Maximum Matching in Almost Linear Time on Graphs of Bounded Clique-Width

Guillaume Ducoffe

Maximum Matching in almost linear time on graphs of bounded clique-width *

Guillaume Ducoffe
guillaume.ducoffe@ici.ro[1,2]

[1]National Institute for Research and Development in Informatics, Romania
[2]University of Bucharest, Romania

Abstract

Recently, independent groups of researchers have presented algorithms to compute a maximum matching in $\tilde{\mathcal{O}}(f(k)\cdot(n+m))$ time, for some computable function f, within the graphs where some clique-width upper bound is at most k (*e.g.*, tree-width, modular-width and P_4-sparseness). However, to the best of our knowledge, the existence of such algorithm within the graphs of bounded clique-width has remained open until this paper. Indeed, we cannot even apply Courcelle's theorem to this problem directly, because a matching cannot be expressed in MSO_1 logic.

Our first contribution is an almost linear-time algorithm to compute a maximum matching in any bounded clique-width graph, being given a corresponding clique-width expression. We also present how to compute the Edmonds-Gallai decomposition in almost linear time by using the same framework. For that, we do apply Courcelle's theorem but to the classic Tutte-Berge formula, that can easily be expressed as a $CMSO_1$ optimization problem. Doing so, we can compute the cardinality of a maximum matching, but not the matching itself. To obtain with this approach a maximum matching, we need to combine it with a recursive dissection scheme for bounded clique-width graphs and with a distributed version of Courcelle's theorem (Courcelle and Vanicat, *DAM 2016*) – of which we present here a slightly stronger version than the standard one in the literature.

Finally, for the bipartite graphs of clique-width at most k, we present an alternative $\tilde{\mathcal{O}}(k^2\cdot(n+m))$-time algorithm for the problem. The algorithm is randomized and it is based on a completely different approach than above: combining various reductions to matching and flow problems on bounded tree-width graphs with a very recent result on the parameterized complexity of linear programming (Dong et. al., *STOC'21*). Our results for bounded clique-width graphs extend many prior works on the complexity of Maximum Matching within cographs, distance-hereditary graphs, series-parallel graphs and other subclasses.

*Results of this paper were partially presented at the IPEC'21 conference [28].

1 Introduction

For any undefined graph terminology, see [7, 24]. Throughout the paper, for any graph $G = (V, E)$, let $n := |V|$ be its *order* (number of vertices) and $m := |E|$ be its *size* (number of edges). Recall that a *matching* in a graph is a set of pairwise end-disjoint edges. A *maximum matching* is one of maximum cardinality. The *matching number* of G, denoted by $\nu(G)$, is the cardinality of a maximum matching of G. Matchings (possibly, with additional constraints) are ubiquitous in scheduling, markets, resource allocation schemes and even chemistry [73]. We refer to [47, 59] for a compendium of matching problems and their applications. This paper is about the (parameterized) complexity of Maximum Matching in graphs. Unsurprisingly, a lot of research in Computer Science has been devoted to this question. The first polynomial-time algorithm for Maximum Matching was proposed by Edmonds [32]. Later, Micali and Vazirani presented the state-of-the-art $\mathcal{O}(m\sqrt{n})$-time algorithm for this problem [64], that has remained unchallenged since forty years. We study Maximum Matching in the context of "Fine-Grained Complexity in P" (see [77] for a survey of this blossoming field). Specifically, *can a maximum matching be computed in (almost) linear time?* There are a few reasons to believe that it is indeed the case. For instance, unlike for the diameter problem and other fundamental graph problems, for which over the last decades, conditional superlinear lower bounds were obtained, it is known [8] that proving such lower bound for Maximum Matching would falsify the so-called Nondeterministic Strong Exponential Time Hypothesis (NSETH). Furthermore, computing a maximum matching is related to Maximum Flow [76], that is sometimes conjectured to be solvable in linear time.

The idea of using tools and concepts from parameterized complexity in the context of polynomial-time solvable problems has been scarce [51]. In part motivated by the recent "SETH-hardness" results, and other conditional lower bounds for such problems [78], a richer theory of "FPT in P" has started to emerge recently [1, 5, 34, 39, 48]. In its simplest form, the former is about the existence of $\mathcal{O}(f(k)\cdot(n+m)^{1+o(1)})$-time algorithms for various graph problems when some fixed parameter is at most k [1]. As far as we are concerned here, such running times were obtained in [13, 31, 30, 39, 48, 52, 53, 55, 63] for Maximum Matching, for different parameterizations. For instance, for the graphs of *tree-width* at most k, Fomin et. al [39] presented a randomized $\tilde{\mathcal{O}}(k^4 \cdot n)$-time algorithm for computing a maximum matching [2]. This was later improved by Iwata et. al. [53], who designed a deterministic $\tilde{\mathcal{O}}(k^2 \cdot n)$-time algorithm for that problem. – We recall the definition of tree-width in Sec. 2.2. – Remarkably, the parameterized study of Maximum Matching has led to the development of many nice techniques in this area, which brought Mertzios et. al. [63] to nickname the problem the "drosophilia" of the study of the FPT algorithms in P.

Clique-width is one of the most studied graph parameters. It is a rough estimate of the closeness of a graph to a cograph (*a.k.a.*, P_4-free graph). We refer to Sec. 2.1 for a formal definition. Note that unlike for the tree-width, there exist dense graphs of bounded clique-width (*e.g.*, the complete graphs and the complete bipartite graphs). The applications of clique-width to NP-hard problems, including Courcelle's theorem [17] and some general algorithmic frameworks [33], are now rather well understood [36, 37, 38]. However, the study of its applications to polynomial-time solvable problems is comparatively much more recent and, so far, limited to cycle problems [5, 13] and distance problems [13, 20, 29, 56]. Parameterized almost linear-time algorithms for Maximum Matching are known for the important subclasses of bounded tree-width graphs [39, 53], graphs of bounded modular-width [13, 55], and some others [13, 30, 31]. However, as far as we know, the complexity of this problem on bounded clique-width graphs has been open until this article. Indeed we stress that, even allowing a super-polynomial dependency on the clique-width in the running time, the existence of an almost linear-time (parameterized) algorithm for Maximum Matching does not follow from Courcelle's theorem, because a matching cannot be expressed in MSO_1 logic. This is in sharp contrast with bounded tree-width graphs, for which we can apply Courcelle's theorem for the stronger MSO_2 logic (allowing quantification over subsets of edges), and so, in particular in order to express a matching [14]. – We refer to Sec. 3 for a reminder about MSO logic. – Furthermore if we consider the related problem Maximum-Weight Matching, then it has been observed [55] that it is as hard on bounded clique-width

[1] More generally, the goal is, for some problem solvable in $\mathcal{O}(m^{q+o(1)})$ time on arbitrary m-edge graphs, to design an $\mathcal{O}(f(k)\cdot m^{p+o(1)})$-time algorithm, for some $p < q$, within the class of graphs where some fixed parameter is at most k.

[2] The $\tilde{\mathcal{O}}()$ notation suppresses poly-logarithmic factors.

weighted graphs as on general weighted graphs under $\mathcal{O}(n^2)$-time reductions. Again, this differs from the case of bounded tree-width graphs, for which an $\tilde{\mathcal{O}}(k^2 n)$-time algorithm also exists for this problem [53].

Beyond the study of the FPT algorithms in P, it also makes sense to study Maximum Matching on restricted graph classes, both as a way to better understand the hard instances for this problem, and to better model some of its real-life applications (see [49] for an example of the latter). In this respect, a considerable amount of positive results have been proved [9, 23, 27, 41, 40, 49, 58, 62, 65, 79, 81, 80]. Many such classes, starting from the cographs [19], are known to have bounded clique-width. Therefore, having at hands an almost linear-time algorithm for Maximum Matching on bounded clique-width graphs, one can unify and generalize many prior works in this area.

1.1 Our results

Recall that a graph has clique-width at most k if and only if it admits a *k-expression* [19]. Such a k-expression can be computed in linear time on many interesting subclasses of bounded clique-width graphs: ranging from cographs [19], switched cographs [11], distance-hereditary graphs [50], $(q, q-3)$-graphs [61], and graphs of either bounded tree-width [12], modular-width [19] or split-width [70]. Until recently, the best-known approximation algorithms for clique-width on general graphs had a running-time in $\mathcal{O}(n^3)$, that is slower than the state-of-the-art algorithm for Maximum Matching [67]. However, this has been recently improved to $\mathcal{O}(n^2)$ for constant clique-width graphs [35].

Hereafter, we use the notation $\tilde{\mathcal{O}}_{x_1,x_2,\ldots,x_t}(n+m)$ for a running time in $\tilde{\mathcal{O}}(f(x_1, x_2, \ldots, x_t) \cdot (n+m))$, for some computable function f. The following theorem is our first main result in the paper:

Theorem 1. *Given a graph G and a corresponding k-expression, one can compute a maximum matching for G in deterministic $\tilde{\mathcal{O}}_k(n+m)$ time.*

To the best of our knowledge, this is the first almost linear-time algorithm for Maximum Matching on bounded clique-width graphs. The $\tilde{\mathcal{O}}_k()$ notation hides huge constants in k due to our use of Courcelle's theorem. Indeed, while we cannot express a matching in MSO_1 logic, we can write a *Counting* MSO_1 formula in order to evaluate the matching number (Theorem 5). For that, we use the well-known *Tutte-Berge* formula [6]. This alone does not lead to an efficient computation of a maximum matching, but only of its size. However, by carefully evaluating our formula for the matching number on various subgraphs, obtained by removing specific vertex- and edge-subsets, one can compute a maximum matching incrementally. A similar approach also works for computing the *Edmonds-Gallai decomposition* [32, 45, 46], which encodes the structure of all the maximum matchings in a graph (Theorem 7). The main difficulty here is that the number of subgraphs on which we need to evaluate our formula can be linear in the size of the graph. Thus, applying Courcelle's theorem to each subgraph separately would result in a quadratic running time. We overcome this issue by using a distributed version of this theorem [21]. In doing so, after we computed the matching number of a bounded clique-width graph G in linear time, it becomes possible to evaluate our formula on any subgraph H in time roughly proportional to the number of basic operations needed to obtain H from G.

It seems that improving the dependency on k in the running time will require new techniques. Our second main result is that it can be done for *bipartite* graphs of bounded clique-width:

Theorem 2. *Given a bipartite graph G and a corresponding k-expression, one can compute a maximum matching for G in randomized $\tilde{\mathcal{O}}(k^2 \cdot (n+m))$ time.*

Let us sketch below the main lines of our approach toward proving Theorem 2. We first reduce Maximum Matching on bounded clique-width graphs to a related problem on the graphs of bounded *tree-width*. The reduction preserves the property for a graph to be bipartite. Its intuition goes as follows. Roughly, graphs of bounded tree-width can be recursively disconnected by some small balanced vertex-separators. By comparison, graphs of bounded clique-width can be recursively disconnected by some balanced edge-cuts of small "neighbourhood diversity" (partitionable in a small number of complete joins) [16]. To reduce to the bounded tree-width case, we propose a transformation of edge-cuts of small neighbourhood diversity into

small vertex-separators (Sec. 5.1). The transformation forces us to deal with a more general problem than Maximum Matching, sometimes called Maximum b-Matching and well-studied on its own [4, 43, 44, 60, 68, 69]. We thus exchange Maximum Matching for a more complex problem, but on a structurally simpler graph class. Furthermore, because we restrict ourselves to bipartite graphs, we can solve Maximum b-Matching as a linear program. To the matrix representation of any such linear program, one can associate various graphs. Then, it becomes possible to define the tree-width of a linear program. In [39], Fomin et. al. asked whether all linear programs of bounded tree-width could be solved in almost linear time. Very recently, Dong et. al. gave a positive answer [26]. – This is where we need randomization. – We apply this nice result to the problem Maximum b-Matching within bipartite graphs. Here, some final technicalities arise due to the algorithm of Dong et. al. only outputting an approximate fractional b-matching whereas we aim at computing an exact integral solution. This can be overcome by using the close connection between Maximum Flow and Maximum b-Matching on bipartite graphs, along with a nice result from Madry to apply rounding to a fractional flow [60].

1.2 Related work

There are several meta-theorems deduced from Courcelle's theorem in the literature. Indeed, Courcelle's approach not only applies to decision problems, but also to counting [15] and optimization problems [17]. We actually use in our proof the optimization version of his theorem. In [18], Courcelle and Mosbah designed a very general framework in order to evaluate some function over the satisfying set of a $CMSO_1$ formula. It is unclear whether we could express a maximum matching within their framework as the result of some suitable evaluation over the certificates that satisfy the Tutte-Berge formula (at the very least, we cannot do so by using the evaluation structures presented in [18, Sec. 4]). Applications to the design of distance-labelling schemes were proposed in [21], and later refined in [20, 29] using alternative techniques. However, insofar most applications of Courcelle's theorem are about NP-hard problems. Indeed, Abboud et. al. [1] observed that its use leads to huge dependencies on the parameter involved, that can often be sharpened by preferring other techniques (their observation, on the other hand, also remains valid for NP-hard problems). What we find intriguing in our case is, first, the nontrivial use we need to make of Courcelle's theorem for a polynomial-time solvable problem, and second, that we currently do not see any other way to obtain a quasi linear-time algorithm for Maximum Matching on the bounded clique-width graphs. This is evidence, we believe, that Courcelle's theorem could help in expanding the nascent field of "FPT in P".

The proof of our Theorem 1 also has several aspects that, we think, are equally intriguing. For one, we avoid computing augmenting paths, and we do not need the Tutte matrix [74] either. Both concepts are the cornerstone of almost all maximum matching algorithms in the literature. At the core of our approach is a new algorithmic application of the Tutte-Berge formula. The latter also got used in [8], but the algorithm in this related work was non-deterministic. Our repeated use of this formula in order to compute a maximum matching is not unlike the celebrated result of Anari and Vazirani that reduces the efficient parallel computation of such matching to the design of an oracle for a decision version of the problem [3]. Nevertheless, both results have fairly different proofs.

About Theorem 2, we note that different reductions from Maximum Matching to Maximum b-Matching have already been considered for graphs of bounded modular-width [55] or bounded split-width [30], that are subclasses of bounded clique-width graphs. However, from the algorithmic point of view, the instances of Maximum b-Matching outputted by these former reductions are of bounded size, a much more restricted case than bounded tree-width. To our best knowledge, the Maximum b-Matching problem has only been solved in almost linear time on subclasses of graphs of tree-width at most two [31]. We left open the parameterized complexity of Maximum b-Matching within the graphs of bounded tree-width. For general graphs, the so-called "Russian method" starts from the linear relaxation of this problem (written as an integer program) and it repeatedly adds "blossom constraints" that are violated by the current solution until it finds an optimal integral outcome [68]. These blossom constraints are deduced from the good characterization of the b-matching polytope by Edmonds and Pulleyblank [69]. It seems, however, that a super-linear (but polynomial) number of linear programs needs to be solved on general graphs. See also Anstee [4] and

Gabow [43] for alternative algorithms.

1.3 Organization of the paper

In Sec. 2, we introduce some basic notations and terminology, as well as the two graph parameters considered in this article. Sec. 3 is devoted to Courcelle's theorem for bounded clique-width graphs. We need a distributed version of this theorem, for optimization functions, of which a proof by Courcelle and Vanicat can be found in [21] but, unfortunately, for a more restricted setting than what we need. While it is not excessively difficult to check that Courcelle and Vanicat's proof indeed works in the broader setting that is here needed, the proof is fairly long and it has several intermediate steps, which is why we found it better to write it down almost completely. Then, in Sec. 4, we apply this result in order to compute the matching number, a corresponding maximum matching, and the Edmonds-Gallai decomposition, within bounded clique-width graphs. Sec. 5 is devoted to the proof of Theorem 2. We then conclude in Sec. 6.

2 Preliminaries

First we complete the basic graph terminology given in Sec. 1. By a graph, we mean a finite, simple, unweighted undirected graph. Let $G = (V, E)$ be such a graph. The (open) *neighbourhood* of a vertex v is defined as $N_G(v) := \{u \in V \mid uv \in E\}$. Its *closed neighbourhood* is defined as $N_G[v] := N_G(v) \cup \{v\}$. Let $d_G(v) := |N_G(v)|$ be the *degree* of vertex v. Similarly, for a vertex-subset S, let $N_G(S) := \bigcup_{v \in S} N_G(v) \setminus S$ denote its (open) neighbourhood and let $N_G[S] := S \cup N_G(S)$ denote its closed neighbourhood. We sometimes omit the subscript if the graph G is clear from the context.

2.1 Clique-width

A *k-labeled graph* is a graph $G = (V, E)$ equipped with a labeling function $\ell : V \to \{1, 2, \ldots, k\}$. A k-expression is an algebraic expression where the four allowed operations are:

- $i(v)$: we add a new isolated vertex with label $\ell(v) = i$;
- $G_1 \oplus G_2$: we make the disjoint union of two k-labeled graphs;
- $\eta_{i,j}$: for some distinct labels i and j, we add an edge uv for each pair (u, v) of a vertex u of label i and a vertex v of label j. Doing so, we create a *join* (complete bipartite subgraph) whose respective sides are the subset of vertices of label i and the subset of vertices of label j.
- $\rho_{i \to j}$: for some distinct labels i and j, for all vertices v s.t. $\ell(v) = i$, we set $\ell(v) = j$.

The *generated graph* is the one obtained from the k-expression by ignoring all the labels. The clique-width of a graph G, denoted by $cw(G)$, is the least k such that it is the graph generated by some k-expression [19].

It is useful to see a k-expression as a rooted tree. Namely, the leaves of this tree are labeled by the operations $i(v)$ for vertex-creation. The internal nodes are labeled by the other operations, with the degree of each such node being the arity of the corresponding operation: 1 for the join operations and the relabeling operations, and 2 for the disjoint union operations. See Fig. 1 for an example. We call this tree representation the *parse tree* of the k-expression.

The size of a k-expression is its number of operations (= number of nodes in its parse tree). Throughout the remainder of the paper, we assume each given k-expression for a graph to be of linear size $\mathcal{O}(n + m)$. Note that it is always the case if we restrict ourselves to a subclass where a k-expression can be computed in linear time, that is anyway the most relevant case for which our results in this paper could be applied. More generally, any k-expression can be transformed into an equivalent k-expression of size $\mathcal{O}(n + m)$ [42]. This transformation can be done in time $\mathcal{O}(L)$ where L denotes the size of the input k-expression (see Lemma 7 in the paper). In [17], the authors remarked that any k-expression of size L can be transformed in $\mathcal{O}_k(L)$ time into an equivalent k-expression of size $\mathcal{O}_k(n)$. However, they left unspecified the dependency on k, both in the runtime of this transformation and in the size of its output.

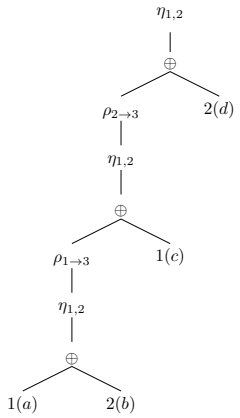

Figure 1: The parse tree of some 3-expression of P_4.

2.2 Tree-width

A *tree decomposition* $(T, \mathcal{X})$ of $G = (V, E)$ is a pair consisting of a tree T and of a family $\mathcal{X} = (X_t)_{t \in V(T)}$ of subsets of V indexed by the nodes of T and satisfying:

- $\bigcup_{t \in V(T)} X_t = V$;
- for any edge $e = \{u, v\} \in E$, there exists $t \in V(T)$ such that $u, v \in X_t$;
- for any $v \in V$, the set of nodes $\{t \in V(T) \mid v \in X_t\}$ induces a subtree T_v of T.

The sets X_t are called *the bags* of the decomposition. The *width* of a tree decomposition is the size of a largest bag minus one. Finally, the *tree-width* of a graph G, denoted by $\mathsf{tw}(G)$, is the least possible width over its tree decompositions. We only use tree-width in Sec. 5.

3 Courcelle's theorem

We refer to [15] for a thorough treatment of graph logics and their algorithmic applications. Recall that in *monadic second-order logic* (for short, MSO logic), we are given first-order variables x (written in lower-case), and set variables X (written in upper-case). We allow atomic formulas of the form $x \in X$, expressing the membership of x to a set X. The *counting* MSO (for short, $CMSO$) further allows atomic formulas of the form $Card_{p,q}(X)$, expressing that $|X| \equiv p \pmod q$ and sometimes called *counting predicates*. – We here assume p and q to be fixed constants, but it should be noted that in the general case, the complexity of $CMSO$ model checking also depends on the values of p and q. – The $CMSO$ logic is stronger than MSO logic: for instance, there is no MSO formula expressing that a set has even cardinality [15]. In Sec. 4, we will need the counting predicates of $CMSO$ logic in order to express the Tutte-Berge formula. Before that, we need to define $CMSO$ *optimization functions*.

Let φ be some $CMSO$ formula whose set of free variables is arbitrarily split in two disjoint subsets of respective sizes r and s. Let also $a_1, a_2, \ldots, a_s$ be fixed integers. Let $Z_1, Z_2, \ldots, Z_r$ be fixed subsets of the domain of the first-order variables. We define $\psi(Z_1, Z_2, \ldots, Z_r)$ as the minimum value $\sum_{i=1}^{s} a_i \cdot |X_i|$ amongst all subsets $X_1, X_2, \ldots, X_s$ such that $\varphi(Z_1, Z_2, \ldots, Z_r, X_1, X_2, \ldots, X_s)$ is true. Then, ψ is a $CMSO$ optimization function of arity r. We define the size of ψ as $|\psi| = r + s$ (*a.k.a.*, as the arity of the underlying $CMSO$ formula φ).

To a c-labelled graph $G = (V, E, \ell)$, we can associate the *relational structure* $\langle V, \{adj, lab_1, lab_2, \ldots, lab_c\}\rangle$ where the vertex-set V is the domain of first-order variables, the binary operator $adj : V \times V \to \{0, 1\}$ asserts

whether two vertices are adjacent in G and, for each i, $lab_i : V \to \{0,1\}$ asserts whether the label of a given vertex equals i. The $(C)MSO_1$ logic on graphs is the above $(C)MSO$ logic restricted to such structures. We define $CMSO_1$ optimization functions in the exact same way. There is a more general $(C)MSO_2$ logic, where we also allow variables to represent edges, but it is not discussed here. Finally, define the underlying graph of G as the graph obtained from G by removing all its labels.

The following Theorem 3 should not be considered as completely new. Indeed, it was first proved by Courcelle and Vanicat [21], but only for MSO_1 whereas we prove it here for $CMSO_1$ logic. Another difference between the following Theorem 3 and the original result of Courcelle and Vanicat is that, in [21], the authors restricted their study to optimization functions where all but one free variables must be fixed (with our notations, the latter corresponds to the case $s = 1$). Still, their proof also applies to the more general case presented below. We note that a proof of Theorem 3 could also be deduced from the heavy machinery from [15]. Our presentation marginally differs from these previous works, while it avoids using explicitly some logic concepts such as MSO transductions.

Theorem 3. *Let ψ be a $CMSO_1$ optimization function on c-labelled graphs, for some fixed constant c, and of arity r. For every c-labelled graph G of clique-width at most k, if a k-expression is given for the underlying graph of G, then after a pre-processing in $\tilde{\mathcal{O}}_{k,|\psi|}(n+m)$ time, for every vertex-subsets $Z_1, Z_2, \ldots, Z_r$ of G, we can compute the value $\psi(Z_1, Z_2, \ldots, Z_r)$ in $\tilde{\mathcal{O}}_{k,|\psi|}\left(\sum_{j=1}^{r} |Z_j|\right)$ time.*

In order to prove Theorem 3, roughly, we rewrite a $CMSO_1$ formula on a c-labelled graph as a longer $CMSO$ formula on the parse tree of its clique-width expression. Then, we make this parse tree of logarithmic depth, using a modified centroid decomposition, updating the $CMSO$ formula along the way. We end up designing a dynamic programming procedure on this parse tree, using prior work of Doner [25] and Thatcher and Wright [72] on tree automata.

Proof of Theorem 3. We start from the given k-expression, which we transform into a kc-expression whose output is the c-labelled graph G. For that, roughly, we encode in the label of each vertex its former label in the original k-expression and its final label in G. See [17, Lemma 30]. Consider the parse tree T of the resulting expression. Let ψ be defined by a $CMSO_1$ formula φ of arity $r + s$ and by integers $a_1, a_2, \ldots, a_s$.

As an intermediate step toward proving the theorem, we now define a term as a node-labelled binary tree where the labels of the leaves are taken from some finite set of constants C and the labels of the internal nodes are taken from some some finite set F of binary functions. To each term T', we can associate a relational structure $\langle V(T'), \{left, right\} \cup \{lab_\alpha \mid \alpha \in F \cup C\}\rangle$ and define $CMSO$ formulas. Here, the node-set $V(T')$ is the domain of first-order variables, $left : V(T') \times V(T') \to \{0,1\}$ asserts whether the first given node is the left child of the second node, $right : V(T') \times V(T') \to \{0,1\}$ does the same for the right child relation, and finally, for each $\alpha \in F \cup C$, $lab_\alpha : V(T') \to \{0,1\}$ asserts whether the label of a given node equals α. In general, a parse tree T is not a term, because it is not a binary tree. Indeed, nodes labelled by either a relabelling or join operation have arity equal to one. However, to each such node, one may simply attach a new leaf with a special label equal to 0. We call T' the resulting term.

Claim 1. *Consider the structure associated to T'. The following properties can be expressed by a MSO formula for this structure (whose length depends on k and c):*

1. *whether two given vertices of G are adjacent;*
2. *whether a given vertex of G has label i, for any i.*

Proof. We first define $child(x, y) := left(x, y) \vee right(x, y)$ as a shorthand to determine whether x is a child of node y in T'. Then, we define a MSO formula $desc(x, y)$ to determine whether x is a descendant of y:

$$\begin{aligned} desc(x,y) :=& \exists P(x \in P \wedge y \in P \wedge \forall z \in P(\\ & z \neq x \iff \exists z' \in P(child(z', z)) \wedge z \neq y \iff \exists z'' \in P(child(z, z'')))) \end{aligned}$$

where P represents the path from x to y. We also define $leaf(x) := \neg(\exists z(child(z,x)))$ and $root(x) := \neg(\exists z(child(x,z)))$ to determine whether a node x is a leaf or the root of T', respectively. Now, let v be a node of T'. It is a vertex of G if it satisfies the MSO formula $vertex(v) := leaf(v) \wedge \neg lab_0(v)$. Fix an ancestor x of v. It represents a kc-subexpression. We can determine whether, in the subgraph output by this subexpression, vertex v has label j, by keeping track in some subset of nodes X of all the relabelling operations which involve vertex v. Note that all such operations label nodes of T' on the path from v to x. Let $ordered(X) := \forall y, y' \in X(desc(y,y') \vee desc(y',y))$ to determine whether all nodes in X lie on a path from a leaf to the root. Let. also $anc(X,x) = \forall z \in X(desc(z,x))$ to determine whether x is an ancestor of all nodes in X. We will need to compare every two "consecutive" operations stored in X. For that, let us define the formula:

$$\begin{aligned} succ(z,z',X) :=& z \neq z' \wedge desc(z,z') \wedge \forall z'' \in X(\\ & (z'' \neq z \wedge z'' \neq z') \Longrightarrow (\neg desc(z,z'') \vee \neg desc(z'',z'))) \end{aligned}$$

If X is a subset of nodes on the path from a leaf to the root, then it associates to a node z its closest ancestor node z' in X.

Let $relab_{in}(z,i) := \bigvee_{j=1}^{k} lab_{\rho_{i\to j}}(z)$ be satisfied if, for some j, all vertices with label i are assigned label j (we say that a relabelling operation for label i occurs at node z). In the same way, let $relab_{out}(z,j) := \left(\bigvee_{i=1}^{k} lab_{\rho_{i\to j}}(z)\right) \vee lab_j(z)$ be satisfied if either we create a new vertex with label j or, for some i, all vertices with label i are assigned label j. Informally speaking, we use $relab_{in}(z,i)$ (resp., $relab_{out}(z,j)$) in order to keep track of the label of vertex v before (resp., after) each relabelling operation; the initial creation of vertex v is abusively considered the same as a relabelling operation.

The MSO formula:

$$no - relab(z,z',i) := \forall y((desc(z,y) \wedge desc(y,z') \wedge y \neq z') \Longrightarrow \neg relab_{in}(y,i))$$

asserts that there is no relabelling operation for label i that occurs from node z (included) to z' (excluded) in T'. Let

$$next(z,z') := \bigvee_{i=1}^{k} (relab_{out}(z,i) \wedge no - relabel(z,z',i) \wedge relab_{in}(z',i))$$

to check whether at z some more vertices get assigned label i (as a result of either the creation of a new vertex, or a relabelling operation) and z' is the next relabelling operation for i after z. Informally speaking, in our final formula we use $next(z,z')$ as a way to check whether z, z' are two consecutive changes of label for vertex v. Finally, let

$$final(z,x,j) := relab_{out}(z,j) \wedge (z = x \vee (no - relab(z,x,j) \wedge \neg relab_{in}(x,j)))$$

to check that the label of vertex v stays unchanged from z. We are now ready to check the label of v in the graph outputted by the subexpression:

$$\begin{aligned} label(v,x,j) := \exists X(& v \in X \wedge ordered(X) \wedge anc(X,x) \\ & \wedge \forall z \in X(\exists z'(z' \in X \wedge succ(z,z',X)) \Longrightarrow next(z,z') \\ & \wedge \nexists z'(z' \in X \wedge succ(z,z',X)) \Longrightarrow final(z,x,j))). \end{aligned}$$

Equipped with the above formula, we can decide whether v has label i in G using the formula $lab_G(v,i) := \exists x(root(x) \wedge label(v,x,i))$. Furthermore, we can also determine whether two vertices u and v are adjacent in G by using the formula:

$$adj_G(u,v) := \bigvee_{1 \leq i,j \leq k} (\exists x(label(v,x,i) \wedge label(u,x,j) \wedge lab_{\eta_{i,j}}(x))).$$

$\diamond$

Doing so, we may now regard the $CMSO_1$ formula φ for G as a $CMSO$ formula φ' for the structure associated to T'. Let ψ' be the $CMSO$ optimization function defined from φ' and $a_1, a_2, \ldots, a_s$ in the natural way. We stress that φ' and ψ' express properties of T' and of subsets of leaves of T': indeed, there is a one-to-one mapping between the vertices of G and the leaves of T' with positive label. Furthermore, we have $|\psi'| = \mathcal{O}_{k,|\psi|}(1)$.

In order to prove the theorem, it suffices to prove the following. Let ψ' be any fixed $CMSO$ optimization function which expresses properties of terms and of subsets of their leaves. For any term T' with N nodes, we can associate $\tilde{\mathcal{O}}_{|\psi'|}(1)$-size labels $L(v)$ to each leaf-node v in such a way that, for each subsets of leaves $Z_1, Z_2, \ldots, Z_r$, it becomes possible to compute $\psi'(Z_1, Z_2, \ldots, Z_r)$ in $\mathcal{O}(\sum_{i=1}^{r} \sum_{v \in Z_i} |L(v)|)$ time. Moreover, all these labels can be pre-computed in $\tilde{\mathcal{O}}_{|\psi'|}(N)$ time.

For that, a nice intermediate result of Courcelle and Vanicat [21, Sec. 4] is that we can always assume the term to have logarithmic depth. Specifically, add one more binary operator $\circ$ to the label-set of internal nodes, and one more constant I_d to the label-set of leaves. Let T' be a term with N nodes and at most one leaf whose label equals I_d (note that initially, there is no such a leaf, and so, the property holds). Choose a node w such that:

- If there is no leaf whose label equals I_d, then w is a centroid of T (*i.e.*, a node whose removal leaves subtrees of order at most $N/2$). Such a node always exists by a classic theorem from Jordan [54].
- Otherwise, let u be the unique leaf labelled I_d. We choose w as the deepest ancestor of u with more than $N/2$ descendants.

In both cases, we can compute the node w in $\mathcal{O}(N)$ time by dynamic programming. Let T'_1 be obtained from T' by removing all strict descendants of w and relabelling w with the new constant I_d. Let. also T'_2, T'_3 be the left and right subtrees of w. We create a new term whose root is labelled by the new operator $\circ$, whose left subtree is T'_1 and whose right subtree is the subtree rooted at w (*i.e.*, with respective right-left and right-right subtrees T'_2 and T'_3). Furthermore, we may repeat this process on T'_1 and T'_2, T'_3 until all the gotten subtrees have constant-depth. We may see the overall construction as a special case of centroid decomposition. Let T'' be the resulting term.

Claim 2. *The number of recursive steps is in $\mathcal{O}(\log N)$. In particular, the depth of T'' is in $\mathcal{O}(\log N)$ and we can compute this term from T' in $\tilde{\mathcal{O}}(N)$ time.*

Proof. By the choice of w, the subtree T'_1 has order at most $N/2$, and so does at least one of T'_2, T'_3. The remaining subtree $T'_i,\ i \in \{2, 3\}$ also has order at most $N/2$ except maybe if we fall in the second case when there exists a leaf labelled I_d and this leaf is not contained in T'_i. Then, at the next recursive step, T'_i falls in the first case, and therefore, after at most two steps, all subtrees considered have order at most $N/2$. $\diamond$

Observe that there is a one-to-one mapping between the nodes of T' and the nodes of T'' with any other label than $I_d, \circ$. This mapping also preserves the labels and the property for a node to be a leaf. Moreover,

Claim 3. *Consider the structure associated to T''. Given two nodes, the property for the first one to be the left child of the second one in T' (resp., its right child) can be expressed by a constant-size MSO formula for this structure.*

Proof. For clarity, we shall write $lab'_\alpha, left', right'$ the operations in the structure associated to T', and $lab''_\beta, left'', right''$ the operations in the structure associated to T''.

Consider a leaf y whose label is I_d (representing one step of the centroid decomposition). There is a corresponding internal node x, labelled $\circ$. This node x is an ancestor of y, however after we are done with our centroid decomposition, there may also exist intermediate nodes labelled $\circ$ on the xy-path. In order to find x, we first need to observe that at any step of our recursive construction, if there exists a unique leaf y labelled I_d then, in the new term we create at this step (with root labelled $\circ$), this leaf y always ends up in the right subtree. As a result, x is the closest ancestor to y labelled $\circ$ such that y is in its

left subtree. Next, we characterize the relation between x and y with a MSO formula. As before, let $child(z, z') := left''(z, z') \vee right''(z, z')$. Let also

$$\begin{aligned} path(z, P, z') :=& (z, z' \in P) \wedge \forall u \in P(\\ & (u \neq z \iff \exists u' \in P(child(u', u))) \\ & \wedge (u \neq z' \iff \exists u' \in P(child(u, u')))) \end{aligned}$$

denote the property for P to be a path from z to one of its ancestors z'. We characterize the relation between x and y by the following MSO formula:

$$\begin{aligned} cut(x, y) :=& lab_{\circ}(x) \wedge lab_{I_d}(y) \wedge \exists P(path(y, P, x) \wedge \forall z \in P(\\ & z = x \iff (lab_{\circ}(z) \wedge \exists z' \in P(left''(z', z))))) \end{aligned}$$

where the final line of the formula enforces x to be the only node of P with label $\circ$ and its left child also in P. The original node of T' that is replaced by y is always the right child of x in T''.

Now, let u, v be nodes of T'. By symmetry, we only need to express the property whether u is the left child of v in T'. If u is the left child of v in T'' then it is also the case in T'. Otherwise, assume the left child of v in T'' to be labelled either $\circ$ or I_d (if it is not the case then, u cannot be the left child of v in T'). If the left child of v is a leaf y labelled I_d then, let x be the corresponding internal node labelled $\circ$; in order for u to be the left child of v in T', it must be the right child of x in T''. Finally, we consider the case when the left child of v is an internal node x labelled $\circ$. Observe that at each step of our recursive construction, the root of the term considered is put in the left subtree. Therefore, u is the left child of v in T' if and only if it is reached from x by always going left until the first node whose label does not equal $\circ$. Consider the following formula in order to relate the root x of some term in T'', whenever x is labeled $\circ$, to the root u of the corresponding term in T':

$$\begin{aligned} root'(u, x) :=& \exists P(path(u, P, x) \wedge \forall z \in P(\\ & (z \neq x \iff \exists z' \in P(left''(z, z'))) \\ & \wedge (z \neq u \iff lab_{\circ}(z)))). \end{aligned}$$

We can express whether u is the left child of v in T' as follows:

$$\begin{aligned} left'(u, v) :=& left''(u, v) \\ & \vee \exists x, y(left''(y, v) \wedge cut(x, y) \wedge right''(u, x)) \\ & \vee \exists x(left''(x, v) \wedge root'(u, x)). \end{aligned}$$

$\diamond$

It follows from the above claim that any $CMSO$ formula on the structure associated to T' can be transformed into an equivalent $CMSO$ formula on the structure associated to T'', with constant-size overhead. Thus, from now on, we assume without loss of generality (w.l.o.g). the depth of T' to be in $\mathcal{O}(\log N)$.

Let φ' be a MSO formula with $r + s$ free variables, expressing properties of terms and subsets of their leaves. A celebrated result from Doner [25] and Thatcher and Wright [72] is that for every fixed φ', for every term T' and leaf-subsets $Z_1, Z_2, \ldots, Z_{r+s}$, we can decide whether $\varphi'(Z_1, Z_2, \ldots, Z_{r+s})$ holds true in time linear in the input. – Formally, these prior works only apply to terms, and not to terms and subsets of their leaves. However, as observed by Courcelle and Vanicat [21], we may multiply the number of labels by 2^{r+s} in order to encode, for each leaf, its belonging to some subset Z_i. – This result also holds for $CMSO$ formulas because, for terms and more generally for bounded-degree forests, $CMSO$ has the same expressive power as MSO [15, Lemma 5.27 and Proposition 5.30]. We apply this result to the formula φ' corresponding to our $CMSO$ optimization function ψ' on terms. Recall that $|\psi'| = \mathcal{O}_{k,|\psi|}(1)$, where ψ is a $CMSO_1$ optimization function on c-labelled graphs.

The result of Doner, Thatcher and Wright is achieved through the construction of a finite *tree automaton* $\mathcal{A}_{\varphi'}$, of which the formal definition can be found, *e.g.*, in [15, Definition 3.46]. Suffice it to say for our purpose

that $\mathcal{A}_{\varphi'}$ is a finite-state machine which, given $T', Z_1, Z_2, \ldots, Z_{r+s}$ as input, assigns a state to each node of T' using bottom-up dynamic programming. Specifically, the state of a node is determined according to transition rules, in function of its label and of the states of its children. We say that $\mathcal{A}_{\varphi'}$ accepts the input if the state of the root is from a pre-defined set of so-called accepting states.

Note that on any input to $\mathcal{A}_{\varphi'}$, since the state of each node is computed bottom-up, it only depends on the leaves in its rooted subtree. We denote by F_x the leaves in the subtree rooted at a node x. Then, for every node x of T' and state q of $\mathcal{A}_{\varphi'}$, let $M(x,q)$ minimize $\sum_{i=1}^{s} a_i|X_i|$ amongst all subsets $X_1, X_2, \ldots, X_s \subseteq F_x$ such that, being given $T', \emptyset, \emptyset, \ldots, \emptyset, X_1, X_2, \ldots, X_s$ as input, the state assigned to node x is q (*i.e.*, we first assume $Z_1 = Z_2 = \ldots = Z_r = \emptyset$ and we restrict the other sets to F_x). If x is a leaf then, each X_i is either empty or reduced to $\{x\}$, and therefore, we are left testing $2^s = \mathcal{O}_{|\psi'|}(1)$ possibilities. Otherwise, let ℓ be the label of node x and let y, z be its children; then, by linearity, $M(x,q)$ is the minimum of $M(y,q_1) + M(z,q_2)$ amongst all transitions $(q_1, q_2, \ell) \to q$. Overall, since there are only $\mathcal{O}_{|\psi'|}(1)$ states q to consider at each node, the whole dynamic programming can be done in $\mathcal{O}_{|\psi'|}(N)$ time, where N is the number of nodes.

To each leaf v, we assign a label $L(v)$ that contains: the automaton $\mathcal{A}_{\varphi'}$, the path P_v from v to the root in T', and finally, for each $x \in P_v \setminus v$, all the values $M(y,q)$ for its unique child not in P_v. Since the depth of T' is assumed to be $\mathcal{O}(\log N)$, $|L(v)| = \tilde{\mathcal{O}}_{|\psi'|}(1)$. Let now $Z_1, Z_2, \ldots, Z_r$ be arbitrary fixed subsets. In order to compute $\psi'(Z_1, Z_2, \ldots, Z_r)$, the same as above we could compute the minimal values $M_Z(x,q)$ for each node x and state q, then keep the minimum such value when x is the root and q is an accepting state. However, that would require $\mathcal{O}_{|\psi'|}(N)$ time. To do that faster, let T'_Z be the smallest subtree that contains the root of T' and whose leaves are exactly the nodes in $Z = \bigcup_{j=1}^{r} Z_j$. Clearly, $|V(T'_Z)| \leq \sum_{v \in Z} |L(v)| \leq \tilde{\mathcal{O}}_{|\psi'|}(\sum_{j=1}^{r} |Z_j|)$. Furthermore, for every subtree of $T' \setminus T'_Z$, we stored the values $M(y,q)$ for its root y in at least one label $L(v)$, for some $v \in Z$. Note also that $M(y,q) = M_Z(y,q)$ because there is no leaf from Z in its rooted subtree. Hence, using these labels, we can compute the values $M_Z(x,q)$ for each state q and for each node x of T'_Z, including the root, by dynamic programming on T'_Z. It takes $\mathcal{O}_{|\psi'|}(|V(T'_Z)|) = \mathcal{O}_{|\psi'|}(\sum_{j=1}^{r} \sum_{v \in Z_j} |L(v)|) = \tilde{\mathcal{O}}_{|\psi'|}(\sum_{j=1}^{r} |Z_j|)$ time. □

As a particular case of the above Theorem 3 (for $r = 0$), we retrieve the optimization version of Courcelle's theorem for bounded clique-width graphs (see [17]). In this special case, the runtime can be improved to linear time because we needn't compute a modified centroid decomposition. Namely:

Theorem 4. *Let φ be a $CMSO_1$ formula on c-labelled graphs, for some fixed constant c, and with s free variables. Let also $a_1, a_2, \ldots, a_s$ be fixed integers. For every c-labelled graph G of clique-width at most k, if a k-expression is given for the underlying graph of G, then in $\mathcal{O}_{k,|\varphi|}(n+m)$ time, one can compute the minimum value $\sum_{i=1}^{s} a_i \cdot |X_i|$ amongst all vertex-subsets $X_1, X_2, \ldots, X_s$ such that $\varphi(X_1, X_2, \ldots, X_s)$ is true.*

4 Algorithms: the general case

Our main result in this section (Theorem 1) is proved in Sec. 4.3. In Sec. 4.1 we first compute the matching number, a key step toward the final proof of Theorem 1. Sec. 4.2 is devoted to computing the Edmonds-Gallai decomposition, and it is a gentle introduction to the techniques we also use in Sec. 4.3.

4.1 Size of a maximum matching

We explain in this section how to compute the matching number of bounded clique-width graphs. For that, we need a classic result from Matching theory:

Lemma 1 (Tutte-Berge formula [6, 7])**.** *For any graph $G = (V, E)$, we have:*

$$\nu(G) = \min_{U \subseteq V} \frac{1}{2} \left(|V| + |U| - odd(G \setminus U) \right)$$

where $odd(G \setminus U)$ denotes the number of connected components of odd size of $G \setminus U$.

Our main insight below is that evaluating the Tutte-Berge formula can be written as a $CMSO_1$ optimization problem. We prove it next:

Theorem 5. *For any graph $G = (V, E)$ of clique-width at most k, if a k-expression is given then, we can compute $\nu(G)$ in $\mathcal{O}_k(n + m)$ time.*

Proof. By Theorem 4, it suffices to prove that the Tutte-Berge formula (see Lemma 1) can be written as a $CMSO_1$ optimization problem. For that, let us first define $adj(x, y, U) := adj(x, y) \wedge x \notin U \wedge y \notin U$ in order to suppress all edges incident to a given set U. The following formula can be used to test whether two vertices are in the same connected component of $G \setminus U$, for a given set U: $connected(x, y, U) := \forall X((x \in X \wedge y \notin X) \implies \exists x', y'(x' \in X \wedge y' \notin X \wedge adj(x', y', U)))$. Then, we can relate a vertex to its connected component of $G \setminus U$ as follows: $comp(x, X, U) := \forall y(y \in X \iff connected(x, y, U))$. We are now ready to define our formula for computing $\nu(G)$. It has two free variables.

$$\begin{aligned} TutteBerge(U, W) :=& \forall x \in W(x \notin U \wedge \exists X(comp(x, X, U) \wedge Card_{1,2}(X))) \\ & \wedge \forall x, y \in W(x = y \vee \neg connected(x, y, U)). \end{aligned}$$

This above formula expresses that all vertices of W are in pairwise different odd components of $G \setminus U$. The first line ensures that every vertex of W is in an odd component of $G \setminus U$. The second line ensures that two distinct vertices of W are in different components of $G \setminus U$. If we set $a_1 = 1, a_2 = -1$, the objective becomes to minimize $|U| - |W|$. Therefore, we get as solution $\delta = \min_{U \subseteq V} (|U| - odd(G \setminus U))$. By Lemma 1, we have $\nu(G) = \frac{1}{2}(n + \delta)$. □

We stress that by using Courcelle's optimization theorem, an optimal certificate U for the Tutte-Berge formula could also be computed. However, this approach does not lead to an efficient computation of a maximum matching. For all that, the $CMSO_1$ formula in this above Theorem 5 is the cornerstone of all the remainder of Sec. 4.

4.2 Edmonds-Gallai decomposition

We continue with a known structural result about maximum matchings in a graph. Recall that a graph is hypomatchable if the removal of any one vertex results in a graph with a perfect matching.

Theorem 6 (Edmonds-Gallai [32, 45, 46])**.** *Let $G = (V, E)$ be a graph, and let $A \subseteq V$ be the set of all vertices v so that there is a maximum matching of G that does not cover v. Set $B = N_G(A)$ and $C = V \setminus (A \cup B)$. Then:*

- *Every odd component H of $G \setminus B$ is hypomatchable and it has $V(H) \subseteq A$;*
- *Every even component H of $G \setminus B$ has a perfect matching and it has $V(H) \subseteq C$;*
- *For every non-empty $X \subseteq B$, the set $N(X)$ contains vertices in $> |X|$ odd components of $G \setminus B$.*

The partition (A, B, C) is called the Edmonds-Gallai decomposition *of G.*

In [10], the author proposes a randomized $\mathcal{O}(n^\omega)$-time algorithm for computing the Edmonds-Gallai decomposition of an n-vertex graph, where $\omega < 2.37286$ [2] denotes the exponent of square matrix multiplication. We improve this result to deterministic almost linear-time for all classes of bounded clique-width graphs (under the standard assumption in the field that a corresponding clique-width expression is given in the input):

Theorem 7. *For any graph $G = (V, E)$ of clique-width at most k, if a k-expression is given then, we can compute its Edmonds-Gallai decomposition in $\tilde{\mathcal{O}}_k(n + m)$ time.*

Proof. If we are given the set A of all vertices left exposed by at least one maximum matching then, by Theorem 6, the sets B and C can be computed in additional $\mathcal{O}(n+m)$ time. Recall (see Theorem 5) that there exists a $CMSO_1$ formula $TutteBerge(U, W)$ to express that all vertices of W are in pairwise different odd components of $G \setminus U$. Let $EdmondsGallai(X, U, W) := TutteBerge(U \cup X, W)$. It is also a $CMSO_1$ formula since the union of two subsets can be easily expressed in MSO [15]. Then, for any X, let $\psi(X)$ be the problem of minimizing $|U| - |W|$ among all the subsets U, W such that $EdmondsGallai(X, U, W)$ is true. Observe that ψ is a $CMSO_1$ optimization function. We apply Theorem 3 to ψ. Then, we claim that $v \in A$ if and only if $\psi(\{v\}) = 2\nu(G) + 1 - n$. Indeed, by construction we have $\psi(\{v\}) = \min_{U \subseteq V \setminus \{v\}}(|U| - odd(G \setminus (\{v\} \cup U)))$, and therefore by Lemma 1, $\nu(G \setminus \{v\}) = \frac{1}{2}(n - 1 + \psi(\{v\}))$. Then:

$$\begin{aligned} v \in A \iff & \nu(G) = \nu(G \setminus \{v\}) \iff \nu(G) = \frac{1}{2}(n - 1 + \psi(\{v\})) \\ \iff & 2\nu(G) = n - 1 + \psi(\{v\}) \iff \psi(\{v\}) = 2\nu(G) + 1 - n. \end{aligned}$$

Computing $\nu(G)$ can be done in $\mathcal{O}_k(n + m)$ time (Theorem 5). Computing $\psi(\{v\})$ takes $\tilde{\mathcal{O}}_k(1)$ time per vertex v up to an $\tilde{\mathcal{O}}_k(n + m)$-time pre-processing (Theorem 3). As a result, we can compute the set A, and so, the Edmonds-Gallai decomposition, in $\tilde{\mathcal{O}}_k(n + m)$ time. □

4.3 Computation of a maximum matching

Let us first recall our main result in this section:

Theorem 1. *Given a graph G and a corresponding k-expression, one can compute a maximum matching for G in deterministic $\tilde{\mathcal{O}}_k(n + m)$ time.*

Let us sketch our strategy to prove this above result. Given a graph $G = (V, E)$, we first recall that an edge-cut is, for some non-empty proper subset A, the set of all edges between A and $V \setminus A$. It is balanced if we further have $\max\{|A|, |V \setminus A|\} \leq 2n/3$. Roughly, we compute a balanced edge-cut for G, we compute a subset of edges of the cut to be included in some maximum matching of G, then we recurse on subgraphs of $G[A]$ and $G[V \setminus A]$ separately.

Computing a balanced edge-cut

The computation of a balanced edge-cut in $\tilde{\mathcal{O}}(k \cdot (n + m))$ time follows from prior works [16, 29]. We need to introduce some additional terminology. A width-k partition tree is the parse tree of a k-expression where we iteratively contracted all internal nodes of degree two. Such trees have a purely combinatorial characterization, that can be found in [16]. A representation graph is a compact encoding of a partition tree (*i.e.*, in $\mathcal{O}(kn)$ space if the width of the partition tree is at most k). Roughly, for every node y in the partition tree, we add in the representation graph one new vertex for every non-empty label class in the labeled subgraph G_y that corresponds to its subtree. For every child z of y and for every labels i and j, we add an arc from (y, j) to (z, i) if and only if all vertices of label i in G_z are of label j in G_y.

Lemma 2 ([16])**.** *There is an algorithm that transforms a k-expression into the representation graph of a width-k partition tree in $\mathcal{O}(k \cdot (n + m))$ time.*

For a graph $G = (V, E)$, let $(U, W = V \setminus U)$ be a cut. We call its left side U an ℓ-module if it can be ℓ-partitioned into $U_1, U_2, \ldots, U_\ell$ such that, for each $1 \leq i \leq \ell$ and $u_i, u'_i \in U_i$, we have $N_G(u_i) \setminus U = N_G(u'_i) \setminus U$. Note in particular that there is a join between U_i and $N_G(U_i)$. Finally, the neighbourhood diversity of a cut (U, W) is the least ℓ such that U is an ℓ-module. An easy observation (see [29, Lemma 2]) is that for every node y in a width-k partition tree of G, the vertex-subset $V(G_y)$ is an ℓ-module where $\ell \leq k$ is the number of non-empty label classes in G_y. Based on this observation and the standard computation of a centroid node in a tree, an algorithm is proposed in the proof of [29, Theorem 2] in order to compute a balanced cut of neighbourhood diversity at most k. In a nutshell, for some centroid node c of the partition tree this algorithm starts from the non-empty label classes of G_c, as vertices in the representation graph. Then, it

computes the intersection of each label class with all leaves in the subtrees rooted at some children nodes of c. For that, it is sufficient to traverse the arcs in the representation graph until we reach all possible sinks. We summarize this above discussion in the following lemma.

Lemma 3 ([29])**.** *Let G be a graph, with some representation graph of a width-k partition tree. One can compute in $\mathcal{O}(k \cdot n)$ time a cut (U, W) s.t. $\max\{|U|, |W|\} \leq 2n/3$, and the representation graphs of some width-k partition trees for the subgraphs $G[U]$ and $G[W]$. Furthermore, U is an ℓ-module of G, for some $\ell \leq k$, and one can also compute in $\mathcal{O}(k \cdot n)$ time a corresponding ℓ-partition $U_1, U_2, \ldots, U_\ell$.*

We reuse this above Lemma 3 also in Sec. 5.1. Note that a weaker version could also be deduced from [21, Lemma 3], but at the price of a higher (exponential) dependency on the clique-width in both the construction time and the neighbourhood diversity of the output cut.

Handling of a join

An important property for the cut computed using Lemma 3 is that it can be edge-partitioned into at most k joins. We handle each join separately. For that, both Lemma 4 and Lemma 5 below apply Theorem 3 (distributed Courcelle's theorem).

Lemma 4. *Let X, Y be the two sides of a join in a graph $G = (V, E)$, where $|X| \leq |Y|$ and $cw(G) \leq k$. If a k-expression is given, then in $\tilde{\mathcal{O}}_k(n+m)$ time, we can compute an inclusion-wise minimal subset $X' \subseteq X$ such that, in some maximum matching of G:*

1. *every vertex of X' is matched to some vertex of Y;*
2. *no vertex of $X \setminus X'$ is matched to a vertex of Y.*

Proof. The idea is similar to Theorem 7 but, instead of removing subsets of vertices, we simulate the removal of some edges of the join. While we cannot express a subset of edges in $CMSO_1$ logic, such removal can be expressed in MSO_1 if we add vertex-labels. Specifically, let $G_Y = (V, E, \ell)$ be the 2-labelled graph such that: $\ell(y) = 1$ for every $y \in Y$; and $\ell(v) = 2$ for every $v \in V \setminus Y$. Through the formula

$$adj(u, v, Z) := adj(u, v) \wedge (u \notin Z \vee \neg lab_1(v)) \wedge (v \notin Z \vee \neg lab_1(u))$$

we ignore all edges between a given subset Z and Y. Take the formula $TutteBerge(U, W)$ of Theorem 5 and, for some new free variable Z, replace all occurences of $adj(u, v)$ in this formula by $adj(u, v, Z)$. Doing so, we get a new $CMSO_1$ formula $\varphi(Z, U, W)$. Let $\psi(Z)$ minimize $|U| - |W|$ among all subsets U, W such that $\varphi(Z, U, W)$ is true. If we denote $E(Z, Y)$ the set of edges between Z and Y then, by Lemma 1, we have $\nu(G \setminus E(Z, Y)) = \frac{1}{2}(n + \psi(Z))$.

We start from $Z := \emptyset$ and we consider all vertices $x \in X$ sequentially. We add x in Z if and only if $\nu(G) = \frac{1}{2}(n + \psi(Z \cup \{x\}))$. Finally, after we are done scanning X, we claim that we can choose $X' = X \setminus Z$. Indeed, by construction Z is an inclusion-wise maximal subset of X such that $\nu(G) = \nu(G \setminus E(Z, Y))$. Consider a maximum matching M with no edge between Z and Y. By maximality of Z, all the vertices of X' needs to be matched to some vertex of Y, thus proving the claim. Overall, we need to compute $\psi(Z)$ for $|X|$ different subsets Z, each of cardinality at most $|X|$. By Theorem 3, we can do all these computations in time $\tilde{\mathcal{O}}_k(n + m + |X|^2) = \tilde{\mathcal{O}}_k(n + m + |X||Y|) = \tilde{\mathcal{O}}_k(n + m)$, where the last two equalities follow from $|X| \leq |Y|$ and the fact there is a join between X and Y. □

Lemma 5. *Let X, Y be the two sides of a join in a graph $G = (V, E)$, where $|X| \leq |Y|$ and $cw(G) \leq k$. We are given a subset $X' \subseteq X$ as stated in Lemma 4. If a k-expression is given then, in $\tilde{\mathcal{O}}_k(n+m)$ time, we can compute the intersection of the edges of the join with some maximum matching of G.*

Proof. Let $G' = (V, E, \ell)$ be the 4-labelled graph such that: $\ell(x) = 1$ for every $x \in X'$; $\ell(u) = 2$ for every $u \in X \setminus X'$; $\ell(y) = 3$ for every $y \in Y$; $\ell(v) = 4$ for every $v \in V \setminus (X \cup Y)$. Consider the following formula:

$$\begin{aligned} adj(u, v, Z) := \; & adj(u, v) \\ & \wedge (lab_2(u) \implies \neg lab_3(v)) \wedge (lab_2(v) \implies \neg lab_3(u)) \\ & \wedge (u \in Z \implies lab_1(v)) \wedge (v \in Z \implies lab_1(u)). \end{aligned}$$

The second line excludes all edges between Y and $X \setminus X'$. The third line restricts to X' the neighbourhoods of the vertices in a given subset Z. As for the previous Lemma 4, we take the formula $TutteBerge(U, W)$ of Theorem 5 and, for some new free variable Z, we replace all occurences of $adj(u, v)$ in this formula by $adj(u, v, Z)$. Doing so, we get a new $CMSO_1$ formula $\varphi(Z, U, W)$. Let $\psi(Z)$ minimize $|U| - |W|$ among all subsets U, W such that $\varphi(Z, U, W)$ is true.

We start from $Z = \emptyset$ and we consider each vertex $y \in Y$ sequentially until we get $|Z| = |X'|$. We add y into Z if and only if we have $\nu(G) = \frac{1}{2}(n + \psi(Z \cup \{y\}))$. Finally, we output any perfect matching between X' and Z. Before proving correctness of the algorithm, let us discuss its time complexity. Computing $\nu(G)$ can be done in $\mathcal{O}_k(n+m)$ time by Theorem 5. We also need to evaluate $\psi(Z)$ for up to $|Y|$ different subsets Z, each having size at most $|X'| \leq |X|$. By Theorem 3, we can do that in total time $\tilde{\mathcal{O}}_k(n+m+|X||Y|) = \tilde{\mathcal{O}}_k(n+m)$, where the last equality follows from the fact there is a join between X and Y.

To prove correctness, let $Z \subseteq Y$ be of cardinality $\leq |X'|$. We claim that, if $\nu(G) = \frac{1}{2}(n + \psi(Z))$, then there exists a maximum matching of G where all the vertices of Z are matched to some vertices of X'. To see that, observe first that by Lemma 1, $\frac{1}{2}(n+\psi(Z))$ is the cardinality of a maximum matching in the graph H_Z: obtained from G by removing all edges between Y and $X \setminus X'$ and all edges between Z and $V \setminus X'$. Let M be a maximum matching of H_Z. Since $\nu(H_Z) = \nu(G)$, it is also a maximum matching of G. Assume that in M, there is a $z \in Z$ not matched to a vertex of X'. Then, z is left exposed by M (*i.e.*, because $N_{H_Z}(z) \subseteq X'$ by construction). By the pigeonhole principle, there exists a $x \in X'$ that is not matched in M to any vertex of Z. We remove from M any edge incident to x, which we replace by xz. The claim follows by repeating this process for all $z \in Z$ until each such vertex is matched to some vertex of X'.

Note that a Z as above for which we have $\nu(G) = \frac{1}{2}(n + \psi(Z))$ always exists, namely, we can set $Z = \emptyset$. Indeed, by Lemma 4 we have $\nu(G) = \nu(H_\emptyset)$. Assume now the set Z to be inclusion-wise maximal and suppose for the sake of contradiction $|Z| < |X'|$ (*i.e.*, our algorithm would stay blocked). Let M be a maximum matching of H_Z with all vertices of Z being matched to some vertices of X'. Since H_Z is a subgraph of $H_\emptyset$ and $\nu(H_Z) = \nu(G) = \nu(H_\emptyset)$, M is also a maximum matching of both $H_\emptyset$ and G. By minimality of X' (see Lemma 4) the vertices of X' must be matched to some vertices of Y in any maximum matching of $H_\emptyset$, and so, in M. Let $Z' \subseteq Y$, $Z \cap Z' = \emptyset$ such that the vertices of Y that are matched in M to the vertices of X' are exactly $Z \cup Z'$. We get $\nu(H_{Z \cup Z'}) = \nu(H_\emptyset) = \nu(G)$, thus contradicting the maximality of Z.

Overall, we may assume $|Z| = |X'|$. Since there is a join between Z and X', any perfect matching between Z and X' is included in a maximum matching M of H_Z, and so, of both G and $H_\emptyset$. Recall that all edges between Y and $X \setminus X'$ got removed in $H_\emptyset$. As a result, the intersection of M with the join is a perfect matching between Z and X'. □

We stress that if we apply Lemma 4 and Lemma 5 to two different joins X_1, Y_1 and X_2, Y_2, then doing so we may compute their respective intersections with two different maximum matchings of the graph. Therefore, we need to remove from the graph the end-vertices of all edges in the intersection between the join X_1, Y_1 considered and some maximum matching before we can process another join X_2, Y_2 of the graph. However, this is still not enough in order to ensure the compatibility of multiple applications of Lemma 4 and Lemma 5. The reason is that there may exist a maximum matching with an even *larger* intersection with the join X_1, Y_1. Indeed, the subset X'_1 of Lemma 4 is only inclusion-wise minimal. Hence, if we now apply Lemma 4 and Lemma 5 to another join X_2, Y_2, then doing so we may compute the intersection of the latter with some maximum matching with an even larger intersection with X_1, Y_1 than the one we previously computed. Our approach in order to prevent such complications from happening consists in removing all edges from the join X_1, Y_1. The resulting subgraph is not induced, but it still has bounded clique-width (see Lemma 6 below for a proof).

Updating the k-expression

Finally, once we computed from a join a subset of edges to be added into a maximum matching, all other edges of the join can be removed from the graph (and in fact they *must* be removed, as we discussed it above). We must also remove all the end-vertices of the edges included into the matching. Doing so, we need the following two lemmas in order to update the k-expression of the graph considered.

Lemma 6. *Let $G = (V, E)$ be a graph, let (U, W) be a cut of G, and let $U' \subseteq U$. If $cw(G) \leq k$ then the graph H, obtained from G by removing all edges between U' and $W' := N(U') \cap W$, has clique-width at most $3k$. Furthermore, we can compute a $3k$-expression of H from a k-expression of G in $\mathcal{O}(n + m)$ time.*

Proof. Fix a k-expression of G. We replace all operations in it as follows:

- *Creation of a vertex v with label i*: we assign to vertex v the label i if $v \in W$, the label $i + k$ if $v \in U'$, and the label $i + 2k$ otherwise.
- *Disjoint union*: unchanged.
- *Relabelling $\rho_{i \to j}$*: we perform three consecutive relabelling operations, namely, $\rho_{i \to j}$, $\rho_{i+k \to j+k}$ and $\rho_{i+2k \to j+2k}$.
- *Addition of a join $\eta_{i,j}$*: we add 7 different joins, namely, $\eta_{i,j}$, $\eta_{i,j+2k}$, $\eta_{i+k,j+k}$, $\eta_{i+k,j+2k}$, $\eta_{i+2k,j}$, $\eta_{i+2k,j+k}$, and $\eta_{i+2k,j+2k}$. Since we excluded from this above list the joins $\eta_{i,j+k}$ and $\eta_{i+k,j}$, we do not add any edge between the subsets U' and W.

Overall, we just need to scan the k-expression once and replacing each of its operations by at most seven new operations, and so, the running time is linear in its size. □

Lemma 7. *Let $G = (V, E)$ be a graph and let $H = (X, E_X)$ be an induced subgraph of G. If a k-expression of G is given, then in $\mathcal{O}(n + m)$ time, we can compute a k-expression of H of size $\mathcal{O}(|X| + |E_X|)$.*

Proof. We present a transformation of any k-expression of G into a k-expression of H of size at most $2|X| + |E_X|$. The runtime of the transformation is shown to be in $\mathcal{O}(L)$ if the k-expression of G has size L. Recall that a k-expression of size $L = \mathcal{O}(n + m)$ always exists and that we assume to be given one for our algorithms. Therefore, this aforementioned transformation will prove the lemma.

Let us call a k-expression irredundant if for any join operation $\eta_{i,j}$ in it, there was previously no edge between the vertices labelled i and the vertices labelled j. Being given a k-expression of size L, we can compute an irredundant k-expression in $\mathcal{O}(L)$ time [19]. Thus, from now on, we assume to be given an irredundant k-expression of G of size $\mathcal{O}(L)$. We transform the parse tree T of this irredundant k-expression of G into the parse tree T' of an irredundant k-expression of H. For that, we remove some nodes with at most one child from T. Note that, in what follows, whenever we remove such a node that is neither a leaf nor the root, we implicitly reconnect its unique child to its father.

The algorithm works in two phases, that we describe next. During the first phase of the transformation, we perform a post-order traversal. For every operation y, considered as a node of T, let $p(y)$ denote its post-order number. Let also $s(y)$ denote the size of its rooted subtree. Before we start the traversal we precompute the values $p(y)$ and $s(y)$, for every operation y, in total $\mathcal{O}(L)$ time. Furthermore, as we traverse the tree we store some vertices in X in k different stacks, according to their respective labels. In what follows, a vertex v is identified with the operation $i(v)$ of creating this vertex. We keep the vertices in the stacks totally ordered with respect to their respective post-order numbers. In particular at any moment during the traversal, for every $1 \leq i \leq k$, the top entry of the i^{th} stack is the vertex in X of current label i with maximum post-order number. Let us now describe how to process every operation y:

1. *Case $y = i(v)$ for some i.* We remove the node if $v \notin X$. Otherwise, we push vertex v in the i^{th} stack.
2. *Case $y = \oplus$.* We remove the node if it has at most one child, or equivalently if at least one of the subgraphs G_1, G_2 of which we take the disjoint union does not contain a vertex of X.
3. *Case $y = \eta_{i,j}$ for some distinct i and j.* If either the i^{th} stack or the j^{th} stack is empty, then we remove the node. Otherwise, let v_i and v_j denote the respective top entries of the i^{th} and the j^{th} stacks. We remove the node if and only if at least one of v_i, v_j is not in the subtree rooted at y. It can be checked in constant time by verifying whether $p(y) - s(y) + 1 \leq p(v_i) \leq p(y)$, resp. whether $p(y) - s(y) + 1 \leq p(v_j) \leq p(y)$. Indeed, recall that nodes in the subtree rooted at y are post-ordered consecutively.

4. *Case $y = \rho_{i\to j}$ for some distinct i and j.* If either the i^{th} stack is empty or its top entry is not in the subtree rooted at y, then we remove the node. If either the j^{th} stack is empty or its top entry is not in the subtree rooted at y, then we *mark* this node. Indeed, such an operation is useless because it is just a local permutation of the labels i and j. However in order to remove this operation we may need to rewrite several other operations, and therefore we postpone its removal to the next phase of the algorithm. If we do not remove y (that also includes the sub-case when we mark this operation), then we pop the respective top entries of the i^{th} stack and (if it exists) the j^{th} stack, pushing back in the j^{th} stack the vertex with maximum post-order number. Then, we iteratively pop vertices from the i^{th} stack until either the former is empty or its top entry is not in the subtree rooted at y.

By taking as our potential function the cumulative sizes of all stacks, we can observe that we can process every operation in amortized constant time. Therefore, the total runtime is in $\mathcal{O}(L)$.

During the second phase of the transformation, we perform a DFS traversal in order to remove the marked nodes. We further need to rewrite some unmarked operations, that is done depending on some permutation σ over the labels which we dynamically update during the traversal. Initially, σ is the identity function. We store a k-size array that map each label i to $\sigma(i)$. Let us now describe how to process every node y during the traversal:

1. Case y is unmarked. If $y = \oplus$, then we leave this node unchanged. However, if $y = i(v)$ for some label i and some $v \in X$, then we rewrite the operation as $i'(v)$, where $i' = \sigma(i)$. Similarly, if $y = \eta_{i,j}$ (resp., $y = \rho_{i\to j}$) for some distinct i and j, then we rewrite the operation as $\eta_{\sigma(i),\sigma(j)}$ (resp., as $\rho_{\sigma(i)\to\sigma(j)}$). Note that we only need to rewrite an unmarked node at most once, namely, at the first time it is visited during the DFS traversal.

2. Case y is marked. Let $y = \rho_{i\to j}$ for some distinct i and j, and let z denote its unique child. Recall that we marked y because there is no vertex of label j in H_z, where H_z denotes the labelled subgraph output by the k-sub-expression whose subtree rooted at z is the parse tree. In order to suppress this operation, it suffices to switch the labels i and j for every operation in the subtree rooted at z. This is done in two steps. If we visit node y for the first time, then we actualize the permutation σ into τ so that: $\tau(i) = \sigma(j)$, $\tau(j) = \sigma(i)$. Then, we can process the subtree rooted at y. If we visit the node for the last time, then we go back from τ to σ, and then we can delete this node.

The runtime of this second phase, and so the runtime of the whole algorithm, is in $\mathcal{O}(L)$.

It remains to prove that indeed, the output k-expression of H has size at most $2|X| + |E_X|$. We first observe that it is an irredundant k-expression of H because it is constructed from an irredundant k-expression of G and we remove useless join operations. Then, let T' denote in what follows its parse tree. Recall that for each node y of T', we denote by H_y the labelled subgraph output by the k-sub-expression whose subtree rooted at y is the parse tree. We define $\Phi(y) := |V(H_y)| + |E(H_y)| + n_{\oplus}(y) - |\ell(H_y)|$ where $n_{\oplus}(y)$ denotes the number of disjoint union operations in the subtree rooted at y and $\ell(H_y)$ is the set of labels assigned to the vertices of H_y. Let $C(y)$ be the set of children of y (of cardinality at most two). We have that $\Phi(y) > \sum_{z\in C(y)} \Phi(z)$ (otherwise, the k-expression would not be irredundant, or some more operations could be discarded). In particular, if r is the root of T', we have that $\Phi(r)$ is an upper bound on the number of nodes. Since for any y we have $0 \leq \Phi(y) \leq 2|X|+|E_X|$ (where the upper bound also follows from the fact that the k-expression is irredundant), the total number of nodes in T' is at most $2|X|+|E_X| = \mathcal{O}(|X|+|E_X|)$. □

Divide-and-conquer algorithm

We are now ready to prove the main result in this section:

Proof of Theorem 1. Compute from the k-expression of G some width-k partition tree (Lemma 2) and then, from the latter, some balanced cut (U, W) as in Lemma 3. It takes $\mathcal{O}(k \cdot (n + m))$ time. For some partition $U_1, U_2, \ldots, U_\ell$ of U, for some $\ell \leq k$, the edges of the cut are partitioned into ℓ joins, with respective sides U_i and $W_i = N(U_i) \cap W$ for every i (we allow the side W_i to be empty, that may happen for at most one i).

We consider these ℓ joins sequentially, from $i = 1$ to $i = \ell$. At each step i, we are given a subgraph G_i of G such that $cw(G_i) \leq 3k$ and a corresponding $3k$-expression is given (initially, $G_1 := G$). We apply Lemmas 4 and 5 in order to compute the intersection of a maximum matching of G_i with the join with sides U_i, W_i. It takes $\tilde{\mathcal{O}}_k(n+m)$ time. Denote $F_i \subseteq U_i \times W_i$ the set of edges in this intersection, and let $V(F_i)$ be the set of vertices incident to an edge of F_i. We obtain G_{i+1} from G_i by removing the vertices in $V(F_i)$ and then removing all remaining edges between $U_i \setminus V(F_i)$ and $W_i \setminus V(F_i)$.

Let $M_i := \bigcup_{j=1}^{i} F_j$ be the matching constructed so far, and let $V(M_i)$ be the set of vertices incident to it. Let also $X_i := \bigcup_{j=1}^{i} U_j$. By induction, the union of M_i with a maximum matching of G_{i+1} is a maximum matching of G. Also by induction, G_{i+1} is obtained from $G \setminus V(M_i)$ by removing all edges between $X_i \setminus V(M_i)$ and $W \setminus V(M_i)$. Therefore, we can apply Lemma 7 (for $X = V \setminus V(M_i)$) then Lemma 6 (for $U' = X_i \setminus V(M_i)$) to compute a $3k$-expression of G_{i+1}. It takes $\mathcal{O}(n+m)$ time. Note that since we always apply Lemma 6 to G, and not to the G_i's, there is no blow-up of the clique-width value, *i.e.*, the clique-width of any G_i is at most $3k$.

Since all edges between U and W eventually got removed, we are left with computing a maximum matching in $G[U \setminus V(M_\ell)]$ and in $G[W \setminus V(M_\ell)]$ respectively. Apply Lemma 7 to compute k-expressions for both subgraphs, then call our above algorithm recursively in order to compute a maximum matching. Since the cut is balanced, the recursive depth is in $\mathcal{O}(\log n)$. □

5 Algorithms: the bipartite case

We present in this section an alternative to Theorem 1, with polynomial dependency on the clique-width, but only for bipartite graphs (see Theorem 2). The structure of bipartite graphs of bounded clique-width can be quite complex. For instance, let $G = (V, E)$ be any graph and let $V' = \{v' \mid v \in V\}$ be a disjoint copy of V. We can define the bipartite graph $B_G = (V \cup V', \{u'v \mid uv \in E\})$ and, if G has clique-width at most k, then B_G has clique-width at most $2k$. Some classes of monogenic bipartite graphs are also known to have bounded clique-width [22]. In general, bipartite graphs have bounded clique-width if and only if their corresponding binary matroids have bounded branch-width [66].

5.1 Reduction to Maximum b-matching

We define $\mathbb{N}$ to be the set of non-negative integers. Let $G = (V, E)$ be a graph and let $b : V \to \mathbb{N}$ assign a non-negative capacity to each vertex. We say that $x : E \to \mathbb{N}$ is a b-matching if we have $\sum_{u \in N_G(v)} x_{uv} \leq b(v)$ for each vertex v. Observe that a matching is a b-matching for the trivial function $b(v) = 1$ for each $v \in V$. The cardinality of a b-matching is defined as $\|x\|_1 = \sum_{e \in E} x_e$. We denote by $\nu(G, b)$ the cardinality of a maximum b-matching in G. Let also $\|b\|_1 = \sum_{v \in V} b(v)$ be the sum of all the vertex capacities. For every vertex-subset S, let $b(S) = \sum_{v \in S} b(v)$. Given a b-matching x and a vertex v, let $d_x(v) := \sum_{u \in N_G(v)} x_{uv} \leq b(v)$.

We start with the following reduction rule:

Lemma 8. *Let (G, b) be some instance of* Maximum b-Matching*, and let U and W be disjoint vertex-subsets such that there is a join between U and W. Consider the new instance (G', b') obtained from (G, b) by removing all edges between U and W, adding two new vertices $u, w \notin V(G)$ and edges $\{uw\} \cup \{uv \mid v \in U\} \cup \{wv' \mid v' \in W\}$, and finally setting $b'(u) = b'(w) = \min\{b(U), b(W)\}$. Then, we have $\nu(G', b') = \nu(G, b) + \min\{b(U), b(W)\}$.*

Moreover, if G is bipartite, then so is G'.

Proof. We refer to Fig. 2 for an illustration. First, let x be any b-matching for (G, b). Observe that we have $\sum_{v \in U, v' \in W} x_{vv'} \leq \min\{b(U), b(W)\}$. We transform it into a b-matching x' for (G', b'), such that: $x'_{uv} = \sum_{v' \in W} x_{vv'}$ for every $v \in U$; $x'_{wv'} = \sum_{v \in U} x_{vv'}$ for every $v' \in W$; and $x'_{uw} = \min\{b(U), b(W)\} - \sum_{v \in U, v' \in W} x_{vv'}$. Notice that, $\|x'\|_1 = \|x\|_1 + \min\{b(U), b(W)\}$. Hence, $\nu(G', b') \geq \nu(G, b) + \min\{b(U), b(W)\}$.

Conversely, let x' be any maximum b-matching for (G', b'). Without loss of generality, $\sum_{v \in U} x'_{uv} \leq \sum_{v' \in W} x'_{wv'}$. In particular, $x'_{uw} = \min\{b(U), b(W)\} - \sum_{v' \in W} x'_{wv'}$ (otherwise, x' could not be maximum).

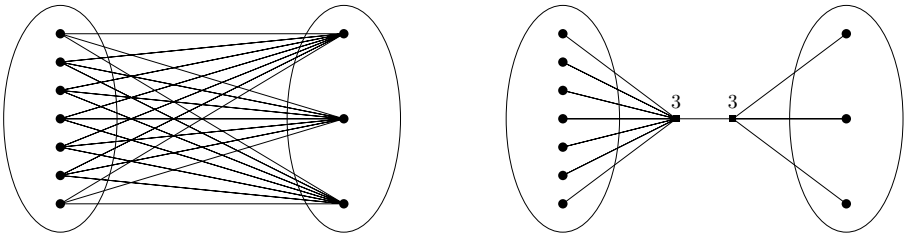

Figure 2: Transformation of Lemma 8.

Let $y : U \times W \to \mathbb{N}$ such that: $\sum_{v' \in W} y_{vv'} = x'_{vu}$ for every $v \in U$ (in particular, $\|y\|_1 = \sum_{v \in U} x'_{uv}$); and $\sum_{v \in U} y_{vv'} \leq x'_{v'w}$ for every $v' \in W$. Such a b-matching (for the bipartite graph with sides U, W) always exists because there is a complete join between U and W. We now construct x from x' (restricted to E) by adding to it the b-matching y. Doing so, $\|x\|_1 = \|x'\|_1 - \left(x'_{uw} + \sum_{v \in U} x'_{vu} + \sum_{v' \in W} x'_{v'w}\right) + \|y\|_1 = \|x'\|_1 - \left(\min\{b(U), b(W)\} + \sum_{v \in U} x'_{vu}\right) + \sum_{v \in U} x'_{uv} = \|x'\|_1 - \min\{b(U), b(W)\}$. As a result, $\nu(G, b) \geq \nu(G', b') - \min\{b(U), b(W)\}$.

Finally, let us further assume G to be bipartite. The sides U, W of the join must be in different colour classes. To obtain a proper bicolouring of G', we include vertices u and w, respectively, in the same colour class as the vertices in W and U, respectively. □

We stress that the transformation of Lemma 8 holds even if either side U or W of the join is empty. Indeed, in this degenerate situation we only add to G two new adjacent vertices u and w with zero capacity. By repeatedly applying Lemma 8 to some special balanced edge-cuts, we obtain:

Proposition 1. *There is an $\mathcal{O}(k \cdot (n+m) \log n)$-time reduction from* Maximum Matching *on graphs with clique-width at most k (if a k-expression is known) to* Maximum b-Matching *on graphs with tree-width $\mathcal{O}(k \log n)$. For the resulting instance (H, b), the algorithm also outputs a corresponding tree decomposition. Furthermore, $|V(H)| \leq \|b\|_1 \leq \mathcal{O}(\min\{n+m, n \log n\})$, and if G is bipartite, then so is H.*

Proof. We assume in what follows the input graph G to be given with some $b : V \to \mathbb{N}$. Initially, $b(v) = 1$ for every vertex $v \in V$. Let us further assume $|V(G)| > k + 1$ (otherwise, $tw(G) \leq k$ and there is nothing to be done). We apply Lemma 2, then Lemma 3, and call (U, W) the resulting balanced cut. It takes $\mathcal{O}(k \cdot (n+m))$ time. Recall that U is an ℓ-module of G, for some $\ell \leq k$, and that we also computed a corresponding ℓ-partition $U_1, U_2, \ldots, U_\ell$. For every $1 \leq i \leq \ell$, let $W_i := N_G(U_i) \cap W$. Since all the U_i's are disjoint, one can compute all the W_i's in $\mathcal{O}(m)$ time. We then apply Lemma 8, for the join with sides U_i and W_i, calling u_i, w_i the two new vertices created by this transformation. Notice that, in the intermediate graph H^0 obtained after our ℓ applications of Lemma 8, the vertices $u_1, w_1, u_2, w_2, \ldots, u_\ell, w_\ell$ disconnect U from W. We apply Lemma 7 to compute k-expressions of $G[U]$ and $G[W]$, then we call our reduction recursively to both subgraphs. Let (H_U, b_U) and (H_W, b_W) be the resulting instances, and let $(T_U, \mathcal{X}_U)$ and $(T_W, \mathcal{X}_W)$ be their tree decompositions. The final instance (H, b) is such that:

- $V(H) = V(H_U) \cup V(H_W) \cup \{u_1, w_1, \ldots, u_\ell, w_\ell\}$;
- H_U, H_W are induced subgraph of H and, for each $1 \leq i \leq \ell$, the edges of $\{u_i w_i\} \cup \{u_i v_i \mid v_i \in U_i\} \cup \{w_i v'_i \mid v'_i \in W_i\}$ are in H;
- $b(v) = b_U(v)$ for each $v \in V(H_U)$; $b(v') = b_W(v')$ for each $v' \in V(H_W)$; $b(u_i) = b(w_i) = \min\{|U_i|, |W_i|\}$ for each $1 \leq i \leq \ell$.

We get a tree decomposition for H by adding an edge between a bag of $(T_U, \mathcal{X}_U)$ and a bag of $(T_W, \mathcal{X}_W)$, then adding vertices $u_1, w_1, u_2, w_2, \ldots, u_\ell, w_\ell$ to all the bags. Correctness of the reduction follows from Lemma 8 (applied to all the cuts considered, at each recursive stage of the reduction, that are pairwise edge-disjoint). In particular, if G is bipartite then so is H. Since $\max\{|U|, |W|\} \leq 2n/3$ (Lemma 3), there are $\mathcal{O}(\log n)$

recursive stages, and so, the running time is in $\mathcal{O}(k \cdot (n+m) \log n)$. Similarly, since the treewidth increases by at most $2k$ at each stage, $tw(H) = \mathcal{O}(k \log n)$.

We can always ensure $|V(H)| \leq \|b\|_1$ by removing vertices with null capacity. At the start of the reduction, $\|b\|_1 = n$. Then, we observe that for each $1 \leq i \leq \ell$, the vertices u_i, w_i replace some edge between U_i, W_i (if no such edge exists, then $b(u_i) = b(w_i) = |W_i| = 0$, and we may discard u_i, w_i from H). In particular, $b(u_i) = b(w_i) = \min\{|U_i|, |W_i|\} \leq |U_i| \cdot |W_i|$, that is the number of edges part of the join. Since all the cuts considered throughout the reduction are edge-disjoint, we increase the sum of all vertex-capacities by at most $2m$ in total, and so, $\|b\|_1 \leq n + 2m$. Alternatively, observe that we have $\sum_{i=1}^{\ell}(b(u_i) + b(w_i)) \leq 2|U| \leq 4n/3$, and therefore, the sum of all capacities increases by $\mathcal{O}(n)$ at each stage. It thus also follows that $\|b\|_1 = \mathcal{O}(n \log n)$. □

5.2 Reduction to Linear Programming

The Maximum b-Matching problem is a classic example of an integer linear program. It is well-known that for the special case of Maximum Matching within *bipartite* graphs, we can drop the condition for all variables to be integers, thus reducing the computation of the matching number to the solving of a linear program [59]. Because of Tutte's quasi-polynomial reduction from Maximum b-Matching to Maximum Matching [75], this is also true for Maximum b-Matching within bipartite graphs. Very recently, Dong et. al. [26] answered an open question from Fomin et. al. [39] about bounded tree-width linear programs. We restate below their main result:

Theorem 8 ([26])**.** *Given a linear program* $\max_{Mx=b, \ell \leq x \leq u} c^\intercal x$ [3]*, where* $M \in \mathbb{R}^{d \times n}$ *is a full-rank matrix with* $d \leq n$*, define the* **dual graph** G_M *to be the graph with vertex-set* $\{1, 2 \ldots, d\}$*, such that* $ij \in E(G_M)$ *if there is a column* r *such that* $M_{i,r} \neq 0$ *and* $M_{j,r} \neq 0$*.*

Suppose that a tree decomposition of G_M *with width* τ *is given, and* R *is the diameter of the polytope, namely, for any* $\ell \leq x \leq u$ *with* $Mx = b$*, we have* $\|x\|_2 \leq R$*.*

Then, for any $0 < \varepsilon \leq 1$*, we can find* $\ell \leq x^* \leq u$ *such that*

$$c^\intercal x^* \geq \max_{Mx=b, \ell \leq x \leq u} c^\intercal x - \varepsilon \cdot \|c\|_2 \cdot R \text{ and } \|Mx^* - b\|_2 \leq \varepsilon \cdot (\|M\|_2 \cdot R + \|b\|_2)$$

in expected time $\tilde{\mathcal{O}}(n \cdot \tau^2 \log(1/\varepsilon))$*.*

Dong et. al. [26] referred to [71] and [57] for a detailed discussion about converting an approximate solution to an exact solution. We give a direct proof for Maximum b-Matching within bipartite graphs. For that, we combine a folklore reduction to Maximum Flow with a nice rounding technique by Madry [60].

Proposition 2. *The* Maximum b-Matching *problem within bipartite graphs of tree-width at most* τ *can be solved in expected* $\tilde{\mathcal{O}}(n\tau^2 \log \|b\|_1)$ *time, if a corresponding tree decomposition is given in the input.*

Proof. Let $G = (V_0 \cup V_1, E)$ be a bipartite graph and let $b : V_0 \cup V_1 \to \mathbb{N}$. The incidence matrix of G is the $n \times m$ matrix M such that $M_{v,e} = 1$ if v is an end-vertex of edge e and $M_{v,e} = 0$ otherwise. Let also c be the all-one vector. To compute the cardinality of a maximum b-matching for G, it suffices to solve the linear program $\max_{Mx \leq b} c^\intercal x$. We slightly modify this above program so that we fit in the conditions of Theorem 8. First, if we let $\ell, u \in \mathbb{R}^m$ such that ℓ is all-zero and $u_{vv'} = \min\{b(v), b(v')\}$ for each edge $vv' \in E$, then we must now solve $\max_{Mx \leq b, \ell \leq x \leq u} c^\intercal x$. If we further add n new variables $(x_v)_{v \in V}$ such that $0 \leq x_v \leq b(v)$ for each $v \in V$, then we can replace all constraints by $x_v + \sum_{vv' \in E} x_{vv'} = b(v)$ for each vertex $v \in V$, thus getting a new linear program $\max_{M'x=b, \ell' \leq x \leq u'} c'^\intercal x$ to solve, where $M' \in \mathbb{R}^{n \times (m+n)}$ and we obtain c' from c by completing the latter with n zero coordinates for the slack variables $(x_v)_{v \in V}$.

Note that, since we constructed M' from M by adding n new columns with exactly one nonzero value each, we have $G_{M'} = G_M = G$. Furthermore, an easy upper bound on the diameter R of the polytope is $R \leq \|b\|_2 \leq \|b\|_1$. We also have $\|c'\|_2 = \sqrt{m}$ and $\|M'\|_2 \leq \sqrt{\sum_v (1 + d(v))^2} \leq n^{3/2}$.

[3] The result is stated in [26] for minimization problems. Since we only consider it here for Maximum b-Matching, we rather write it as a maximization problem.

Assume G to be given with a tree decomposition of width at most τ, and apply Theorem 8 to the above linear program with $\varepsilon = 1/(4 \cdot \|b\|_1 \cdot n^2)$. We denote by x^* its output. Set all variables $x^*_v, v \in V$ to 0. Then, for all vertices v such that $\sum_{vv' \in E} x^*_{v'v} > b(v)$, we decrease the variables $x^*_{v'v}$ of incident edges until we reach equality. In doing so, we obtain a fractional b-matching y. By construction:

$$\|x^*\|_1 - \|y\|_1 \leq \sum_v \max\{0, (M'x^*)_v - b(v)\} \leq \|M'x^* - b\|_1 \leq \sqrt{n} \cdot \|M'x^* - b\|_2$$
$$\leq \varepsilon\sqrt{n} \cdot (\|M'\|_2 \cdot R + \|b\|_2) \leq \varepsilon\sqrt{n} \cdot \|b\|_1 \cdot (n^{3/2} + 1) \leq 2\varepsilon n^2 \cdot \|b\|_1 \leq 1/2.$$

We now construct a network D from G by adding two new vertices s and t, an arc (s, u) for every $u \in V_0$, an arc (v, t) for every $v \in V_1$, and finally by orienting all the edges of G from V_0 to V_1. The capacities of the arcs are defined as follows: $\kappa(s, v) = b(v)$ for every $v \in V_0$; $\kappa(v', t) = b(v')$ for every $v' \in V_1$; $\kappa(v, v') = \min\{b(v), b(v')\}$ for every $vv' \in E$ such that $v \in V_0, v' \in V_1$. Then, we construct a fractional st-flow as follows: $f_y(s, v) = \sum_{vv' \in E} y_{vv'}$ for every $v \in V_0$; $f_y(v', t) = \sum_{vv' \in E} y_{vv'}$ for every $v' \in V_1$; and $f_y(v, v') = y_{vv'}$ for every $vv' \in E$ such that $v \in V_0, v' \in V_1$. Note that the value of this flow is exactly $\|y\|_1$. Let f'_y be an integral st-flow of value $\lfloor \|y\|_1 \rfloor$. It can be constructed from f_y in $\tilde{\mathcal{O}}(m) = \tilde{\mathcal{O}}(\tau n)$ time [60, Corollary 3.4]. There is a one-to-one mapping between b-matchings in G and integral st-flows in D. In particular, the maximum value of a st-flow is exactly $\nu(G, b)$. Furthermore,

$$\|y\|_1 \geq \|x^*\|_1 - 1/2 \geq \nu(G, b) - \varepsilon \cdot \|c\|_2 \cdot R - 1/2 \geq \nu(G, b) - 1.$$

Therefore, we can transform f'_y into a maximum st-flow f_z by computing at most one augmenting path in the residual graph $D_{f'_y}$. It can be done in $\mathcal{O}(m) = \mathcal{O}(\tau n)$ time. The resulting b-matching is maximum: for every $vv' \in E$ with $v \in v_0, v' \in V_1$, we set $z_{vv'} = f_z(v, v')$. □

We are finally ready to prove our second main result in this paper:

Theorem 2. *Given a bipartite graph G and a corresponding k-expression, one can compute a maximum matching for G in randomized $\tilde{\mathcal{O}}(k^2 \cdot (n + m))$ time.*

Proof. The result follows from the combination of Proposition 1 with Proposition 2. □

6 Conclusion

We left open whether an $\tilde{\mathcal{O}}(k^c \cdot (n+m))$-time algorithm exists for computing a maximum matching within the graphs of clique-width at most k, for some constant c. Recall that our algorithm in the paper runs in almost linear time, but that it has a super-polynomial dependency on the clique-width. Relatedly, we left open the complexity of Maximum b-Matching within bounded tree-width graphs. It follows from Proposition 1 that an $\tilde{\mathcal{O}}(k^c \cdot (n + m))$-time algorithm for this problem within the graphs of tree-width at most k would imply a similar algorithm for Maximum Matching within graphs of clique-width at most k.

Acknowledgements. We thank the anonymous reviewers for their valuable feedback.

Conflict of Interest. This work was supported by Grant TC ICUB-SSE 15109-26.07.2021, "The complexity landscape of Maximum Matching". It was also supported by project PN-19-37-04-01 "New solutions for complex problems in current ICT research fields based on modelling and optimization", funded by the Romanian Core Program of the Ministry of Research and Innovation (MCI) 2019-2022.

References

[1] A. Abboud, V. Vassilevska Williams, and J. R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the twenty-seventh annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 377–391. SIAM, 2016.

[2] J. Alman and V. Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

[3] N. Anari and V. Vazirani. Matching Is as Easy as the Decision Problem, in the NC Model. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, volume 151 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 54:1–54:25. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020.

[4] R. Anstee. A polynomial algorithm for b-matchings: an alternative approach. *Information Processing Letters*, 24(3):153–157, 1987.

[5] M. Bentert, T. Fluschnik, A. Nichterlein, and R. Niedermeier. Parameterized aspects of triangle enumeration. *Journal of Computer and System Sciences*, 103:61–77, 2019.

[6] C. Berge. Sur le couplage maximum dun graphe. *COMPTES RENDUS HEBDOMADAIRES DES SEANCES DE L'ACADEMIE DES SCIENCES*, 247(3):258–259, 1958.

[7] J. A. Bondy and U. S. R. Murty. *Graph theory*, volume 244 of *Graduate Texts in Mathematics*. Springer-Verlag London, 2008.

[8] M. Carmosino, J. Gao, R. Impagliazzo, I. Mihajlin, R. Paturi, and S. Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 261–270, 2016.

[9] M. Chang. Algorithms for maximum matching and minimum fill-in on chordal bipartite graphs. In *ISAAC*, pages 146–155. Springer, 1996.

[10] J. Cheriyan. Randomized $\tilde{\mathcal{O}}(M(|V|))$ Algorithms for Problems in Matching Theory. *SIAM Journal on Computing*, 26(6):1635–1655, 1997.

[11] V. Cohen-Addad, M. Habib, and F. de Montgolfier. Algorithmic aspects of switch cographs. *Discrete Applied Mathematics*, 200:23–42, 2016.

[12] D. G. Corneil and U. Rotics. On the relationship between clique-width and treewidth. *SIAM J. Comput.*, 34(4):825–847, 2005.

[13] D. Coudert, G. Ducoffe, and A. Popa. Fully polynomial FPT algorithms for some classes of bounded clique-width graphs. *ACM Transactions on Algorithms (TALG)*, 15(3):1–57, 2019.

[14] B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.

[15] B. Courcelle and J. Engelfriet. *Graph structure and monadic second-order logic: a language-theoretic approach*, volume 138. Cambridge University Press, 2012.

[16] B. Courcelle, P. Heggernes, D. Meister, C. Papadopoulos, and U. Rotics. A characterisation of clique-width through nested partitions. *Discrete Applied Mathematics*, 187:70–81, 2015.

[17] B. Courcelle, J. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.

[18] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109(1-2):49–82, 1993.

[19] B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1):77–114, 2000.

[20] B. Courcelle and A. Twigg. Constrained-path labellings on graphs of bounded clique-width. *Theory of Computing Systems*, 47(2):531–567, 2010.

[21] B. Courcelle and R. Vanicat. Query efficient implementation of graphs of bounded clique-width. *Discrete Applied Mathematics*, 131(1):129–150, 2003.

[22] K. Dabrowski and D. Paulusma. Classifying the clique-width of H-free bipartite graphs. *Discrete Applied Mathematics*, 200:43–51, 2016.

[23] E. Dahlhaus and M. Karpinski. Matching and multidimensional matching in chordal and strongly chordal graphs. *Discrete Applied Mathematics*, 84(1-3):79–91, 1998.

[24] R. Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2010. 4*th* edition.

[25] J. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4(5):406–451, 1970.

[26] S. Dong, Y. T. Lee, and G. Ye. A nearly-linear time algorithm for linear programs with small treewidth: a multiscale representation of robust central path. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1784–1797, 2021.

[27] F. Dragan. On greedy matching ordering and greedy matchable graphs. In *WG'97*, volume 1335 of *LNCS*, pages 184–198. Springer, 1997.

[28] G. Ducoffe. Maximum Matching in Almost Linear Time on Graphs of Bounded Clique-Width. In P. A. Golovach and M. Zehavi, editors, *International Symposium on Parameterized and Exact Computation (IPEC 2021)*, volume 214 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

[29] G. Ducoffe. Optimal Centrality Computations Within Bounded Clique-Width Graphs. In P. A. Golovach and M. Zehavi, editors, *International Symposium on Parameterized and Exact Computation (IPEC 2021)*, volume 214 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

[30] G. Ducoffe and A. Popa. The b-Matching Problem in Distance-Hereditary Graphs and Beyond. In *29th International Symposium on Algorithms and Computation (ISAAC 2018)*, volume 123 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

[31] G. Ducoffe and A. Popa. The use of a pruned modular decomposition for Maximum Matching algorithms on some graph classes. *Discrete Applied Mathematics*, 291:201–222, 2021.

[32] J. Edmonds. Paths, trees, and flowers. *Canadian J. of mathematics*, 17(3):449–467, 1965.

[33] W. Espelage, F. Gurski, and E. Wanke. How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2001)*, volume 1 of *Lecture Notes in Computer Science*, pages 117–128. Springer, 2001.

[34] T. Fluschnik, G. Mertzios, and A. Nichterlein. Kernelization lower bounds for finding constant-size subgraphs. In *Conference on Computability in Europe*, pages 183–193. Springer, 2018.

[35] F. Fomin and T. Korhonen. Fast FPT-Approximation of Branchwidth. Technical Report 2111.03492, arXiv, 2021.

[36] F. V. Fomin, P. A. Golovach, D. Lokshtanov, and S. Saurabh. Intractability of clique-width parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010.

[37] F. V. Fomin, P. A. Golovach, D. Lokshtanov, and S. Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM J. Comput.*, 43(5):1541–1563, 2014.

[38] F. V. Fomin, P. A. Golovach, D. Lokshtanov, S. Saurabh, and M. Zehavi. Clique-width III: Hamiltonian Cycle and the Odd Case of Graph Coloring. *ACM Transactions on Algorithms (TALG)*, 15(1):9, 2019.

[39] F. V. Fomin, D. Lokshtanov, S. Saurabh, M. Pilipczuk, and M. Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Transactions on Algorithms (TALG)*, 14(3):34:1–34:45, 2018.

[40] J.-L. Fouquet, V. Giakoumakis, and J.-M. Vanherpe. Bipartite graphs totally decomposable by canonical decomposition. *International J. of Foundations of Computer Science*, 10(04):513–533, 1999.

[41] J.-L. Fouquet, I. Parfenoff, and H. Thuillier. An $O(n)$-time algorithm for maximum matching in P_4-tidy graphs. *Information processing letters*, 62(6):281–287, 1997.

[42] M. Fürer. A natural generalization of bounded tree-width and bounded clique-width. In *Latin American Symposium on Theoretical Informatics*, pages 72–83. Springer, 2014.

[43] H. Gabow. Data structures for weighted matching and extensions to b-matching and f-factors. *ACM Transactions on Algorithms (TALG)*, 14(3):1–80, 2018.

[44] H. Gabow and P. Sankowski. Algorithms for Weighted Matching Generalizations I: Bipartite Graphs, b-matching, and Unweighted f-factors. *SIAM Journal on Computing*, 50(2):440–486, 2021.

[45] T. Gallai. Kritische graphen II. *Magyar Tud. Akad. Mat. Kutato Int. Kozl.*, 8:373–395, 1963.

[46] T. Gallai. Maximale systeme unabhangiger kanten. *Magyar Tud. Akad. Mat. Kutato Int. Kozl.*, 9:401–413, 1964.

[47] A. Gerards. Matching. *Handbooks in operations research and management science*, 7:135–224, 1995.

[48] A. C. Giannopoulou, G. B. Mertzios, and R. Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *Theoretical Comput. Sci.*, 689:67–95, 2017.

[49] F. Glover. Maximum matching in a convex bipartite graph. *Naval Research Logistics (NRL)*, 14(3):313–316, 1967.

[50] M. C. Golumbic and U. Rotics. On the clique-width of some perfect graph classes. *International Journal of Foundations of Computer Science*, 11(03):423–443, 2000.

[51] T. Hagerup, J. Katajainen, N. Nishimura, and P. Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *Journal of Computer and System Sciences*, 57(3):366–375, 1998.

[52] F. Hegerfeld and S. Kratsch. On Adaptive Algorithms for Maximum Matching. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132, pages 71:1–71:16. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019.

[53] Y. Iwata, T. Ogasawara, and N. Ohsaka. On the power of tree-depth for fully polynomial FPT algorithms. In *International Symposium on Theoretical Aspects of Computer Science (STACS 2018)*, volume 96 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 41:1–41:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

[54] C. Jordan. Sur les assemblages de lignes. *J. Reine Angew. Math*, 70(185):81, 1869.

[55] S. Kratsch and F. Nelles. Efficient and Adaptive Parameterized Algorithms on Modular Decompositions. In *Annual European Symposium on Algorithms (ESA 2018)*, pages 55:1–55:15, 2018.

[56] S. Kratsch and F. Nelles. Efficient Parameterized Algorithms for Computing All-Pairs Shortest Paths. In *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*, volume 154, pages 38:1–38:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.

[57] Y. T. Lee and A. Sidford. Path Finding I: Solving Linear Programs with $\tilde{\mathcal{O}}(\sqrt{\text{rank}})$ Linear System Solves. Technical Report 1312.6677, arXiv, 2013.

[58] Y. Liang and C. Rhee. Finding a maximum matching in a circular-arc graph. *Information processing letters*, 45(4):185–190, 1993.

[59] L. Lovász and M. Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.

[60] A. Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 253–262. IEEE, 2013.

[61] J. A. Makowsky and U. Rotics. On the clique-width of graphs with few P_4's. *International Journal of Foundations of Computer Science*, 10(03):329–348, 1999.

[62] G. Mertzios, A. Nichterlein, and R. Niedermeier. A Linear-Time Algorithm for Maximum-Cardinality Matching on Cocomparability Graphs. *SIAM Journal on Discrete Mathematics*, 32(4):2820–2835, 2018.

[63] G. B. Mertzios, A. Nichterlein, and R. Niedermeier. The power of linear-time data reduction for maximum matching. In *International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*, volume 83 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 46:1–46:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.

[64] S. Micali and V. Vazirani. An $O(\sqrt{V}E)$ algorithm for finding maximum matching in general graphs. In *FOCS'80*, pages 17–27. IEEE, 1980.

[65] A. Moitra and R. Johnson. A parallel algorithm for maximum matching on interval graphs. In *ICPP*, 1989.

[66] S. Oum. Rank-width and vertex-minors. *Journal of Combinatorial Theory, Series B*, 95(1):79–100, 2005.

[67] S. Oum and P. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006.

[68] M. Padberg and M. Rao. The Russian method for linear inequalities III: Bounded integer programming. Technical Report 78, INRIA, 1981.

[69] W. Pulleyblank. *Faces of Matching Polyhedra*. PhD thesis, Univ. of Waterloo, Dept. Combinatorics and Optimization, 1973.

[70] M. Rao. Solving some NP-complete problems using split decomposition. *Discrete Applied Mathematics*, 156(14):2768–2780, 2008.

[71] J. Renegar. A polynomial-time algorithm, based on Newton's method, for linear programming. *Mathematical programming*, 40(1):59–93, 1988.

[72] J. Thatcher and J. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical systems theory*, 2(1):57–81, 1968.

[73] N. Trinajstić, D. Klein, and M. Randić. On some solved and unsolved problems of chemical graph theory. *International Journal of Quantum Chemistry*, 30(S20):699–742, 1986.

[74] W. Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 1(2):107–111, 1947.

[75] W. Tutte. A short proof of the factor theorem for finite graphs. *Canad. J. Math*, 6(1954):347–352, 1954.

[76] W. Tutte. Antisymmetrical digraphs. *Canadian Journal of Mathematics*, 19:1101–1117, 1967.

[77] V. Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, volume 3, pages 3431–3472. World Scientific, 2018.

[78] V. Vassilevska Williams and R. Williams. Subcubic equivalences between path, matrix, and triangle problems. *Journal of the ACM (JACM)*, 65(5):1–38, 2018.

[79] M.-S. Yu and C.-H. Yang. An $O(n)$-time algorithm for maximum matching on cographs. *Information processing letters*, 47(2):89–93, 1993.

[80] R. Yuster. Maximum matching in regular and almost regular graphs. *Algorithmica*, 66(1):87–92, 2013.

[81] R. Yuster and U. Zwick. Maximum matching in graphs with an excluded minor. In *Proceedings of the eighteenth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 108–117. Society for Industrial and Applied Mathematics, 2007.