 Hans L. Bodlaender, Jitender S. Deogun, Klaus Jansen, Ton Kloks, Dieter Kratsch, Haiko Müller, and Zsolt Tuza. Rankings of Graphs. *SIAM J. Discrete Math.*, 11(1):168–181, 1998.

7 Shiva Chaudhuri and Christos D. Zaroliagis. Shortest paths in digraphs of small treewidth. part I: sequential algorithms. *Algorithmica*, 27(3):212–226, 2000.

8 Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and Distance Queries via 2-Hop Labels. *SIAM J. Comput.*, 32(5):1338–1355, 2003.

9 David Coudert, Guillaume Ducoffe, and Alexandru Popa. Fully polynomial FPT algorithms for some classes of bounded clique-width graphs. *CoRR*, abs/1707.05016, 2017. URL: `http://arxiv.org/abs/1707.05016`.

10 Jack Edmonds. Paths, trees, and flowers. *Canad. J. Math.*, 17(3):449–467, 1965.

11 Fedor V. Fomin, Daniel Lokshtanov, Michał Pilipczuk, Saket Saurabh, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. In *SODA*, pages 1419–1432, 2017.

12 Harold N. Gabow. Data Structures for Weighted Matching and Nearest Common Ancestors with Linking. In *SODA*, pages 434–443, 1990.

13 Harold N. Gabow. Data Structures for Weighted Matching and Extensions to b-matching and f-factors. *CoRR*, abs/1611.07541, 2016. URL: `http://arxiv.org/abs/1611.07541`.

14 Harold N. Gabow and Robert Endre Tarjan. A Linear-Time Algorithm for a Special Case of Disjoint Set Union. *J. Comput. Syst. Sci.*, 30(2):209–221, 1985.

15 Harold N. Gabow and Robert Endre Tarjan. Faster Scaling Algorithms for General Graph-Matching Problems. *J. ACM*, 38(4):815–853, 1991.

16 Archontia C. Giannopoulou, George B. Mertzios, and Rolf Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *Theor. Comput. Sci.*, 689:67–95, 2017.

17 Zvi Gotthilf and Moshe Lewenstein. Improved algorithms for the k simple shortest paths and the replacement paths problems. *Inf. Process. Lett.*, 109(7):352–355, 2009.

18 John Hershberger, Subhash Suri, and Amit M. Bhosle. On the difficulty of some shortest path problems. *ACM Trans. Algorithms*, 3(1):5:1–5:15, 2007.

19 David R. Karger, Daphne Koller, and Steven J. Phillips. Finding the hidden path: Time bounds for all-pairs shortest paths. *SIAM J. Comput.*, 22(6):1199–1217, 1993.

20 Meir Katchalski, William McCuaig, and Suzanne M. Seager. Ordered colourings. *Discrete Mathematics*, 142(1-3):141–154, 1995.

21 Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

22 Kavindra Malik, Ashok K. Mittal, and Santosh K. Gupta. The k most vital arcs in the shortest path problem. *Oper. Res. Lett.*, 8(4):223–227, 1989.

23 George B. Mertzios, André Nichterlein, and Rolf Niedermeier. The power of data reduction for matching. *CoRR*, abs/1609.08879, 2016. URL: `http://arxiv.org/abs/1609.08879`.

24 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity: Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012.

25 Jaroslav Nešetřil and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006.

26 James B. Orlin and Antonio Sedeño-Noda. An $O(nm)$ time algorithm for finding the min length directed cycle in a graph. In *SODA*, pages 1866–1879, 2017.

27 Léon Planken, Mathijs de Weerdt, and Roman van der Krogt. Computing all-pairs shortest paths by leveraging low treewidth. *J. Artif. Intell. Res.*, 43:353–388, 2012.

28 Liam Roditty and Uri Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. *ACM Trans. Algorithms*, 8(4):33:1–33:11, 2012.

29 Alejandro A. Schäffer. Optimal Node Ranking of Trees in Linear Time. *Inf. Process. Lett.*, 33(2):91–96, 1989.

30 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003.

31 Vijay V. Vazirani. A theory of alternating paths and blossoms for proving correctness of the $O(\sqrt{V}E)$ general graph maximum matching algorithm. *Combinatorica*, 14(1):71–109, 1994.

32 Fang Wei. TEDI: efficient shortest path query answering on graphs. In *SIGMOD*, pages 99–110, 2010.

33 Virginia Vassilevska Williams and Ryan Williams. Subcubic Equivalences between Path, Matrix and Triangle Problems. In *FOCS*, pages 645–654, 2010.