Efficient and Adaptive Parameterized Algorithms on Modular Decompositions

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

Florian Nelles
Department of Computer Science, Humboldt-Universität zu Berlin, Germany
nelles@informatik.hu-berlin.de

Abstract

We study the influence of a graph parameter called modular-width on the time complexity for optimally solving well-known polynomial problems such as MAXIMUM MATCHING, TRIANGLE COUNTING, and MAXIMUM s-t VERTEX-CAPACITATED FLOW. The modular-width of a graph depends on its (unique) modular decomposition tree, and can be computed in linear time $\mathcal{O}(n+m)$ for graphs with n vertices and m edges. Modular decompositions are an important tool for graph algorithms, e.g., for linear-time recognition of certain graph classes.

Throughout, we obtain efficient parameterized algorithms of running times $\mathcal{O}(f(\mathsf{mw})n + m)$, $\mathcal{O}(n + f(\mathsf{mw})m)$, or $\mathcal{O}(f(\mathsf{mw}) + n + m)$ for low polynomial functions f and graphs of modular-width mw. Our algorithm for MAXIMUM MATCHING, running in time $\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw}\, n + m)$, is both faster and simpler than the recent $\mathcal{O}(\mathsf{mw}^4\, n + m)$ time algorithm of Coudert et al. (SODA 2018). For several other problems, e.g., TRIANGLE COUNTING and MAXIMUM b-MATCHING, we give adaptive algorithms, meaning that their running times match the best unparameterized algorithms for worst-case modular-width of $\mathsf{mw} = \Theta(n)$ and they outperform them already for $\mathsf{mw} = o(n)$, until reaching linear time for $\mathsf{mw} = \mathcal{O}(1)$.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases efficient parameterized algorithms, modular-width, adaptive algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.55

Related Version See [26], `https://arxiv.org/abs/1804.10173`, for the full version of the paper.

1 Introduction

Determining the best possible worst-case running times for computational problems lies at the heart of algorithmic research. For many intensively studied problems progress has been stalled for decades and one may suspect that the "correct" running times have already been found. While there is still only little known regarding unconditional lower bounds, the recent success of "fine-grained analysis of algorithms" has brought plenty of tight conditional lower bounds for a wealth of problems (see, e.g., [31, 5, 2]). Indeed, if one is willing to believe in the conjectured worst-case optimality of known algorithms for 3-SUM, ALL-PAIRS-SHORTEST PATHS (APSP), or SATISFIABILITY[1] then lots of other known algorithms must be optimal as well. Even if there is no general agreement on the truth of the conjectures, the previously

[1] It has been conjectured that there is no $\mathcal{O}(n^{2-\varepsilon})$ time algorithm for 3-SUM, no $\mathcal{O}(n^{3-\varepsilon})$ time for APSP, and there is no $c < 2$ such that k-SAT can be solved in time $\mathcal{O}(c^n)$ for each fixed k (SETH).

stalled work can now be focused on beating the best known times for just those problems rather than for a multitude of problems. Complementary to the quest for refuting conjectures and beating long-standing fastest algorithms, what should we do if the conjectures and implied lower bounds are true (or if we simply fail to disprove them)? Certainly, quadratic or cubic time is often too slow, even long before entering the realm of big data. Apart from heuristics and approximate algorithms, a possible solution lies in taking advantage of structure in the input and deriving worst-case running times that depend on parameters that quantify this structure. Consider for example the LONGEST COMMON SUBSEQUENCE problem, where a breakthrough result [1, 6] proved that there is no $\mathcal{O}(n^{2-\varepsilon})$ time algorithm for any $\varepsilon > 0$ unless SATISFIABILITY can be solved in $\mathcal{O}((2-\varepsilon')^n)$ time for some $\varepsilon' > 0$ and SETH fails. Long before this result, algorithms were discovered that run much faster than $\mathcal{O}(n^2)$ time when certain parameters are small (cf. [7]); curiously, a very recent result of Bringmann and Künnemann [7] shows that these are optimal modulo SETH (while giving one new optimal algorithm for binary alphabets). Similarly, for the task of sorting an array of n items, there is the (unconditional) lower bound of $\Omega(n \log n)$ for comparison-based sorting, which is matched by well-known sorting algorithms. The goal in the area of adaptive sorting is to find algorithms that are adaptive to presortedness (a.k.a., input structure) with very low running times for almost sorted inputs while maintaining competitive running times as disorder increases (cf. [12]).

The success of fine-grained analysis has rekindled the interest in outperforming (possibly optimal) worst-case running times by tailoring algorithms to benefit from input structure. This fits naturally into the framework of parameterized complexity where running times are expressed in terms of input size and one or more problem-specific parameters. Usually, this is aimed at NP-hard problems and a key goal is to obtain *fixed-parameter tractable* (FPT) algorithms that run in time $f(k)n^c$ where $f(k)$ is a (usually exponential) function of the parameter and n^c denotes a fixed polynomial in the input size n. Recent work of Giannopoulou et al. [19] has initiated a programmatic study of what they called "FPT in P", i.e., efficient parameterized algorithms for tractable problems. Here, they propose to seek running time $\mathcal{O}(k^\alpha n^\beta)$ when the best dependence on input size alone is $\mathcal{O}(n^\gamma)$ for $\gamma > \beta$; in particular, algorithms with linear dependence on the input size are sought, i.e., time $\mathcal{O}(k^\alpha n)$. Giannopoulou et al. suggest that MAXIMUM MATCHING could become a focal point of study, similar to the related NP-hard VERTEX COVER problem in parameterized complexity.

There have been several recent publications that fit into the FPT in P program [14, 28, 4, 13, 24]. Several works focus on the *treewidth* parameter, which is of core importance in parameterized complexity [14, 23]. In particular, Fomin et al. [14] obtained algorithms that depend polynomially on input size n and treewidth tw to solve a number of problems related to determinants and systems of linear inequalities; e.g., they can solve MAXIMUM MATCHING in time $\mathcal{O}(\mathsf{tw}^3\, n \log n)$ and vertex flow with unit capacities in time $\mathcal{O}(\mathsf{tw}^2\, n \log n)$. (A small caveat of treewidth in this context is that it is NP-hard to compute so one has to resort to an approximation with polynomial blow-up in the treewidth.) Iwata et al. [24] studied the related parameter *tree-depth* and, among other results, showed how to solve MAXIMUM MATCHING in time $\mathcal{O}(\mathsf{td}\, m)$ on graphs of tree-depth td. Very recently, Coudert et al. [8] studied another tree-width related parameter called *clique-width* as well as several related parameters such as modular-width and split-width; they obtain upper and lower bounds for a variety of problems. Their main result is an algorithm for MAXIMUM MATCHING that runs in $\mathcal{O}(\mathsf{mw}^4\, n + m)$ time, where mw stands for the modular-width of the input graph. Note that modular-width and the modular decomposition of a graph can be computed in linear time $\mathcal{O}(n + m)$; the modular-width is an upper bound for the (NP-hard) clique-width but it is itself unbounded already on graphs of constant clique-width.

Table 1 Overview about our results. We denote with n and m the number of vertices and edges, mw denotes the modular-width of the input graph, and λ denotes the edge-connectivity of the graph (which is upper-bounded by the minimum degree δ, so $\lambda \leq \delta \leq 2m/n$. The previous best result for MAXIMUM MATCHING, parameterized by modular-width mw, was $\mathcal{O}(\mathsf{mw}^4\, n + m)$ [8].

Problem	Best unparameterized	Our result
MAXIMUM MATCHING	$\mathcal{O}(m\sqrt{n})$ [29]	$\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw}\, n + m)$
MAXIMUM b-MATCHING[2]	$\mathcal{O}((n \log n) \cdot (m + n \log n))$ [16]	$\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw}\, n + m)$ or $\mathcal{O}((\mathsf{mw} \log \mathsf{mw}) \cdot (m + n \log \mathsf{mw}))$
TRIANGLE COUNTING	$\mathcal{O}(n^\omega)$ [32] or $\mathcal{O}(m^{\frac{2\omega}{\omega+1}}) = \mathcal{O}(m^{1.41})$ [3]	$\mathcal{O}(\mathsf{mw}^{\omega-1}\, n + m)$
EDGE-DISJOINT s-t PATHS	$\mathcal{O}(n^{\frac{3}{2}} m^{\frac{1}{2}})$ [20]	$\mathcal{O}(\mathsf{mw}^3 + n + m)$
GLOBAL MIN CUT	$\mathcal{O}(m + \lambda^2 n \log(n/\lambda))$ [15]	$\mathcal{O}(\mathsf{mw}^3 + n + m)$
MAX s-t VERTEX FLOW	$\mathcal{O}(nm)$ [30]	$\mathcal{O}(\mathsf{mw}^3 + n + m)$
GLOBAL VERTEX MIN CUT	$\mathcal{O}(n^3 \log n)$ [22]	$\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw}\, n + m)$

Our work. We further explore the algorithmic applications of modular-width for well-studied tractable problems. See Table 1 for an overview of our results. First, we improve the running time for MAXIMUM MATCHING from $\mathcal{O}(\mathsf{mw}^4\, n + m)$ to $\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw}\, n + m)$. We follow the same natural recursive approach as in previous work, i.e., computing optimal solutions in a bottom-up fashion on the modular decomposition tree. Unlike Coudert et al. [8], however, we do not seek to use the structure of modules to speed up the computation of augmenting paths, starting from an union of maximum matchings for the child modules. Instead, we simplify the current graph, while retaining the same maximum matching size, such that the found solutions can be encoded into vertex capacities in a graph with at most $3\,\mathsf{mw}$ vertices. This allows us to forget the matchings for the modules and instead of augmenting paths it suffices to find a maximum b-matching subject to vertex capacities; using an $\mathcal{O}(\min\{b(V), n \log n\} \cdot (m + n \log n)) = \mathcal{O}(n^3 \log n)$ time algorithm due to Gabow [16] then yields the claimed running time.[3]

Our algorithm for MAXIMUM MATCHING easily generalizes to computing maximum b-matchings in the same time $\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw}\, n + m)$. By a different summation of the running time, one can also bound the time by $\mathcal{O}((\mathsf{mw} \log \mathsf{mw}) \cdot (m + n \log \mathsf{mw}))$. For large total capacity $b(V)$, Gabow's algorithm runs in time $\mathcal{O}((n \log n) \cdot (m + n \log n))$, which matches our running time for graphs with worst-case modular-width of $\mathsf{mw} = \Theta(n)$.

Thus, when capacities are large, our algorithm interpolates smoothly between linear time $\mathcal{O}(n + m)$ for $\mathsf{mw} = \mathcal{O}(1)$ and the running time of the best unparameterized algorithm for $\mathsf{mw} = \Theta(n)$; i.e., it is an *adaptive algorithm* and already $\mathsf{mw} = o(n)$ gives an improved running time. Such adaptive algorithms (for other problems and parameter) were also considered by Iwata et al. [24]. For MAXIMUM MATCHING, the comparison with the $\mathcal{O}(m\sqrt{n})$ time algorithm of Micali and Vazirani [29] is of course less favorable, but still yields a fairly large regime for mw where we get a faster algorithm.

We next study TRIANGLE COUNTING where, given a graph $G = (V, E)$, we need to determine the number of triangles in G. The fastest known algorithm in terms of n relies on fast matrix multiplication and runs in $\mathcal{O}(n^\omega)$ time [32] where ω is the matrix multiplication

[2] For $b(V) \geq n \log n$

[3] The obvious upper bound of $\mathcal{O}(\mathsf{mw}^3 \log \mathsf{mw}\, n + m)$ of applying Gabow's algorithm on each prime node can be improved by a slightly more careful summation; the same applies in the other results.

exponent.[4] We present an algorithm that runs in $\mathcal{O}(\mathsf{mw}^{\omega-1}\, n + m)$ time. Again, our running time smoothly interpolates between linear time $\mathcal{O}(n+m)$ for $\mathsf{mw} = \mathcal{O}(1)$ and the best unparameterized time for $\mathsf{mw} = \Theta(n)$, making it adaptive for sufficiently dense graphs; else, the $\mathcal{O}(m^{\frac{2\omega}{\omega+1}}) = \mathcal{O}(m^{1.41})$ time algorithm of Alon et al. [3] is faster. Coudert et al. [8] obtained time $\mathcal{O}(\mathsf{cw}^2(n+m))$ where cw is the clique-width of G; this is incomparable with our result because clique-width is a smaller parameter ($\mathsf{cw} \leq \mathsf{mw}$ and there are graphs with $\mathsf{cw} = \mathcal{O}(1)$ but $\mathsf{mw} = \Theta(n)$) but (so far) allows a worse time.

Finally, we turn to problems related to edge- and vertex-disjoint paths. Due to space restrictions the discussion of those problems are deferred to the full version [26]. Our results for the vertex-disjoint paths generalize to vertex-capacitated flows and global min cuts; it is easy to see that there is little use for modular-width for most edge-weighted/capacitated problems because it suffices to solve them on cliques, which have modular-width equal to two (see also Section 5). Note that standard transformations between different variants of path- and flow-type problems do not apply here because they affect the modular-width of the graph. We obtain the following running times: MAXIMUM s-t VERTEX-CAPACITATED FLOW in $\mathcal{O}(\mathsf{mw}^3 + n + m)$ time; GLOBAL VERTEX-CAPACITATED MIN CUT in $\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw}\, n + m)$ time; EDGE-DISJOINT s-t PATHS in $\mathcal{O}(\mathsf{mw}^3 + n + m)$ time; and UNWEIGHTED GLOBAL MIN CUT in $\mathcal{O}(\mathsf{mw}^3 + n + m)$ time. The running times for flows/paths are linear in the graph size and only have an additive contribution in terms of the modular-width, because at most one involved computation (on a prime node) is needed. These also give rise to linear-time kernelization-like algorithms that return an equivalent instance of size $poly(\mathsf{mw})$, which is the one instance that one would run some other algorithm on (i.e., the only source of non-linear time). Such results (for other problems) have also been observed by Coudert et al. [8]. It is easy to see that *any* algorithm of running time $\mathcal{O}(f(k) + n + m)$, for some parameter k, implies a linear-time kernelization: Run the algorithm for $c(n+m)$ steps, for sufficiently large c relative to hidden constants in $\mathcal{O}$; it either terminates and returns the correct answer or allows the conclusion that $n + m < f(k)$, i.e., the input instance itself is the kernel. Again, as done for MAXIMUM b-MATCHING, one can obtain different bounds for the running time by slightly different summations. For example, the running time for MAXIMUM s-t VERTEX-CAPACITATED FLOW can also be bounded by $\mathcal{O}(\mathsf{mw}\, m + n)$, meaning that the algorithm is never worse than the optimal unparameterized algorithm and outperforms it already for $\mathsf{mw} = o(n)$.

To summarize, we obtain several results that fit into the recent FPT in P program (and the much older programs of adaptive algorithms and faster algorithms for restricted settings), i.e., efficient parameterized algorithms with running times $\mathcal{O}(poly(\mathsf{mw})(n+m))$ or $\mathcal{O}(poly(\mathsf{mw}) + m + n)$. All running times are linear for $\mathsf{mw} = \mathcal{O}(1)$ and several algorithms are adaptive so that they match the best known algorithm for $\mathsf{mw} = \Theta(n)$ and outperform it already when $\mathsf{mw} = o(n)$, possibly only for sufficiently dense graphs. Of course, we use the best algorithms as black boxes so the message is that throughout there is little to no overhead even in the worst case for using a modular decomposition-based approach and getting savings in running time already for large (but not worst-case) modular-width.

Related work. TRIANGLE COUNTING is solvable in time $\mathcal{O}(n^\omega)$ using fast matrix multiplication [3], and even for the simpler TRIANGLE DETECTION problem, where only (non-)existence of a single triangle needs to be reported, it has been conjectured that there is no $\mathcal{O}(n^{\omega-\varepsilon})$

[4] It is known that $2 \leq \omega < 2.3728639$ due to Le Gall [17]. By definition of ω the running time is in fact $\mathcal{O}(n^{\omega+o(1)})$; adopting a common abuse of notation we use exponent ω for brevity.

time and no combinatorial $\mathcal{O}(n^{3-\varepsilon})$ time algorithm. The fastest known algorithm for counting triangles in sparse graphs is the AYZ algorithm due to Alon, Yuster, and Zwick [3], which runs in time $\mathcal{O}(m^{\frac{2\omega}{\omega+1}})$ ($\mathcal{O}(m^{1.41})$ for $\omega < 2.373$). Coudert et al. [8] gave a faster algorithm for graphs of bounded clique-width cw, running in time $\mathcal{O}(\mathsf{cw}^2(n+m))$. Bentert et al. [4] have studied TRIANGLE ENUMERATION under various parameters including feedback edge number, distance to d-degenerate graphs, and clique-width. The latter one outputs all triangles in time $\mathcal{O}(\mathsf{cw}^2\, n + n^2 + \#T)$ where $\#T$ denotes the number of triangles in G.

The currently best maximum flow algorithm is due to Orlin [30] and runs in time $\mathcal{O}(nm)$. Using a flow algorithm, one can determine the number of edge- or vertex-disjoint s-t paths in a graph, but in the unweighted case one can do slightly better, e.g., computing the number of edge-disjoint paths in an undirected graph can be done in time $\mathcal{O}(n^{\frac{3}{2}}m^{\frac{1}{2}})$ using an algorithm due to Goldberg and Rao [20]. Finding a global minimum edge cut with weights on the edges in an undirected graph can be done in time $\mathcal{O}(nm + n^2 \log n)$ due to Stoer and Wagner [33]. The unweighted variant can be solved in time $\mathcal{O}(m + \lambda^2 n \log(n/\lambda))$ by Gabow [15], where λ denotes the edge-connectivity of the graph (which is upper-bounded by the minimum degree δ, so $\lambda \leq \delta \leq 2m/n$). There is also a randomized algorithm with running time $\mathcal{O}(m \log^3 n)$ due to Karger [25].

The notion of a modular decomposition was first introduced by Gallai [18] for recognizing comparability graphs. The first linear time algorithm to compute a modular decomposition was independently developed by McConnell and Spinrad [27] and Cournier and Habib [9]. Tedder et al. [34] later gave a new and much simpler linear-time algorithm.

Organization. In Section 2 we briefly introduce basic notation, define the modular decomposition tree, and define modular-width. Then, in Section 3, we consider the problem MAXIMUM MATCHING and the generalization to MAXIMUM b-MATCHING. In Section 4, we study the problem TRIANGLE COUNTING. Due to space restrictions, the remaining results for edge/vertex-disjoint paths, flows, and cuts can be found in the full version [26]. We conclude in Section 5.

2 Preliminaries

We use standard graph notation [10]. An *s-t vertex-capacitated flow* in a graph $G = (V, E)$ with vertex capacities $c\colon V \to \mathbb{R}$ is a weighted collection of s-t paths in G such that the total weight of paths including any vertex $v \in V \setminus \{s, t\}$ is at most the capacity $c(v)$. (Equivalently, one may define this as a function $f\colon E(\overleftrightarrow{G}) \to \mathbb{R}$ where $\overleftrightarrow{G} = (V, A)$ with $A = \{(u, v), (v, u) \mid \{u, v\} \in E\}$ that has flow-conservation at each $v \in V \setminus \{s, t\}$ and with $\sum_{(u,v)\in\delta^-_{\overleftrightarrow{G}}(v)} f((u, v)) \leq c(v)$ for all $v \in V \setminus \{s, t\}$, where $\delta^-_{\overleftrightarrow{G}}(v)$ is the set of arcs with end in v.) The value of such a flow, denoted by $|f|$, is the total weight over all the s-t paths (equivalently, $\sum_{(v,t)\in\delta^-_{\overleftrightarrow{G}}(t)} f(v, t)$). For unit capacities $c \equiv 1$ this is equivalent to a maximum collection of vertex-disjoint s-t paths.

We say that two sets A and B *overlap* if $A \cap B \neq \emptyset$, $A \setminus B \neq \emptyset$, and $B \setminus A \neq \emptyset$ and let $[n] = \{1, 2, \ldots, n\}$ for any $n \in \mathbb{N}$.

Modular Decomposition. Let $G = (V, E)$ be a graph. A *module* is a vertex set $M \subseteq V$ such that all vertices in M have the same neighborhood in $V \setminus M$. In other words, $M \subseteq N(x)$ or $M \cap N(x) = \emptyset$ for every vertex $x \in V \setminus M$. Clearly, $\emptyset$, V, and $\{v\}$ for every $v \in V$ are modules of G; these are called *trivial modules*. If a graph only admits trivial modules, we call G *prime*. Consider a partition $P = \{M_1, M_2, \ldots, M_\ell\}$ of the vertices of G into modules where $\ell \geq 2$, called *modular partition*. If there is $v \in M_i$ and $u \in M_j$ with $\{u, v\} \in E$, then

any vertex in M_i is adjacent to every vertex in M_j. In this case, we can call two modules M_i and M_j of P *adjacent*, and *non-adjacent* otherwise.

▶ **Definition 1.** Let $P = \{M_1, M_2, \ldots, M_\ell\}$ be a modular partition of a graph $G = (V, E)$. The *quotient graph* $G_{/P} = (\{q_{M_1}, q_{M_2}, \ldots, q_{M_\ell}\}, E_P)$ is the graph whose vertices are in a one-to-one correspondence to the modules in P. Two vertices q_{M_i}, q_{M_j} of $G_{/P}$ are adjacent if and only if the corresponding modules M_i and M_j are adjacent (with adjacency as above).

If $P = \{M_1, M_2, \ldots, M_\ell\}$ is a modular partition of a graph G, then the quotient graph $G_{/P}$ is a compact representation of the edges with endpoint in different modules. Together with all subgraphs $G[M_i]$, with $i \in [\ell]$, we can reconstruct G. Each subgraph $G[M_i]$ is called a *factor*. Instead of specifying the factors, one can recursively decompose them as well until one reaches trivial modules $\{v\}$. To make the decomposition unique, one considers modular partitions consisting of strong modules. A module of a graph G is called a *strong* module, if it does not overlap with any other module of G. One can represent all strong modules of a graph G by an inclusion tree $MD(G)$. Each strong module M in G corresponds to a vertex v_M in $MD(G)$. A vertex v_A is an an ancestor of v_B in $MD(G)$ if and only if $B \subsetneq A$ for the corresponding strong modules A and B of G. Hence, the root node of $MD(G)$ corresponds always to the complete vertex set V of G and every leaf of $MD(G)$ corresponds a singleton set $\{v\}$ with $v \in V$. Consider an internal node v_M of $MD(G)$ with the set of children $\{v_{M_1}, \ldots, v_{M_\ell}\}$, i.e., v_M corresponds to a strong module M of G and $P = \{M_1, \ldots, M_\ell\}$ is a modular partition of $G[M]$ into strong modules where M_i is the corresponding module of v_{M_i}, with $i \in [\ell]$. There are three types of internal nodes in $MD(G)$. A node v_M in $MD(G)$ is *degenerate*, if for any non-empty subset of the children of v_M in $MD(G)$, the union of the corresponding modules induces a (not necessarily strong) module. In this case the quotient graph $G[M]_{/P}$ is either a clique or an independent set. In the former case one calls v_M a *series* node, in the later a *parallel* node. Another case are so called *prime* nodes. Here, for no proper subset of the children of v_M, the union of the corresponding modules induces a module. In this case the quotient graph of v_M is prime. Gallai showed there are no further nodes in $MD(G)$.

▶ **Theorem 2** ([18]). *For any graph $G = (V, E)$ one of the three conditions is satisfied:*

- *G is not connected,*
- *$\overline{G}$ is not connected,*
- *G and $\overline{G}$ are connected and the quotient graph $G_{/P}$, where P is the maximal modular partition of G, is a prime graph.*

Theorem 2 implies that $MD(G)$ is unique. The tree $MD(G)$ is called the *modular decomposition tree* and the *modular-width*, denoted by $\mathsf{mw} = \mathsf{mw}(G)$, is the minimum $k \geq 2$ such that any prime node in $MD(G)$ has at most k children. Since every node in $MD(G)$ has at least two children and there are exactly n leaves, $MD(G)$ has at most $2n - 1$ nodes. It is known that $MD(G)$ can be computed in time $\mathcal{O}(n + m)$ [34]. We refer to a survey of Habib and Paul [21] for more information.

3 Maximum Matching

In the MAXIMUM MATCHING problem we are given a graph $G = (V, E)$ and need to find a maximum set $X \subseteq E$ of pairwise disjoint edges. The size of a maximum matching of a graph G is denoted by $\mu(G)$. Edmond [11] was the first to give a polynomial-time algorithm for this

problem. The fastest known algorithm, due to Micali and Vazirani [29], runs in time $\mathcal{O}(m\sqrt{n})$. A *b-matching* is a generalization of a matching that specifies for each vertex a *degree bound* of how many edges in the matching may be incident with that vertex. Formally, degree bounds are given by a function $b\colon V \to \mathbb{N}$, and a b-matching is a function $x\colon E \to \mathbb{N}$ that fulfills for every vertex $v \in V$ the constraint that $\sum_{e\in\delta(v)} x(e) \leq b(v)$. Gabow [16] showed how to find a b-matching that maximizes $\sum_{e\in E} x(e)$ in time $\mathcal{O}((n \log n) \cdot (m + n \log n))$.

Recently, Coudert et al. [8] gave an $\mathcal{O}(\mathsf{mw}^4\, n{+}m)$ time algorithm for MAXIMUM MATCHING, where mw denotes the modular-width of the input graph. In the following we will improve this result by providing an algorithm for MAXIMUM MATCHING that runs in time $\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw} \cdot n + m)$. The main idea of our algorithm is to compress the computation of a matching in G to a computation of a b-matching, instead of using the structure of modular decompositions to speed up the search for augmenting paths (like in [8]).

▶ **Theorem 3.** *For every graph $G = (V, E)$ with modular-width mw,* MAXIMUM MATCHING *can be solved in time $\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw} \cdot n + m)$.*

Algorithm. First, we compute the modular decomposition tree $MD(G)$. We will traverse the decomposition tree in a bottom-up manner. For each v_M in $MD(G)$, with M denoting the corresponding module of G, we will compute a maximum matching in $G[M]$. Note that for the root module v_M of $MD(G)$ it holds that $G[M] = G$. For any leaf module v_M of $MD(G)$, we have $\mu(G[M]) = 0$, since $G[M]$ is a graph consisting of a single vertex. Let v_M be a non-leaf vertex in $MD(G)$ with the set of children $\{v_{M_1}, \ldots, v_{M_\ell}\}$. This means that $\{M_1, \ldots, M_\ell\}$ is a modular partition of $G[M]$, where $M_i \subseteq M$ corresponds to the vertex v_{M_i} in $MD(G)$ for $i \in [\ell]$. In the following, we can always assume that we have already computed $\mu(G[M_i])$ for $i \in [\ell]$. The next lemma shows that the concrete structure inside a module is irrelevant for the maximum matching size of the whole graph, i.e., only the number of vertices and the maximum matching size is important. The lemma is a more general version of [8, Lemma 5.1], but can be proven in a similar way.

▶ **Lemma 4.** *Let M be a module of $G = (V, E)$ and let $G[M] = (M, E_M)$. Let $A \subseteq \binom{M}{2}$ be any set of edges on the vertices of M such that $\mu((M, A)) = \mu((M, E_M))$. Then, the size of a maximum matching of $G' = (V, (E \setminus E_M) \cup A)$ is equal to the size of a maximum matching of G.*

Proof. We first show that $\mu(G') \geq \mu(G)$. Let us consider a maximum matching $F \subseteq E$ in $G = (V, E)$. To get a maximum matching in G' we replace all edges in F that are incident with M: First, replace all edges in $F \cap E(G[M])$ by an arbitrary matching $A' \subseteq A$ of the same size; such a matching must exist because $F \cap E(G[M])$ is not larger than a maximum matching in $G[M]$ and $\mu((M, A)) = \mu((M, E_M))$. Second, we replace all edges in F that have exactly one endpoint in M as follows: Let $X \subseteq M$ be the set of vertices in M that are endpoints of an edge in F whose other endpoint is not in M. By assumption, $|M \setminus V(A')| \geq |X|$ and since all vertices in $V \setminus M$ that are connected to a vertex in X in G are also connected to all vertices in $M \setminus V(A')$ in G', we can replace all edges of F that have exactly one endpoint in M. Thus, $\mu(G') \geq \mu(G)$, i.e., replacing the edges in a module by an arbitrary set of edges with same maximum matching size does not decrease the size of the maximum matching for the whole graph. Applying this argument for $A' := E_M$ to swap back to the original edge set yields, $\mu(G) \geq \mu(G')$ and completes the proof. ◀

We now describe how to compute $\mu(G[M])$ for a node v_M in $MD(G)$. Let $\{v_{M_1}, \ldots, v_{M_\ell}\}$ be the set of children of v_M in $MD(G)$, meaning that $P = \{M_1, \ldots, M_\ell\}$ is a modular partition of $G[M]$. We can assume that we have already computed $\mu(G[M_i])$ for $i \in [\ell]$.

Let $G[M]_{/P}$ be the quotient graph of $G[M]$. If v_M is a parallel node then $G[M]_{/P}$ is edgeless, i.e., $G[M]$ is the disjoint union of all $G[M_i]$. In this case a maximum matching for $G[M]$ simply consists of the union of maximum matchings for each $G[M_i]$ and we set $\mu(G(M)) = \sum_{i\in[\ell]} \mu(G[M_i])$. Next, suppose that v_M is a prime node. We will reduce the problem of computing a maximum matching in $G[M]$ to computing a maximum b-matching in an auxiliary graph closely related to the quotient graph of v_M that we will define next.

▶ **Definition 5.** Let $G = (V, E)$ be a graph and $P = \{M_1, \ldots, M_\ell\}$ be a modular partition of G. Let n_i denote the number of vertices in $G[M_i]$ and f_i the size of a maximum matching in $G[M_i]$. We define an auxiliary graph $G^* = (V^*, E^*)$ together with degree bounds $b^* \colon V^* \to \mathbb{N}$ as an instance (G^*, b^*) for the maximum b-matching problem as follows:

- For every module $M_i \in P$, with $i \in [\ell]$, we add three vertices v_i^1, v_i^2, v_i^3 to V^* and set $b^*(v_i^1) = b^*(v_i^2) = f_i$ and $b^*(v_i^3) = n_i - 2f_i$.
- We add the edge $\{v_i^1, v_i^2\}$ to E^* for $i \in [\ell]$.
- For each edge between vertices q_i and q_j in $G_{/P}$ that corresponds to modules M_i and M_j, we add the nine edges $\{v_i^c, v_j^d\}$ with $c, d \in \{1, 2, 3\}$ to E^*.

▶ **Lemma 6.** *Let $G = (V, E)$ be a graph and $P = \{M_1, \ldots, M_\ell\}$ be a modular partition of G. Let (G^*, b^*) be the instance of a maximum b-matching problem as defined in Definition 5. Then the size of maximum matchings in G is equal to the size of a maximum b-matching of (G^*, b^*).*

Proof. Consider a graph $G = (V, E)$ with a modular partition $P = \{M_1, \ldots, M_\ell\}$. For $M_i \in P$ let $n_i = |V(G[M_i])|$ and let $f_i = \mu(G[M_i])$. Due to Lemma 4, we can replace each $G[M_i]$, for $i \in [\ell]$, by a graph consisting of a complete bipartite graph K_{f_i,f_i} together with $n_i - 2f_i$ single vertices without changing the size of a maximum matching. We do this for every module $M_i \in P$ and denote the resulting graph by $\overline{G}$. Note, that $\mu(G) = \mu(\overline{G})$. Now, each replacement of $G[M_i]$ can be partitioned into three modules, namely the two parts of the complete bipartite graph K_{f_i,f_i} and the one set consisting of $n_i - 2f_i$ single vertices. This results in a modular partition P' of $\overline{G}$ of size 3ℓ, and for every module $M \in P'$ the factor graph $G[M]$ is an independent set. The quotient graph $\overline{G}_{/P'}$ is exactly the auxiliary graph G^* of G and the degree bound of a vertex v in G^* is equal to the number of vertices in the corresponding module. Since solving a b-matching in (G^*, b^*) directly corresponds to solving maximum matching in $\overline{G}$, this completes the proof. ◀

Finally, suppose that v_M is a series node. Instead of computing $\mu(G[M])$ directly, we will modify the decomposition tree $MD(G)$ (cf. [8]). Let $\{v_{M_1}, \ldots, v_{M_\ell}\}$ be the children of v_M in $MD(G)$. We will iteratively compute a maximum matching for $G_i = G[\cup_{1\le j\le i} M_j]$ by using a modular partition of G_i consisting of the two modules $\cup_{1\le j<i} M_j$ and M_i, for $i \in [\ell]$. This means that we replace a series node with ℓ children by $\ell - 1$ series nodes with only two children. We will treat the newly inserted nodes as prime nodes (with a quotient graph isomorphic to K_2). After replacing the series nodes of the modular decomposition tree $MD(G)$, every node still has at least two children; hence, we still have a most $2n - 1$ nodes in $MD(G)$.

Running Time. Consider a graph $G = (V, E)$ with modular-width mw. Computing the modular decomposition tree $MD(G)$ takes time $\mathcal{O}(n + m)$. Since there are at most $2n - 1$ nodes in $MD(G)$ the total computation for all parallel nodes together takes time $\mathcal{O}(n)$. As described above, we modify the decomposition tree such that every series node of $MD(G)$

with $\ell \geq 3$ children is replaced by $\ell - 1$ 'pseudo-prime' nodes with exactly two children. This replacement can be done in time $O(n)$. Now, every node $v_M \in MD(G)$ that is not a parallel node has a set of children $\{v_{M_1}, \ldots, v_{M_\ell}\}$ with $\ell \leq \mathsf{mw}$. This means that $P = \{M_1, \ldots, M_\ell\}$ is a modular partition of $G[M]$ and the quotient graph $G[M]_{/P}$ consists of $\ell \leq \mathsf{mw}$ vertices. Since we have already computed $\mu(G[M_i])$ for all $i \in [\ell]$, we can construct the auxiliary graph G^* of $G[M]$ as defined in Definition 5 in time $\mathcal{O}(V(G^*) + E(G^*)) = \mathcal{O}(\ell^2)$. Recall, that $|V(G^*)| = 3\ell$. Thus, we can compute a maximum b-matching of G^* subject to b in time $\mathcal{O}(\ell^3 \log \ell)$ using the algorithm due to Gabow [16]. We have to do this for every prime and series node, but a slightly more careful summation of running times over all nodes gives an improvement over the obvious upper bound of $\mathcal{O}(\mathsf{mw}^3 \log \mathsf{mw} \cdot n + m)$: Let t be the number of nodes in $MD(G)$ and for a node v_{M_i} in $MD(G)$ let ℓ_i denote the number of children, i.e. the number of vertices of the quotient graph of $G[M_i]$. Then, neglecting constant factors and assuming that $MD(G)$ is already computed, we can solve MAXIMUM MATCHING, in time:

$$\sum_{i=1}^{t} \ell_i^3 \log \ell_i \leq \left(\sum_{i=1}^{t} \ell_i \right) \cdot \max_{i \in [t]} \{\ell_i^2 \log \ell_i\} \leq 2n \cdot \max_{i \in [t]} \{\ell_i^2 \log \ell_i\} \leq 2n \cdot (\mathsf{mw}^2 \log \mathsf{mw})$$

The second inequality holds, since $\sum_{i=1}^{t} \ell_i$ counts each node in $MD(G)$ once, except for the root. Since constant factors propagate through the inequality, the total running time of the algorithm is $\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw} \cdot n + m)$, which proves Theorem 3.

Generalization to b-matching. We can easily generalize this result to the more general maximum b-matching problem.

▶ **Theorem 7.** *For every graph $G = (V, E)$ with modular-width* mw*,* MAXIMUM b-MATCHING *can be solved in time $\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw} \cdot n + m)$.*

Again, the concrete structure inside a module will not be important. The only important information is the size of a maximum b-matching and the sum of all b-values in a module. We naturally extend Definition 5 to b-matchings:

▶ **Definition 8.** Let $G = (V, E)$ be a graph with $b \colon V \to \mathbb{N}$ and let $P = \{M_1, \ldots, M_\ell\}$ be a modular partition of G. Let $n_i = \sum_{v \in M_i} b(v)$ and f_i be the size of a maximum b-matching in $G[M_i]$ for $i \in [\ell]$. We define the auxiliary graph $G^* = (V^*, E^*)$ together with degree bounds $b^* \colon V \to \mathbb{N}$ in the same way as done in Definition 5.

▶ **Lemma 9.** *Let $G = (V, E)$ be a graph and $P = \{M_1, \ldots, M_\ell\}$ be a modular partition of G. Let (G^*, b^*) be the instance of a maximum b-matching problem as defined in Definition 8. Then the size of a maximum b-matching in (G, b) is equal to the size of a maximum b-matching of (G^*, b^*).*

Proof. Consider a graph $G = (V, E)$ with a modular partition $P = \{M_1, \ldots, M_\ell\}$. For $M_i \in P$ let $n_i = \sum_{v \in M_i} b(v)$ and let f_i be the size of a maximum b-matching in M_i. Note, that one can solve b-matching by replacing every vertex v by $b(v)$ copies that are connected in the same way as v. After considering this replacement and due to Lemma 4, we can replace $G[M_i]$, for $i \in [\ell]$, by a graph consisting of a complete bipartite graph K_{f_i, f_i} together with $n_i - 2f_i$ single vertices without changing the size of a maximum matching. We do this for every module M_i and denote the resulting graph by $\overline{G}$. As in the proof of Lemma 6, we can subdivide every module in three parts. This yields to the instance (G^*, b^*) as defined in Definition 8. Again, solving a maximum b-matching of (G^*, b^*) directly corresponds to solving a maximum b-matching in $\overline{G}$, which completes the proof. ◀

The running time can be bounded in the same way as before. However, to see that this algorithm is also adaptive for sparse graphs (at least for large b-values), we can modify the computation of the running time: Let t be the number of nodes in $MD(G)$. For a node v_{M_i} in $MD(G)$ let n_i denote the number of vertices and m_i denote the number of edges of the quotient graph of $G[M_i]$. Thus, we can compute a maximum b-matching of G^* subject to b^* in time $\mathcal{O}((n_i \log n_i) \cdot (m_i + n_i \log n_i))$ using the algorithm due to Gabow [16]. Then, neglecting constant factors and assuming that $MD(G)$ is already computed, we can solve MAXIMUM b-MATCHING in time:

$$\begin{aligned}\sum_{i=1}^{t}(n_i \log n_i) \cdot (m_i + n_i \log n_i) &= \sum_{i=1}^{t} m_i n_i \log n_i + \sum_{i=1}^{t} n_i^2 \log^2 n_i \\ &\leq \left(\sum_{i=1}^{t} m_i\right) \max_{i\in[t]}\{n_i \log n_i\} + \left(\sum_{i=1}^{t} n_i\right) \max_{i\in[t]}\{n_i \log^2 n_i\} \\ &\leq m \cdot \mathsf{mw} \log \mathsf{mw} + 2n \cdot (\mathsf{mw} \log^2 \mathsf{mw})\end{aligned}$$

Since constant factors propagate through the inequality, the total running time of the algorithm is $\mathcal{O}((m + n \log \mathsf{mw}) \cdot (\mathsf{mw} \log \mathsf{mw}))$. Therefore, even for $\mathsf{mw} = \Theta(n)$ our algorithm is not worse than the (currently) best unparameterized algorithm, assuming $b(V) \geq n \log n$, where $b(V) = \sum_{v\in V} b(v)$.

4 Triangle Counting

In this section we consider the TRIANGLE COUNTING problem, in which one is interested in the number of triangles in the input graph.

▶ **Theorem 10.** *For every graph $G = (V, E)$ with modular-width* mw*,* TRIANGLE COUNTING *can be solved in time $\mathcal{O}(n \cdot \mathsf{mw}^{\omega-1} + m)$.*

Algorithm. First, we compute the modular decomposition tree $MD(G)$. We will process $MD(G)$ in a bottom-up manner. For each v_M in $MD(G)$, with corresponding module M in G, we will compute the following three values: the number of vertices $n_M = |V(G[M])|$, the number of edges $m_M = |E(G[M])|$, and the number of triangles t_M in $G[M]$. For any leaf node v_M in $MD(G)$ we have $n_M = 1$ and $m_M = t_M = 0$, because $G[M]$ consists of a single vertex. Let v_M be a non-leaf node in $MD(G)$ with children $\{v_{M_1}, \ldots, v_{M_\ell}\}$. Since we process $MD(G)$ in a bottom-up manner, the values for $G[M_i]$ are already computed for $i \in [\ell]$. If v_M is a parallel node, the values simply add up, i.e. $n_M = \sum_{i=1}^{\ell} n_{M_i}$, $m_M = \sum_{i=1}^{\ell} m_{M_i}$, and $t_M = \sum_{i=1}^{\ell} t_{M_i}$. If v_M is a series node, we will use the same approach as in Section 3 and replace v_M by $\ell - 1$ series nodes with only two children each. Afterwards, we compute the values for a series node v_M with children v_{M_1} and v_{M_2} as follows:

$$\begin{aligned}n_M &= n_{M_1} + n_{M_2} \\ m_M &= m_{M_1} + m_{M_2} + n_{M_1} n_{M_2} \\ t_M &= t_{M_1} + t_{M_2} + m_{M_1} n_{M_2} + m_{M_2} n_{M_1}\end{aligned}$$

Finally, let v_M be a prime node in $MD(G)$ and let $\{v_{M_1}, \ldots, v_{M_\ell}\}$ be the children of v_M in $MD(G)$. This means that $P = \{M_1, \ldots, M_\ell\}$ is a modular partition of $G[M]$. Again, $n_M = \sum_{i=1}^{\ell} n_{M_i}$ and we can compute m_M by traversing all edges in the quotient graph

$G[M]_{/P}$, i.e., $m_M = \sum_{i=1}^{\ell} m_{M_i} + \sum_{\{q_i,q_j\}\in E(G[M]_{/P})} n_{M_i} n_{M_j}$. For computing t_M we count triangles in $G[M]$ of three types: Triangles using vertices in exactly one module, in two (adjacent) modules, or in three modules of P. We call a triangle with vertices in three different modules a *separated* triangle. To compute the number of separated triangles, we use the following lemma:

▶ **Lemma 11.** *Let $G = (V, E)$ be a graph with a modular partition $P = \{M_1, \ldots, M_\ell\}$ and quotient graph $G_{/P}$. Let $n_{M_i} := |M_i|$ and consider the weight function $w\colon E(G_{/P}) \to \mathbb{R}^+$ with $w(\{q_i, q_j\}) = \sqrt{n_{M_i} n_{M_j}}$. Let A be the weighted adjacency matrix of $G_{/P}$ with respect to w. Then, the number of separated triangles in G is:*

$$\sum_{i,j=1}^{\ell} \frac{1}{3}(A^2 \circ A)_{i,j},$$

where $A \circ B$ denotes the Hadamard product of the matrices A and B, i.e., $(A \circ B)_{i,j} = A_{i,j} B_{i,j}$.

Proof. To count all separated triangles in G we need to sum up the values $n_{M_i} n_{M_j} n_{M_k}$ for each triangle (q_i, q_j, q_k) in $G_{/P}$. We show, that $(A^2 \circ A)_{i,j}$ exactly corresponds to the number of separated triangles in G with one vertex in M_i and one in M_j; here, a *wedge* is a path on three vertices (and a wedge (q_i, q_k, q_j) requires the presence of edges $\{q_i, q_k\}$ and $\{q_k, q_j\}$):

$$\begin{aligned}
\left(A^2\right)_{i,j} &= \sum_{k=1}^{\ell} A_{i,k} A_{k,j} \\
&= \sum_{\substack{k:(q_i,q_k,q_j) \\ \text{is a wedge in } G_{/P}}} \sqrt{n_{M_i} n_{M_k}} \sqrt{n_{M_k} n_{M_j}} \\
&= \sqrt{n_{M_i} n_{M_j}} \sum_{\substack{k:(q_i,q_k,q_j) \\ \text{is a wedge in } G_{/P}}} n_{M_k} \\
\Rightarrow \left(A^2 \circ A\right)_{i,j} &= \sum_{\substack{k:(q_i,q_k,q_j) \\ \text{is a triangle in } G_{/P}}} n_{M_i} n_{M_j} n_{M_k}
\end{aligned}$$

Since every triangle is counted three times (once for each edge) the lemma follows. ◀

Using Lemma 11, we can compute t_M by

$$t_M = \sum_{i=1}^{\ell} t_{M_i} + \sum_{\{q_i,q_j\}\in E(G_{/P})} \left(m_{M_i} n_{M_j} + n_{M_i} m_{M_j}\right) + \sum_{i,j=1}^{\ell} \frac{1}{3} \left(A^2 \circ A\right)_{i,j},$$

where the three terms refer to triangles with vertices from only one module, triangles using vertices of two adjacent modules, and separated triangles with vertices in three different (pairwise adjacent) modules.

Running Time. Computing the modular decomposition tree $MD(G)$ takes time $\mathcal{O}(n + m)$. Consider a node v_M in $MD(G)$ with children $\{v_{M_1}, \ldots, v_{M_\ell}\}$. If v_M is a parallel or a series node then we can compute the values n_M, m_M, and t_M for $G[M]$ in time $\mathcal{O}(\ell)$. Thus, since the number of nodes in $MD(G)$ is at most $2n - 1$, the total running time for all parallel and series nodes is $O(n)$. Assume that v_M is a prime node. Recall, that $P = \{M_1, \ldots, M_\ell\}$ is a modular partition of $G[M]$. Computing n_M takes time $\mathcal{O}(\ell)$ and computing m_M takes

time $\mathcal{O}(|E(G[M]_{/P})|) = \mathcal{O}(\ell^2)$. The running time for computing t_M is dominated by the computation of A^2, which takes time $\mathcal{O}(\ell^\omega)$. Note, that $2 \leq \ell \leq \mathsf{mw}$. By a similar careful summation as done in Section 3 we can improve the obvious upper bound of $\mathcal{O}(n \cdot \mathsf{mw}^\omega + m)$: Let p be the number of nodes in $MD(G)$ and for v_{M_i} in $MD(G)$ let ℓ_i be the number of children, i.e., the number of vertices of the quotient graph of $G[M_i]$. Neglecting constant factors and assuming that $MD(G)$ is already computed, the running time is:

$$\sum_{i=1}^{p} \ell_i^\omega \leq \left(\sum_{i=1}^{p} \ell_i\right) \max_{i \in [p]} \ell_i^{\omega-1} \leq 2n \cdot \mathsf{mw}^{\omega-1}$$

Again, since constant factors propagate through the inequalities, the total running time of the algorithm is $\mathcal{O}(n \cdot \mathsf{mw}^{\omega-1} + m)$, which proves Theorem 10. Note, that this algorithm is adaptive for dense graphs, meaning that even for $\mathsf{mw} = \Theta(n)$ our algorithm is not worse than $\mathcal{O}(n^\omega)$.

5 Conclusion

We have obtained efficient parameterized algorithms for MAXIMUM MATCHING, MAXIMUM b-MATCHING, TRIANGLE COUNTING, and several path- and flow-type problems with respect to the modular-width mw of the input graph. All time bounds are of form $\mathcal{O}(f(\mathsf{mw})n + m)$, $\mathcal{O}(n + f(\mathsf{mw})m)$, or $\mathcal{O}(f(\mathsf{mw}) + n + m)$, where the latter can be easily seen to imply linear-time preprocessing to size $\mathcal{O}(f(\mathsf{mw}))$. Throughout, the dependence $f(\mathsf{mw})$ is very low and several algorithms are adaptive in the sense that their time bound interpolates smoothly between $\mathcal{O}(n+m)$ when $\mathsf{mw} = \mathcal{O}(1)$ and the best known unparameterized running time when $\mathsf{mw} = \Theta(n)$. Thus, even if typical inputs may have modular width $\Theta(n)$ (a caveat that all structural parameters face to some degree), using these algorithms costs only a constant-factor overhead and already $\mathsf{mw} = o(n)$ yields an improvement over the unparameterized case.

As mentioned in the introduction, (low) modular-width seems useless in problems where edges are associated with weights and/or capacities. Intuitively, these numerical values distinguish edges between adjacent modules M and M', which could otherwise be treated as largely equivalent. For concreteness, consider an instance (G, s, t, w) of the SHORTEST s,t-PATH problem where $w\colon E(G) \to \mathbb{N}$ are the edge weights. Clearly, the distance from s to t is unaffected if we add the missing edges of G and let their weight exceed the sum of weights in w. However, the obtained graph is a clique and has constant modular-width. Similar arguments work for other edge-weighted/capacitated problems like MAXIMUM FLOW using either huge or negligible weights. In each case, running times of form $\mathcal{O}(f(\mathsf{mw})g(n, m))$ would imply time $\mathcal{O}(g(n, m))$ for the unparameterized case (without considering modular-width), so the best such running times cannot be outperformed even for low modular-width.

Apart from developing further efficient (and adaptive?) parameterized algorithms relative to modular-width there are other directions of future work. Akin to conditional lower bounds via fine-grained analysis of algorithms it would be interesting to prove optimality of efficient parameterized algorithms for all regimes of the parameters (e.g., like Bringmann and Künnemann [7]). Which other (graph) parameters allow for adaptive parameterized running times so that even nontrivial upper bounds on the parameter imply faster algorithms than the unparameterized worst case?

References

1. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78, 2015. doi:10.1109/FOCS.2015.14.
2. Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 41–50. ACM, 2015. doi:10.1145/2746539.2746594.
3. Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.
4. Matthias Bentert, Till Fluschnik, André Nichterlein, and Rolf Niedermeier. Parameterized aspects of triangle enumeration. In *Fundamentals of Computation Theory - 21st International Symposium, FCT 2017, Bordeaux, France, September 11-13, 2017, Proceedings*, pages 96–110, 2017. doi:10.1007/978-3-662-55751-8_9.
5. Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.76.
6. Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97, 2015. doi:10.1109/FOCS.2015.15.
7. Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1216–1235, 2018. doi:10.1137/1.9781611975031.79.
8. David Coudert, Guillaume Ducoffe, and Alexandru Popa. Fully polynomial FPT algorithms for some classes of bounded clique-width graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2765–2784, 2018. doi:10.1137/1.9781611975031.176.
9. Alain Cournier and Michel Habib. A new linear algorithm for modular decomposition. In Sophie Tison, editor, *Trees in Algebra and Programming - CAAP'94, 19th International Colloquium, Edinburgh, U.K., April 11-13, 1994, Proceedings*, volume 787 of *Lecture Notes in Computer Science*, pages 68–84. Springer, 1994. doi:10.1007/BFb0017474.
10. Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
11. Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.
12. Vladimir Estivill-Castro and Derick Wood. A survey of adaptive sorting algorithms. *ACM Comput. Surv.*, 24(4):441–476, 1992. doi:10.1145/146370.146381.
13. Till Fluschnik, Christian Komusiewicz, George B. Mertzios, André Nichterlein, Rolf Niedermeier, and Nimrod Talmon. When can graph hyperbolicity be computed in linear time? In *Algorithms and Data Structures - 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017, Proceedings*, pages 397–408, 2017. doi:10.1007/978-3-319-62127-2_34.
14. 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 *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on*

Discrete Algorithms, pages 1419–1432. Society for Industrial and Applied Mathematics, 2017.

15 Harold N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *J. Comput. Syst. Sci.*, 50(2):259–273, 1995. doi:10.1006/jcss.1995.1022.

16 Harold N. Gabow. Data structures for weighted matching and extensions to b-matching and f-factors. *CoRR*, abs/1611.07541, 2016. arXiv:1611.07541.

17 François Le Gall. Powers of tensors and fast matrix multiplication. In Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303. ACM, 2014. URL: http://dl.acm.org/citation.cfm?id=2608628, doi:10.1145/2608628.2608664.

18 Tibor Gallai. Transitiv orientierbare graphen. *Acta Mathematica Hungarica*, 18(1-2):25–66, 1967.

19 Archontia C. Giannopoulou, George B. Mertzios, and Rolf Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *CoRR*, abs/1506.01652, 2015. arXiv:1506.01652.

20 Andrew V. Goldberg and Satish Rao. Flows in undirected unit capacity networks. *SIAM J. Discrete Math.*, 12(1):1–5, 1999. doi:10.1137/S089548019733103X.

21 Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010. doi:10.1016/j.cosrev.2010.01.001.

22 Jianxiu Hao and James B. Orlin. A faster algorithm for finding the minimum cut in a directed graph. *J. Algorithms*, 17(3):424–446, 1994. doi:10.1006/jagm.1994.1043.

23 Thore Husfeldt. Computing graph distances parameterized by treewidth and diameter. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPIcs*, pages 16:1–16:11. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.16.

24 Yoichi Iwata, Tomoaki Ogasawara, and Naoto Ohsaka. On the power of tree-depth for fully polynomial FPT algorithms. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, pages 41:1–41:14, 2018. doi:10.4230/LIPIcs.STACS.2018.41.

25 David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000. doi:10.1145/331605.331608.

26 Stefan Kratsch and Florian Nelles. Efficient and adaptive parameterized algorithms on modular decompositions. *CoRR*, abs/1804.10173, 2018. arXiv:1804.10173.

27 Ross M McConnell and Jeremy P Spinrad. Linear-time modular decomposition and efficient transitive orientation of comparability graphs. In *Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 536–545. Society for Industrial and Applied Mathematics, 1994.

28 George B. Mertzios, André Nichterlein, and Rolf Niedermeier. Fine-grained algorithm design for matching. *CoRR*, abs/1609.08879, 2016. arXiv:1609.08879.

29 Silvio Micali and Vijay V. Vazirani. An o(sqrt(|v|) |e|) algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 17–27. IEEE Computer Society, 1980. doi:10.1109/SFCS.1980.12.

30 James B. Orlin. Max flows in o(nm) time, or better. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 765–774, 2013. doi:10.1145/2488608.2488705.

31 Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1065–1075. SIAM, 2010. doi:10.1137/1.9781611973075.86.

32 Thomas Schank and Dorothea Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In *Experimental and Efficient Algorithms, 4th InternationalWorkshop, WEA 2005, Santorini Island, Greece, May 10-13, 2005, Proceedings*, pages 606–609, 2005. doi:10.1007/11427186_54.

33 Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *J. ACM*, 44(4):585–591, 1997. doi:10.1145/263867.263872.

34 Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 634–645. Springer, 2008. doi:10.1007/978-3-540-70575-8_52.