The use of a pruned modular decomposition for Maximum Matching algorithms on some graph classes

Guillaume Ducoffe, Alexandru Popa

The use of a pruned modular decomposition for Maximum Matching algorithms on some graph classes [1]

Guillaume Ducoffe[a,b], Alexandru Popa[a,b]

[a] *ICI – National Institute for Research and Development in Informatics, Bucharest, Romania*
[b] *University of Bucharest, Bucharest, Romania*

Abstract

Consider the following general question: if we can solve Maximum Matching in (quasi) linear time on a graph class $\mathcal{C}$, does the same hold true for the class of graphs that can be *modularly decomposed* into $\mathcal{C}$? What makes the latter question difficult is that the Maximum Matching problem is *not* preserved by quotient, thereby making difficult to exploit the structural properties of the quotient subgraphs of the modular decomposition. So far, we are only aware of a recent framework in (Coudert et al., SODA'18) that only applies when the quotient subgraphs have bounded order and/or under additional assumptions on the nontrivial modules in the graph. As an attempt to answer this question for distance-hereditary graphs and some other superclasses of cographs, we study the combined effect of modular decomposition with a pruning process over the quotient subgraphs. Specifically, we remove sequentially from all such subgraphs their so-called one-vertex extensions (*i.e.*, pendant, anti-pendant, twin, universal and isolated vertices). Doing so, we obtain a "pruned modular decomposition", that can be computed in quasi linear time. Our main result is that if all the pruned quotient subgraphs have bounded order then a maximum matching can be computed in linear time. The latter result strictly extends the framework of Coudert et al. Our work is the first to use some ordering over the *modules* of a graph, instead of just over its vertices, in order to speed up the computation of maximum matchings on some graph classes.

Keywords: maximum matching; FPT in P; modular decomposition; pruned graphs; one-vertex extensions; P_4-structure.

1. Introduction

Can we compute a maximum matching in a graph in linear-time? – i.e., computing a maximum set of pairwise disjoint edges in a graph. – Despite considerable years of research and the design of elegant combinatorial and linear programming techniques, the best-known algorithms for this fundamental problem have stayed blocked to an $\mathcal{O}(m\sqrt{n})$-time complexity on n-vertex m-edge graphs [41]. Nevertheless, we can use some well-structured graph classes in order to overcome this superlinear barrier for particular cases of graphs [8, 12, 16, 19, 25, 24, 29, 34, 35, 37, 40, 42, 48, 50, 49]). Indeed, the Maximum Matching problem has several applications, some of them being relevant only for specific graph families [7, 18, 29, 38, 44]. Our work combines two successful approaches for this problem, namely, the use of a *pruning sequence* for certain graph

[1] Results of this paper were partially presented at the ISAAC'18 conference [21].

classes [8, 16, 40], and a recent technique based on the decomposition of a graph by its *modules* [12]. We detail these two approaches in what follows, before summarizing our contributions.

1.1. Related work

A cornerstone of most Maximum Matching algorithms is the notion of augmenting paths [3, 22]. However, although we can compute a set of augmenting paths in linear-time [26], this is a tedious task that involves the technical notion of blossoms and this may need to be repeated $\Omega(\sqrt{n})$ times before a maximum matching can be computed [34]. A well-known greedy approach consists in, given some total ordering $(v_1, v_2, \ldots, v_n)$ over the vertices in the graph, to consider the exposed vertices v_i (*i.e.*, not incident to an edge of the current matching) by increasing order, then to try to match them with some exposed neighbour v_j that appears later in the ordering [19]. The vertex v_j can be chosen either arbitrarily or according to some specific rules depending on the graph class we consider. Our initial goal was to extend similar reduction rules to *module-orderings*.

Modular decomposition. A module in a graph $G = (V, E)$ is any vertex-subset X such that every vertex of $V \setminus X$ is either adjacent to every of X or nonadjacent to every of X. The *modular decomposition* of G is a recursive decomposition of G according to its modules [32]. We postpone its formal definition until Section 2. For now, we only want to stress that the vertices in the "quotient subgraphs" that are outputted by this decomposition represent modules of G (*e.g.*, see Fig. 1 for an insightful illustration). The use of modular decomposition in the algorithmic field has a rich history. The successive improvements on the best-known complexity for computing this decomposition are already interesting on their own since they required the introduction of several new techniques [13, 15, 30, 39, 47]. There is now a practical linear-time algorithm for computing the modular decomposition of any graph [47]. Our main motivation for considering modular decomposition in this note is its recent use in the field of parameterized complexity for *polynomial-time solvable* problems. – For some earlier applications, see [1, 5, 14, 23, 27]. – Specifically, let us call *modular-width* of a graph G the minimum $k \geq 2$ such that every quotient subgraph in the modular decomposition of G is either "degenerate" (*i.e.*, complete or edgeless) or of order at most k. With Coudert, we proved in [12, Sec. 4 and 5] that many "hard" graph problems in P – for which no linear-time algorithm is likely to exist – can be solved in $k^{\mathcal{O}(1)}(n + m)$-time on graphs with modular-width at most k. In particular, we proposed an $\mathcal{O}(k^4 n + m)$-time algorithm for Maximum Matching.

One appealing aspect of our approach in [12, Sec. 4] was that, for most problems studied, we obtained a linear-time reduction from the input graph G to some (smaller) quotient subgraph G' in its modular decomposition. – We say that the problem is preserved by quotient. – This paved the way to the design of efficient algorithms for these problems on graph classes with *unbounded* modular-width, assuming their quotient subgraphs are simple enough w.r.t. the problem at hands. We illustrated this possibility through the case of $(q, q-3)$-graphs (*i.e.*, graphs where no set of at most q vertices, $q \geq 7$, can induce more than $q - 3$ paths of length four). However, this approach completely fell down for Maximum Matching. Indeed, our Maximum Matching algorithm in [12, Sec. 5] works on supergraphs of the quotient graphs that need to be repeatedly updated every time a new augmenting path is computed. Such approach did not help much in exploiting the structure of quotient graphs. We managed to do so for $(q, q-3)$-graphs only through the help of a deeper structural theorem on the nontrivial modules in this class of graphs. Nevertheless, to take a shameful example, it was not even known before this work whether Maximum Matching

could be solved faster than with the state-of-the art algorithms on graphs that can be modularly decomposed into *paths*!

1.2. Our contributions

We propose *pruning rules* on the modules in a graph (some of them new and some others revisited) that can be used in order to compute Maximum Matching in linear-time on several new graph classes. More precisely, given a module M in a graph $G = (V, E)$, we recall that M is corresponding to some vertex v_M in a quotient graph G' of the modular decomposition of G. Assuming v_M is a so-called *one-vertex extension* in G' (*i.e.*, it is pendant, anti-pendant, universal, isolated or it has a twin), we show that a maximum matching for G can be computed from a maximum matching of $G[M]$ and a maximum matching of $G \setminus M$ efficiently (see Section 4). Our rules are purely *structural*, in the sense that they only rely on the structural properties of v_M in G' and not on any additional assumption on the nontrivial modules. Some of these rules (*e.g.*, for isolated or universal modules) were first introduced in [12, Sec. 5.2] — although with slightly different correctness proofs. Our main technical contributions in this work are the pruning rules for, respectively, *pendant* and *anti-pendant* modules (see Sections 4.2 and 4.3). The latter two cases are, surprisingly, the most intricate. In particular, they require amongst other techniques: the computation of specified augmenting paths of length up to 7, the addition of some "virtual edges" in other modules, and a careful swapping between some matched and unmatched edges.

Then, we are left with pruning every quotient subgraph in the modular decomposition by sequentially removing the one-vertex extensions. We prove that the resulting "pruned quotient subgraphs" are unique (independent from the removal orderings) and that they can be computed in quasi linear-time by using a *trie* data-structure (Section 3). Furthermore, as a case-study we prove that several superclasses of cographs are totally decomposable w.r.t. this new "pruned modular decomposition"; namely, every graph that can be modularly decomposed into: trees, (co-)distance-hereditary graphs [2], tree-perfect graphs [6]. These classes are further discussed in Section 5. Note that for some of them, such as distance-hereditary graphs, we so obtain the first known linear-time algorithm for Maximum Matching – thereby extending previous partial results obtained for bipartite and chordal distance-hereditary graphs [16]. Our approach actually has similarities with a general greedy scheme applied to distance-hereditary graphs [10]. With slightly more work, we can extend our approach to every graph that can be modularly decomposed into cycles. The case of graphs of bounded *modular treewidth* [43] is left as an interesting open question. We also left open whether our framework in this paper could be applied to other interesting graph classes from the literature.

1.3. Organization of the paper

Definitions and our first results are presented in Section 2. We introduce the pruned modular decomposition in Section 3, where we show that it can be computed in quasi linear-time. Then, the core of the paper is Section 4 where the pruning rules are presented along with their correctness proofs. In particular, we state our main result in Section 4.4. Our reduction rules and algorithms are stated in a high-level description. Technical details related to their implementations and the required data structures are postponed to specific subsections (namely, Sec. 2.1.1, 2.2.1 and 3.1). We use the latter in order to bound the running time of our algorithms: in Sec. 4.1.1, 4.2.1 and 4.3.1, respectively. Applications of our approach to some graph classes are discussed in Section 5. Finally, we conclude in Section 6 with some open questions. Results of this paper were partially presented at the ISAAC'18 conference [21].

2. Preliminaries

For the standard graph terminology, see [4]. We only consider graphs that are finite, simple and unweighted. For any graph $G = (V, E)$ let $n = |V|$ and $m = |E|$. Given a vertex $v \in V$, we denote its (open) neighbourhood by $N_G(v) = \{u \in V \mid \{u, v\} \in E\}$ and its closed neighbourhood by $N_G[v] = N_G(v) \cup \{v\}$. Similarly, we define the neighbourhood of any vertex-subset $S \subseteq V$ as $N_G(S) = \left(\bigcup_{v \in S} N_G(v)\right) \setminus S$. In what follows, we introduce our main algorithmic tool for the paper as well as the graph problems we study.

2.1. Modular decomposition

A *module* in a graph $G = (V, E)$ is any subset $M \subseteq V(G)$ such that for any $u, v \in M$ we have $N_G(v) \setminus M = N_G(u) \setminus M$. There are trivial examples of modules such as $\emptyset$, V, and $\{v\}$ for every $v \in V$. Furthermore a fundamental property with nice algorithmic applications [32] is that, if M is a module of G, and $M' \subseteq M$, then M' is a module of $G[M]$ if and only if it is also a module of G. Let $\mathcal{P} = \{M_1, M_2, \ldots, M_p\}$ be a partition of the vertex-set V. If for every $1 \leq i \leq p$, M_i is a module of G, then we call $\mathcal{P}$ a *modular partition* of G. By abuse of notation, we will sometimes identify a module M_i with the induced subgraph $H_i = G[M_i]$, *i.e.*, we will write $\mathcal{P} = \{H_1, H_2, \ldots H_p\}$. The *quotient subgraph* $G/\mathcal{P}$ has vertex-set $\mathcal{P}$, and there is an edge between every two modules $M_i, M_j \in \mathcal{P}$ such that $M_i \times M_j \subseteq E$. – This terminology finds its roots in the study of equivalence classes. Indeed, a modular partition, as any partition of the vertex-set of a graph, induces an equivalence relation on the vertices. In the literature, the set of the equivalence classes of a relation is often called the quotient set. – Conversely, let $G' = (V', E')$ be a graph and let $\mathcal{P} = \{H_1, H_2, \ldots H_p\}$. be a collection of subgraphs. The *substitution graph* $G'(\mathcal{P})$ is obtained from G' by replacing every vertex $v_i \in V'$ with a module inducing H_i. In particular, for $G' =^{\text{def}} G/\mathcal{P}$ we have that $G'(\mathcal{P}) = G$. We say that G is *prime* if its only modules are trivial (*i.e.*, $\emptyset$, V, and the singletons $\{v\}$). We call a module M *strong* if it does not overlap any other module, *i.e.*, for any module M' of G, either one of M or M' is contained in the other or M and M' do not intersect. Let $\mathcal{M}(G)$ be the family of all inclusion wise maximal strong modules of G that are proper subsets of V. The family $\mathcal{M}(G)$ is a modular partition of G [32], and so, we can define $G' = G/\mathcal{M}(G)$. The following structure theorem is due to Gallai.

Theorem 1 ([28])**.** *For an arbitrary graph G exactly one of the following conditions is satisfied.*

1. *G is disconnected;*
2. *its complement $\overline{G}$ is disconnected;*
3. *or its quotient graph $G' = G/\mathcal{M}(G)$ is prime for modular decomposition.*

We now formally define the modular decomposition of G – introduced earlier in Section 1. We output the quotient graph $G' = G/\mathcal{M}(G)$ and, for any strong module $M \in \mathcal{M}(G)$ that is nontrivial (possibly none if $G = G'$), we also output the modular decomposition of $G[M]$. By Theorem 1 the subgraphs from the modular decomposition are either edgeless, complete, or prime for modular decomposition. See Fig. 1 for an example. The modular decomposition of a given graph $G = (V, E)$ can be computed in linear-time [47]. There are many graph classes that can be characterized using the modular decomposition.In particular, G is a cograph if and only if every quotient subgraph in its modular decomposition is either complete or disconnected [11].

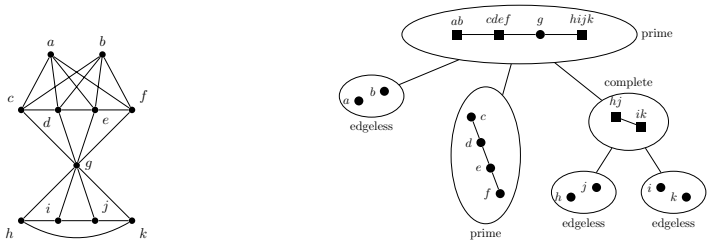


Figure 1: A graph and its modular decomposition.

2.1.1. Data structures and basic operations

Since the modular decomposition is our main algorithmic tool in this paper, it is important to carefully address the operations which we want to perform on the latter, and their time complexity. The state-of-the-art algorithm for computing the modular decomposition [47] outputs a linear-time representation of the quotient subgraphs, sometimes called the *modular decomposition tree*, that we introduce next as a particular case of *modular partition tree*. The definition of the latter is recursive. If a graph G contains a unique vertex v, then its unique modular partition tree is a single-node tree labeled by v. Otherwise, a modular partition tree for G is, given some modular partition $\mathcal{P} = \{M_1, M_2, \ldots, M_p\}$ of G, a rooted tree such that the p subtrees at the root are modular partition trees for $G[M_1], G[M_2], \ldots, G[M_p]$, respectively. In order to avoid degenerate situations, we further impose $p \geq 2$ and that all the modules in $\mathcal{P}$ must be nonempty, which we call a *non-trivial* partition. Finally, we obtain the modular decomposition tree of G by choosing $\mathcal{P} = \mathcal{M}(G)$ and, as subtrees for each of $G[M_1], G[M_2], \ldots, G[M_p]$, their respective modular decomposition trees. Note that there is a one-to-one mapping between the quotient subgraphs in the modular decomposition of G and the internal nodes of its modular decomposition tree. We label these internal nodes by either prime, series or parallel: if the corresponding quotient subgraph is either prime, edgeless or complete. See Fig. 2 for an illustration.

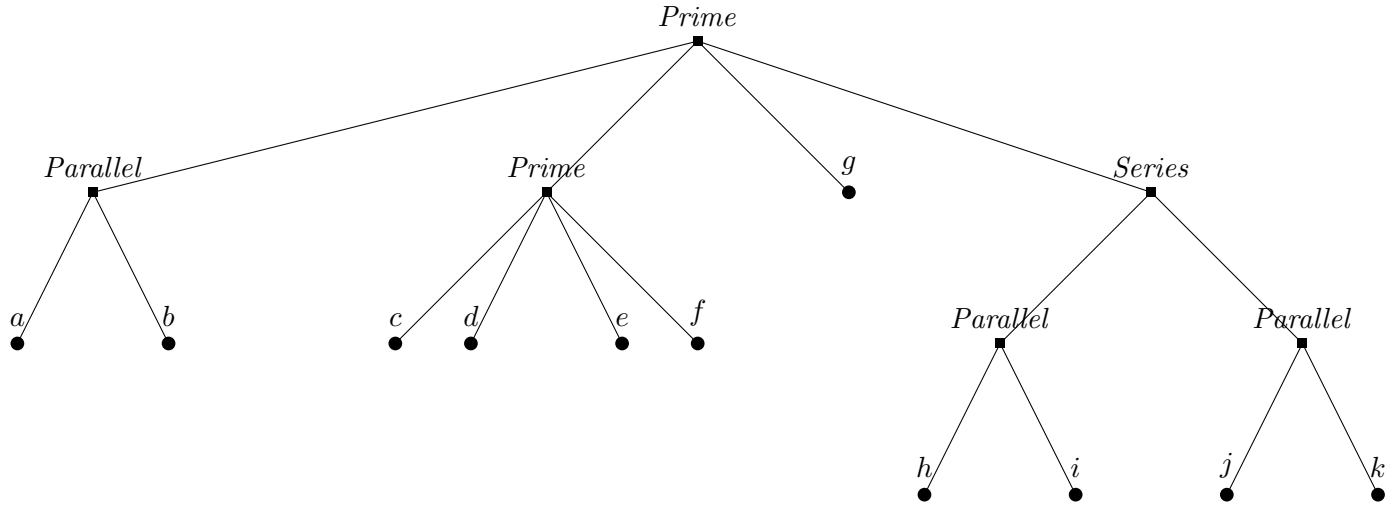


Figure 2: The modular decomposition tree of the graph G in Fig. 1.

Lemma 1. *Let G be a graph and let $\mathcal{P} = \{M_1, M_2, \ldots, M_p\}$, for some $p \geq 3$, be a non-trivial modular partition of G. For any modular partition tree w.r.t. G and $\mathcal{P}$, the following operations can be performed in $\mathcal{O}(1)$-time:*

- *Suppression of a module M_i (resulting in a modular partition tree for $G \setminus M_i$);*
- *Merge of two modules M_i, M_j s.t. $M_i \cup M_j$ is also a module of G (resulting in a new modular partition tree for G).*

Proof. First, in order to remove M_i, it suffices to remove the edge between the root and the modular partition tree of $G[M_i]$. In the same way, in order to merge two modules M_i, M_j (assuming their union is still a module of G), we start removing the two edges between the root and the modular partition trees of $G[M_i], G[M_j]$. Then, we add a new internal node, that we insert in the graph as a child of the root; its two subtrees are the the modular partition trees of $G[M_i], G[M_j]$. All these aforementioned operations take constant-time, assuming a standard pointer structure for trees. □

Since our reduction rules exploit the structure of the quotient subgraphs, it is desirable to explicitly compute, and store, these subgraphs. Our next lemma shows that it can be done "for free" if we are given the modular decomposition tree.

Lemma 2. *Given the modular decomposition tree of a graph G, all the quotient subgraphs in its modular decomposition can be computed in total $\mathcal{O}(n + m)$ time.*

Proof. We first partition the edges of G along the internal nodes of its modular decomposition tree. Indeed, let us consider any edge xy of G. There is a unique quotient subgraph H' of the modular decomposition such that: $H' = H/\mathcal{M}(H)$, for some subgraph H of G (induced by a module); $x, y \in V(H)$; and x, y are in different modules of $\mathcal{M}(H)$. Then, if $M_x, M_y \in \mathcal{M}(H)$ are such that $x \in M_x$ and $y \in M_y$, there must be an edge between v_{M_x} and v_{M_y} in H'. Therefore, we want to map xy to H'. We observe that the quotient H' is mapped to the *lowest common ancestor* of x, y in the modular decomposition tree. So, in order to construct this above partition of the edges, we first create an array of lists that is indexed by the internal nodes of the modular decomposition tree. We pre-process this tree in linear-time such that, for any two nodes, their lowest common ancestor can be computed in constant-time (*e.g.*, see [33]). Since the modular decomposition tree has $\mathcal{O}(n)$ nodes [32], this whole phase takes $\mathcal{O}(n)$ time. Finally, we scan all the edges xy of G. We compute the lowest common ancestor of x and y in the modular decomposition tree, in constant-time, in order to allocate the edge xy to the corresponding list in the array. Overall, the partition of the edges is constructed in total $\mathcal{O}(n + m)$ time.

During the second step of the algorithm, we effectively construct the quotient subgraphs H'. For that, for every quotient H', let E_H denote all the edges mapped to H'. Let G_H be the subgraph of G that is induced by E_H. Two non-adjacent vertices in $V(G_H)$ are *false twins* if they have the same neighbours in G_H (see also Sec. 3). This is an equivalence relation, of which we call "false twin classes" the equivalence classes. Recall that $H' = H/\mathcal{M}(H)$, for some subgraph H of G. By construction, the edges of E_H are exactly the edges of G whose ends are in different strong modules of $\mathcal{M}(H)$. We so obtain that the false twin classes of G_H are exactly the strong modules of $\mathcal{M}(H)$, *i.e.*, the vertices of H'. As a result, in order to construct H', it suffices to compute the false twin classes of G_H, to keep one vertex per false twin class and then to enumerate E_H. This can be done in $\mathcal{O}(|V(G_H)| + |E_H|) = \mathcal{O}(|E_H|)$ time, using partition refinement techniques, if we are given the adjacent list of G_H [31]. Therefore, we are left computing these adjacency lists, for all

the subgraphs G_H, in total $\mathcal{O}(n+m)$ time. For that, during a pre-processing phase, we create an array of lists and a boolean vector, that are both indexed by the vertices of the graph G. In order to create the quotient subgraph H', we first initialize some empty list for $V(G_H)$. We enumerate all edges $xy \in E_H$. If vertex x, resp. y, has not been put in $V(G_H)$ yet (that can be verified and updated in $\mathcal{O}(1)$ time, using our global boolean vector), then we add this vertex to the list. Then, in the global array of lists, we insert the edge xy in both the list indexed by x and the list indexed by y. Note that our approach does not allow us to store simultaneously the adjacency list of all the G_H's. After creating H' from G_H, we do not need the adjacency list of the latter anymore, and therefore we need to reset our two global arrays before being able to construct $G_{H''}$, for some other quotient $H'' \neq H'$ in the modular decomposition. We do so by scanning once $V(G_H)$ and clearing on the way the corresponding cells in these arrays. □

In the next lemma, we present two more operations on the modular decomposition that we need in order to implement efficiently some of our reduction rules (see Sec. 4.2).

Lemma 3. *Let G be a graph and let $\mathcal{P} = \{M_1, M_2, \ldots, M_p\}$ be a non-trivial modular partition of G. Given $G/\mathcal{P}$ and a modular partition tree w.r.t. G and $\mathcal{P}$, it is possible to enumerate, for any module $M_i \in \mathcal{P}$:*

- *all the vertices of M_i, in $\mathcal{O}(|M_i|)$ time;*
- *all the vertices of $N_G(M_i)$, in $\mathcal{O}(|N_G(M_i)|)$ time.*

Proof. First, in order to enumerate M_i, it suffices to perform a tree traversal on the modular partition tree of $G[M_i]$ (subtree at the root), and to enumerate all its leaves. We observe that, since we only allow non-trivial partitions for constructing such tree, there can be no internal node of degree two. In particular, the number of nodes in the modular partition tree of $G[M_i]$ is linear in the number of its leaves, and so, in $\mathcal{O}(|M_i|)$. Second, in order to enumerate $N_G(M_i)$, we start computing the family of modules $M_{j_1}, M_{j_2}, \ldots, M_{j_q} \in \mathcal{P}$ s.t. $N_G(M_i) = \bigcup_{k=1}^{q} M_{j_k}$. This can be done in $\mathcal{O}(q) = \mathcal{O}(deg_{G/\mathcal{P}}(v_{M_i}))$ time using $G/\mathcal{P}$, that is in $\mathcal{O}(|N_G(M_i)|)$ because the modules M_{j_k} are nonempty. We then enumerate M_{j_k}, for every $1 \leq k \leq q$, in $\mathcal{O}(|M_{j_k}|)$ time. □

2.2. Maximum Matching

A matching in a graph is defined as a set of edges with pairwise disjoint end vertices. The maximum cardinality of a matching in a given graph $G = (V, E)$ is denoted by $\mu(G)$.

Problem 1 (Maximum Matching)**.**

Input: *A graph $G = (V, E)$.*

Output: *A matching of G with maximum cardinality.*

We remind the reader that Maximum Matching can be solved in $\mathcal{O}(m\sqrt{n})$-time on general graphs [41] — although we do not use this result directly in our paper. Furthermore, let $G = (V, E)$ be a graph and let $F \subseteq E$ be a matching of G. We call a vertex matched if it is incident to an edge of F, and exposed otherwise. Then, we define an F-augmenting path as a path where the two ends are exposed, and the edges belong alternatively to F and not to F. It is well-known and easy to check that, given an F-augmenting path P, the matching $E(P)\Delta F$ (obtained by symmetric difference on the edges) has larger cardinality than F.

Lemma 4 (Berge, [3]). *A matching F in $G = (V, E)$ is maximum if and only if there is no F-augmenting path.*

We also consider an intermediate matching problem, that was first introduced (informally) in [12, Sec. 5.1].

Problem 2 (Module Matching).

Input: *A graph $G' = (V', E')$, for some $|V'| = p$, with the following additional information;*

- *a collection of subgraphs $\mathcal{P} = \{H_1, H_2, \ldots, H_p\}$, equipped with a bijection between V' and $\mathcal{P}$;*
- *a collection $\mathcal{F} = \{F_1, F_2, \ldots, F_p\}$,*
  with F_i being a maximum matching of H_i for every i.

Output: *A matching of $G = G'(\mathcal{P})$ with maximum cardinality.*

A natural choice for Module Matching would be to take $\mathcal{P} = \mathcal{M}(G)$. However, we will allow $\mathcal{P}$ to take different values for our reduction rules.

Additional notations. Let $\langle G', \mathcal{P}, \mathcal{F} \rangle$ be any instance of Module Matching. The order of G', or equivalently the cardinality of $\mathcal{P}$, is denoted by p. For every $1 \leq i \leq p$ let $M_i = V(H_i)$ and let $n_i = |M_i|$ be the order of H_i. We denote $\delta_i = |E(M_i, \overline{M_i})|$ the size of the cut $E(M_i, \overline{M_i})$ with all the edges between M_i and $N_G(M_i)$. In particular, we have $\delta_i = \sum_{v_j \in N_{G'}(v_i)} n_i n_j$. Let us define $\Delta m(G') = \sum_{i=1}^{p} \delta_i$. In the same way, let $\Delta\mu(G) = \mu(G) - \sum_{i=1}^{p} \mu(H_i)$. We will omit the dependency in, respectively, G' and G, if they are clear from the context.

Our framework is based on the following lemma (inspired from [12, Theorem 5.7]). We recall that a function f is called *superadditive* if we have $\forall x, y \ f(x) + f(y) \leq f(x + y)$.

Lemma 5. *Let $G = (V, E)$ be a graph. Suppose that for every H' in the modular decomposition of G it is possible to solve* Module Matching *on any instance $\langle H', \mathcal{P}, \mathcal{F} \rangle$ in time $T(p, \Delta m, \Delta\mu)$, where T is a superadditive function[2]. Then,* Maximum Matching *on G can be solved in time $\mathcal{O}(T(\mathcal{O}(n), m, n))$.*

Proof. Let N be the sum of the orders of all the subgraphs in the modular decomposition of G. We observe that each such a subgraph H' is the quotient subgraph $H/\mathcal{M}(H)$, for some H that is induced by a module of G. Next, we describe an algorithm for Maximum Matching that runs in time $\mathcal{O}(T(N, m, \mu(G)))$. The latter will prove the lemma since we have $N = \mathcal{O}(n)$ [32]. We prove our result by induction on the number of subgraphs in the modular decomposition of G. There are two cases.

- If G is a singleton, *i.e.*, $N = n = 1$, then we output an empty matching.
- Otherwise, let $\mathcal{M}(G) = \{M_1, M_2, \ldots, M_p\}$, and let $G' = (\mathcal{M}(G), E')$ be the quotient graph of G. For every $1 \leq i \leq p$, we call the algorithm recursively on $H_i = G[M_i]$ and we so obtain a maximum matching F_i for this subgraph. By the induction hypothesis, this step takes

[2]We stress that every polynomial function is superadditive, for an exponent ≥ 1.

time $\mathcal{O}(\sum_{i=1}^{p} T(N_i, m_i, \mu(H_i)))$, with N_i being the sum of the orders of all the subgraphs in the modular decomposition of H_i and $m_i = |E(H_i)|$. Furthermore, let $\mathcal{F} = \{F_1, F_2, \ldots, F_p\}$. Observe that we have $\sum_{i=1}^{p} N_i = N - p$, $\sum_{i=1}^{p} m_i = m - \Delta m$ and $\sum_{i=1}^{p} \mu(H_i) = \mu(G) - \Delta\mu$. In order to compute a maximum matching for G, we are left with solving Module Matching on $\langle G', \mathcal{M}(G), \mathcal{F}\rangle$, that takes time $T(p, \Delta m, \Delta\mu)$ by the hypothesis. Overall, since T is superadditive, the total running time is an $\mathcal{O}(T(N, m, \mu(G)))$.

□

An important observation for our subsequent analysis is that, given any module M of a graph G, the internal structure of $G[M]$ has no more relevance after we computed a maximum matching F_M for this subgraph. More precisely, we will use the following lemma:

Lemma 6 (Lemma 5.2 in [12])**.** *Let M be a module of $G = (V, E)$, let $G[M] = (M, E_M)$ and let $F_M \subseteq E_M$ be a maximum matching of $G[M]$. Then, every maximum matching of $G'_M = (V, (E \setminus E_M) \cup F_M)$ is a maximum matching of G.*

By Lemma 6 we can modify our algorithmic framework as follows. For every instance $\langle G', \mathcal{P}, \mathcal{F}\rangle$ for Module Matching, we can assume that $H_i = (M_i, F_i)$ for every $1 \leq i \leq p$. Finally, a *canonical ordering* of H_i (w.r.t. F_i) is a total ordering over $V(H_i)$ such that the exposed vertices appear first, and every two vertices that are matched together are consecutive. We make intensive use of these orderings in what follows.

2.2.1. Data structures and basic operations

Our data structure to handle a matching is a linear-size global array, indexed by the vertices of the input graph G. If a vertex v is exposed, then its cell in the array is initialized to `Null`, otherwise it contains a pointer to the unique vertex matched with v. Doing so, we can make the following standard assumption:

Property 1. Let F be a (not necessarily maximum) matching for the substitution $G = G'(\mathcal{P})$. For every $v \in V(G)$, we can decide in constant-time whether v is matched by F, and if so, we can also access in constant-time to the vertex matched with v.

The canonical ordering of a module M_i, w.r.t. some internal matching F_i, is simply stored in a doubly-linked list, for which we assume constant-time access to the head and to the tail. Each cell in the list stores an auxiliary integer variable: equal to either 0 (exposed), -1 (matched to the predecessor vertex) or 1 (matched to the successor vertex). Doing so, when we traverse the ordering, we can detect the separation between exposed and matched vertices in $\mathcal{O}(1)$ time.

The bottom-up traversal of the modular decomposition tree, presented in Lemma 5, allows us to initialize all the canonical orderings "on the fly" before we start processing a quotient subgraph H'. Specifically, recall that $H' = H/\mathcal{M}(H)$ for some subgraph H of the input G (induced by a module of G). We need to compute a canonical ordering for every module $M_i \in \mathcal{M}(H)$ w.r.t. some pre-computed matching F_i. For that, it is sufficient to enumerate all vertices of M_i, using Property 1 in order to identify the exposed vertices and the edges of F_i. By Lemma 3, it takes $\mathcal{O}(\sum_i |M_i|)$ time. If furthermore, H' has no isolated vertex (*i.e.*, there is no isolated module in H), then the scanning of all the modules can be done in $\mathcal{O}(\sum_i |M_i|) = \mathcal{O}(\Delta m)$ time. By Theorem 1, the only quotient subgraphs H' with isolated vertices are edgeless. In this situation, there is no need

processing H', and so there is no need computing the canonical orderings either. Hence, the overall time for computing all the canonical orderings, throughout the whole algorithm, is in $\mathcal{O}(n+m)$.

Finally, we stress that $\mathcal{M}(H)$, and so, the canonical orderings computed for H', partitions the vertices of H. We store, for every vertex v, the corresponding cell in the unique canonical ordering containing v (doubly-linked list) in some global linear-size array which is indexed by V. In particular, when we compute a canonical ordering, we simply update the pointers and the auxiliary integer variable that are stored in the (permanent) cells of this global array. The advantage of having at hands this global array is that, after we computed a canonical ordering for a module M_i, for any vertex $u \in M_i$, we can decide in constant-time whether it is exposed or matched in F_i, with F_i the internal matching that is associated to the ordering.

We summarize the properties of this above data structure:

Property 2. For every instance $\langle G', \mathcal{P}, \mathcal{F} \rangle$ for Module Matching, we have access to a canonical ordering for every $1 \leq i \leq p$.

Furthermore, for every $1 \leq i \leq p$ and $u \in M_i$, in $\mathcal{O}(1)$ time we can either assert that vertex u is left exposed by F_i or compute the vertex that is matched to u in this matching.

3. A pruned modular decomposition

In this section, we introduce a pruning process over the quotient subgraphs, that we use in order to refine the modular decomposition.

Definition 1. Let $G = (V, E)$ be a graph. We call $v \in V$ a one-vertex extension if it falls in one of the following cases:

- $N_G[v] = V$ (*universal*) or $N_G(v) = \emptyset$ (*isolated*);
- $N_G[v] = V \setminus u$ (*anti-pendant*) or $N_G(v) = \{u\}$ (*pendant*), for some $u \in V \setminus v$;
- $N_G[v] = N_G[u]$ (*true twin*) or $N_G(v) = N_G(u)$ (*false twin*), for some $u \in V \setminus v$.

A pruned subgraph of G is obtained from G by sequentially removing one-vertex extensions (in the current subgraph) until it can no more be done. This terminology was introduced in [36], where they only considered the removals of twin and pendant vertices. Also, the clique-width of graphs that are totally decomposed by the above pruning process (*i.e.*, with their pruned subgraph being a singleton) was studied in [45] [3]. First, we show that the gotten subgraph is "almost" independent of the removal ordering, *i.e.*, there is a unique pruned subgraph of G (up to isomorphism). The latter can be derived from the following (easy) lemma:

Lemma 7. *Let $G = (V, E)$ be a graph and let $v, v' \in V$ be one-vertex extensions of G. If v, v' are* not *pairwise twins then v' is a one-vertex extension of $G \setminus v$.*

Proof. We need to consider several cases. If v' is either isolated or universal in G then it stays so in $G \setminus v$. If v' is pendant in G then it is either pendant or isolated in $G \setminus v$. Similarly, if v' is anti-pendant in G then it is either anti-pendant or universal in $G \setminus v$. Otherwise, v' has a twin u in G. By the hypothesis, $u \neq v$. Then, we have that u, v' stay pairwise twins in $G \setminus v$. □

[3] They also considered anti-twins in [45]. Their integration to this framework remains to be done.

Corollary 1. *Every graph $G = (V, E)$ has a unique pruned subgraph up to isomorphism.*

Proof. Suppose for the sake of contradiction that G has two non-isomorphic pruned subgraphs. W.l.o.g., G is a minimum counter-example. In particular, for every one-vertex extension v of G, we have that $G \setminus v$ has a unique pruned subgraph up to isomorphism. Therefore, there exist $v, v' \in V$ such that: v, v' are one-vertex extensions of G, and the pruned subgraphs of $G \setminus v$ and $G \setminus v'$ are non isomorphic. We claim that at least one of the following must hold: v is *not* a one-vertex extension of $G \setminus v'$, or v' is *not* a one-vertex extension of $G \setminus v$. Indeed, otherwise, both the pruned subgraphs of $G \setminus v$ and of $G \setminus v'$ would be isomorphic to the pruned subgraph of $G \setminus \{v, v'\}$. By Lemma 7, it implies that v, v' are pairwise twins in G. However, since $G \setminus v$ and $G \setminus v'$ are isomorphic, so are their respective pruned subgraphs. A contradiction. □

For many graph classes a pruning sequence can be computed in linear-time. We observe that the same can be done for any graph (up to a logarithmic factor).

Proposition 1. *For every $G = (V, E)$, a pruned subgraph, and a corresponding pruning sequence, can be computed in $\mathcal{O}(n + m \log n)$-time.*

Proof. By Corollary 1, we are left with greedily searching for, then eliminating, the one-vertex extensions. We can compute the ordered degree sequence of G in $\mathcal{O}(n + m)$-time. Furthermore, after any vertex v is eliminated, we can update this sequence in $\mathcal{O}(|N(v)|)$-time. Hence, up to a total update time in $\mathcal{O}(n + m)$, at any step we can detect and remove in constant-time any vertex that is either universal, isolated, pendant or anti-pendant. Finally, in [36] they proposed a trie data-structure supporting the following two operations: suppression of a vertex; and detection of true or false twins (if any). The total time for all the operations on this data-structure is in $\mathcal{O}(n + m \log n)$ [36]. □

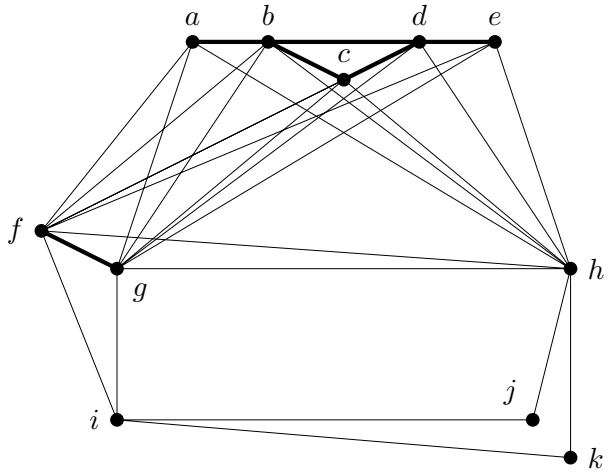
We will term "pruned modular decomposition" of a graph G the collection of the pruned subgraphs for all the quotient subgraphs in the modular decomposition of G (see Fig. 3 for an illustration). Note that there is a unique pruned modular decomposition of G up to isomorphism and that it can be computed in $\mathcal{O}(n + m \log n)$-time by Proposition 1 (applied to every quotient subgraph in the modular decomposition separately). Furthermore, we remark that most cases of one-vertex extensions imply the existence of non trivial modules, and so, they cannot exist in the prime quotient subgraphs of the modular decomposition. Nevertheless, such vertices may appear after removal of pendant or anti-pendant vertices, *e.g.*, in the bull graph.

3.1. Data structures and basic operations

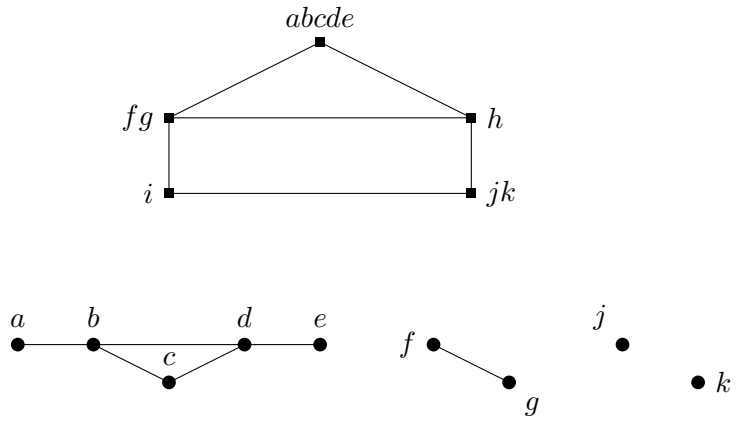
Let H be any graph, H^{pr} be a pruned subgraph of H, and $(v_1, v_2, \ldots, v_q)$ be a corresponding pruning sequence. Let $H_0 := H$ and, for every $1 \leq j \leq q$, let $H_j := H \setminus \{v_1, v_2, \ldots, v_j\}$. An efficient implementation of our reduction rules in this paper (see the next Sec. 4) requires the following two dual operations to be executed on all the quotient subgraphs of the modular decomposition:

- For every $1 \leq j \leq q$, construct $H_j = H_{j-1} \setminus \{v_j\}$ from H_{j-1};
- For every $1 \leq j \leq q$, construct H_{j-1} from $H_j = H_{j-1} \setminus \{v_j\}$.

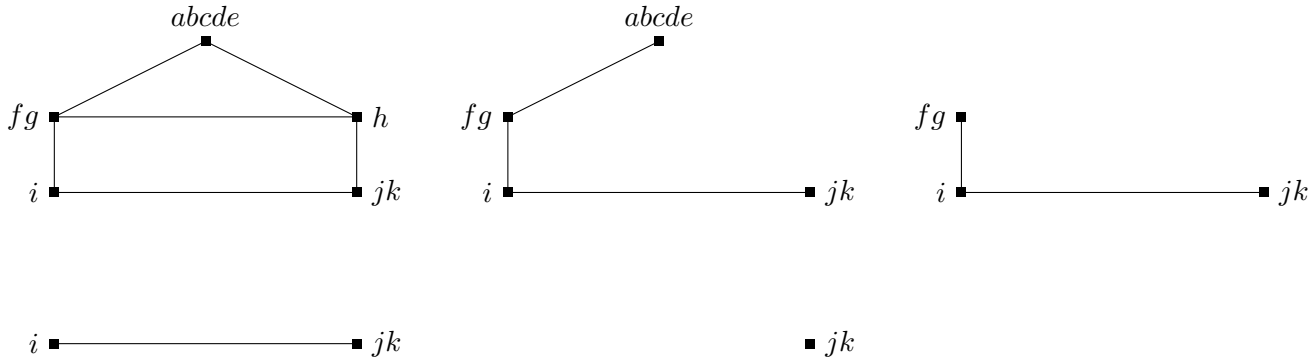
If we order the adjacency list so that $v_1, v_2, \ldots, v_q$ appear first, then both operations can be performed in $\mathcal{O}(deg_{H_{j-1}}(v_j))$ time, and so, in total $\mathcal{O}(|V(H)| + |E(H)|)$ time. Ordering the adjacency list, being given some ordering of the vertex-set, requires $\mathcal{O}(|V(H)| + |E(H)|)$ time and space.



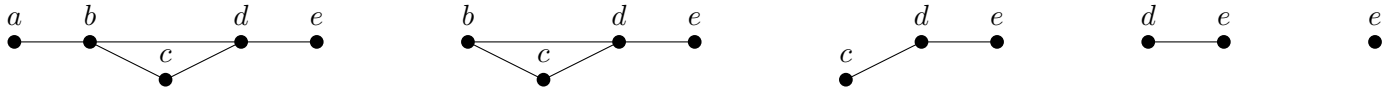
(a) Example of graph G totally decomposable by the pruned modular decomposition.



(b) Quotient subgraphs in the modular decomposition of G (with > 1 vertices).



(c) Pruning the house.



(d) Pruning the bull.

Figure 3: An illustration of the pruned modular decomposition

4. Reduction rules

Let $\langle G', \mathcal{P}, \mathcal{F} \rangle$ be any instance of Module Matching. Suppose that v_1, the vertex corresponding to M_1 in G', is a one-vertex extension. In order to avoid handling explicitly with some degenerate situations, we always assume in what follows $p = |\mathcal{P}| \geq 3$ (if $p \leq 2$, then we rely on a previous result from [12] in order to handle with quotient subgraphs of constant size; see Sec. 4.4). Under these assumptions, we present reduction rules to a smaller instance $\langle G^*, \mathcal{P}^*, \mathcal{F}^* \rangle$ where $|\mathcal{P}^*| < |\mathcal{P}|$. Set $\Delta m(G', G^*) = \Delta m(G') - \Delta m(G^*)$. Each of our rules can be implemented to run in $\mathcal{O}(\Delta m(G', G^*))$-time. In Section 4.1 we recall the rules introduced in [12, Sec. 5.2] for universal and isolated modules (explicitly) and for false or true twin modules (implicitly). Our

main technical contributions are the reduction rules for pendant and anti-pendant modules (in Sections 4.2 and 4.3, respectively), which are surprisingly the most intricate.

4.1. Simple cases

We introduce two local operations on a matching, first used in [48] for cographs. Let $F \subseteq E$ be a matching and let $M \subseteq V$ be a module.

Operation 1 (MATCH)**.** While there are $x \in M,\ y \in N(M)$ exposed, add $\{x, y\}$ to F.

Operation 2 (SPLIT)**.** While there are $x, x' \in M,\ y, y' \in N(M)$ such that x and x' are exposed, and $\{y, y'\} \in F$, replace $\{y, y'\}$ in F by $\{x, y\},\ \{x', y'\}$.

Let $G = H_1 \oplus H_2$ be the join[4] of the two graphs H_1, H_2 and let F_1, F_2 be maximum matchings for H_1, H_2, respectively. The "MATCH and SPLIT" technique consists in applying Operations 1 then 2 to $M = V(H_1)$ and $F = F_1 \cup F_2$, thereby obtaining a new matching F', then to $M = V(H_2)$ and $F = F'$. We observe that the cardinality of this matching can be computed in $\mathcal{O}(1)$ time. In fact, as pointed out to us by a reviewer, if $|M_1| \leq |M_2|$ then the final matching has cardinality $\min\{\lfloor(|M_1| + |M_2|)/2\rfloor, |M_1| + |F_2|\}$. However, some of our reduction rules require to *know* the matching instead of just its cardinality. We postpone our complexity analysis of this 'MATCH and SPLIT" technique to Sec. 4.1.1. Based on the latter, we design the following rules:

Reduction rule 1 (see also *Reduction rules* in [12], Sec. 5.2)**.** Suppose v_1 is isolated in G'. We set $G^* = G' \setminus v_1$, $\mathcal{P}^* = \mathcal{P} \setminus \{H_1\}$, and $\mathcal{F}^* = \mathcal{F} \setminus \{F_1\}$. Furthermore, let F^* be a maximum matching of $G^*(\mathcal{P}^*) = G[V \setminus M_1]$. We output $F^* \cup F_1$.

Reduction rule 2 (see also *Reduction rules* in [12], Sec. 5.2)**.** Suppose v_1 is universal in G'. We set $G^* = G \setminus v_1$, $\mathcal{P}^* = \mathcal{P} \setminus \{H_1\}$, $\mathcal{F}^* = \mathcal{F} \setminus \{F_1\}$. Furthermore, let F^* be a maximum matching of the substitution $G^*(\mathcal{P}^*) = G[V \setminus M_1]$. We apply the "MATCH and SPLIT" technique to M_1, F_1 with $V \setminus M_1, F^*$.

Reduction rule 3. Suppose v_1, v_2 are false twins in G'. We set $G^* = G' \setminus v_1$, $\mathcal{P}^* = \{H_1 \cup H_2\} \cup (\mathcal{P} \setminus \{H_1, H_2\})$, $\mathcal{F}^* = \{F_1 \cup F_2\} \cup (\mathcal{F} \setminus \{F_1, F_2\})$. We output a maximum matching of $G^*(\mathcal{P}^*) = G$.

Reduction rule 4. Suppose v_1, v_2 are true twins in G'. Let F_2^* be the matching of $H_1 \oplus H_2$ obtained from the "MATCH and SPLIT" technique applied to M_1, F_1 with M_2, F_2. We set $G^* = G \setminus v_1$, $\mathcal{P}^* = \{H_1 \oplus H_2\} \cup (\mathcal{P} \setminus \{H_1, H_2\})$, $\mathcal{F}^* = \{F_2^*\} \cup (\mathcal{F} \setminus \{F_1, F_2\})$. We output a maximum matching of $G^*(\mathcal{P}^*) = G$.

Reduction rules 1 and 3 are straightforward. The correctness of Reduction rules 2 and 4 can be readily proved from the following result:

Lemma 8 (Lemma 5.13 in [12])**.** *Let $G = G_1 \oplus G_2$ be the join of two graphs G_1, G_2 and let F_1, F_2 be maximum matchings for G_1, G_2, respectively. For $F = F_1 \cup F_2$, applying the "*MATCH *and* SPLIT*" technique to $V(G_1)$, then to $V(G_2)$ leads to a maximum matching of G.*

[4]We recall that the join of two graphs G_1, G_2 is obtained from these two graphs by adding all possible edges between $V(G_1)$ and $V(G_2)$.

4.1.1. Complexity analysis

In Reduction rules 1, 2, 3 and 4 we replace G' by $G^* := G' \setminus v_1$. Up to some linear-time pre-processing for ordering the adjacency list, the removal of v_1 can be done in $\mathcal{O}(deg_{G'}(v_1))$ time, and so in $\mathcal{O}(\Delta m(G', G^*))$ (*i.e.*, see Sec. 3.1). From now on, we exclude the cost of removing vertex v_1 from our complexity analysis.

- In particular, Reduction rule 1 takes $\mathcal{O}(1)$-time.
- Reduction rule 3 also takes $\mathcal{O}(1)$ time, plus the cost of computing a canonical ordering for $M_1 \cup M_2$ w.r.t. $F_1 \cup F_2$. For that, using Property 2, we scan the canonical ordering of M_1 until there is no more exposed vertex. Then, we make of the last exposed vertex in this ordering (if any) the predecessor of the first vertex in the canonical ordering of M_2; in the same way, we make of the first matched vertex in this ordering (if any) the successor of the last vertex in the canonical ordering of M_2. – Note that conversely, if we store a copy of the canonical ordering of M_1, then during a post-processing phase, the original canonical ordering of M_2 can be retrieved. – The running time is in $\mathcal{O}(n_1)$. Since we can further assume v_1, v_2 are *not* isolated in G' (otherwise, we apply Reduction rule 1), then we have $n_1 = \mathcal{O}(\Delta m(G', G^*))$.
- Furthermore, as explained in Section 2.2.1, for all the matchings considered we assume their so called "canonical ordering" to be given (Property 2). Then, the complexity of Reduction rule 2 is dominated by the Match and Split operations. Specifically, we scan the canonical orderings of M_1 and of the modules in its neighbourhood (*i.e.*, there are $\mathcal{O}(\Delta m(G', G^*))$ different canonical orderings to access). We can use these canonical orderings in order to output the desired exposed vertices, resp. the desired matched edges, for applying our rules. Doing so, every such operation adds, in $\mathcal{O}(1)$-time, one or two edges in the matching with one end in M_1. Observe that there cannot be more than $\mathcal{O}(n_1)$ operations. Furthermore, $\Delta m(G') - \Delta m(G^*) \geq n_1 n_2 = \Omega(n_1)$. Hence, Reduction rule 2 takes $\mathcal{O}(\Delta m(G', G^*))$-time.
- Finally, the complexity analysis of Reduction rule 4 is similar to the one above for Reduction rule 2, plus the time needed for computing a canonical ordering for $M_1 \cup M_2$ w.r.t. the matching F_2^*. For that, using Property 1, it is sufficient to scan $M_1 \cup M_2$, that can be done in $\mathcal{O}(n_1 + n_2)$ time by Lemma 3. – Note that, within the same amount of time, we can store a copy of the original canonical orderings of M_1, M_2. – Recall that, since $v_1 v_2$ is an edge of G', we have $n_1 + n_2 = \mathcal{O}(n_1 n_2) = \mathcal{O}(\Delta m(G', G^*))$.

4.2. Anti-pendant

Suppose v_1 is anti-pendant in G'. W.l.o.g., v_2 is the unique vertex that is nonadjacent to v_1 in G'. By Lemma 6, we can also assume w.l.o.g. that $E(H_i) = F_i$ for every i. In this situation, we start applying Reduction rule 1, *i.e.*, we set $G^* = G' \setminus v_1$, $\mathcal{P}^* = \mathcal{P} \setminus \{H_1\}$, $\mathcal{F}^* = \mathcal{F} \setminus \{F_1\}$. Then, we obtain a maximum matching F^* of $G \setminus M_1$ (*i.e.*, by applying our reduction rules to this new instance). Finally, from F_1 and F^*, we compute a maximum matching F of G, using an intricate procedure. We detail this procedure next.

First phase: pre-processing. Our correctness proofs in what follows will assume that some additional properties hold on the matched vertices in F^*. So, we start correcting the initial matching F^* so that it is the case. For that, we introduce two "swapping" operations. Recall that v_2 is the unique vertex that is nonadjacent to v_1 in G'.

Operation 3 (REPAIR)**.** While there exist $x_2, y_2 \in M_2$ such that $\{x_2, y_2\} \in F_2$ and y_2 is exposed in F^*, we replace any edge $\{x_2, w\} \in F^*$ by $\{x_2, y_2\}$.

Let $F' := REPAIR(F^*)$.

Operation 4 (ATTRACT)**.** While there exist $x_2 \in M_2$ exposed and $\{u, w\} \in F'$ such that $u \in N_G(M_2), w \notin M_2$, we replace $\{u, w\}$ by $\{u, x_2\}$.

Let $F'' = ATTRACT(F')$ and $F^{(0)} = F_1 \cup F''$. Summarizing, we get:

Definition 2. A matching F of G is *good* if it satisfies the following two properties:

1. every vertex matched by $F_1 \cup F_2$ is also matched by F;
2. either every vertex in M_2 is matched, or there is no matched edge in $N_G(M_2) \times N_G(M_1)$.

Fact 1. *$F^{(0)}$ is a good matching of G.*

Indeed, the first and second conditions are enforced, respectively, by Operations 3 and 4.

Main phase: a modified Match **and** Split**.** We now apply the following three operations sequentially:

1. Match$(M_1, F^{(0)})$ (Operation 1). Doing so, we obtain a larger good matching $F^{(1)}$.
2. Split$(M_1, F^{(1)})$ (Operation 2). Doing so, we obtain a larger good matching $F^{(2)}$.
3. the operation Unbreak, defined in what follows (see also Fig. 4 for an illustration):

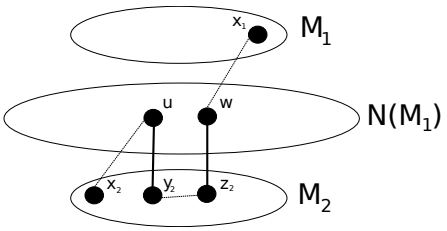


Figure 4: An augmenting path of length 5 with ends x_1, x_2. Matched edges are drawn in bold.

Operation 5 (Unbreak)**.** While there exist $x_1 \in M_1$ and $x_2 \in M_1 \cup M_2$ exposed, and there also exist $\{y_2, z_2\} \in F_2 \setminus F^{(2)}$, we replace any two edges $\{y_2, u\}, \{z_2, w\} \in F^{(2)}$ by the three edges $\{x_2, u\}, \{y_2, z_2\}$ and $\{w, x_1\}$.

We will prove below that $F^{(2)}$ is a good matching of G (Claim 1), and so, the two edges $\{y_2, u\}, \{z_2, w\} \in F^{(2)}$ that are required for the operation Unbreak always exist. Furthermore doing so, we obtain a larger matching $F^{(3)}$.

The resulting matching $F^{(3)}$ is not necessarily maximum. However, this matching satisfies the following crucial property:

Lemma 9. *No vertex of M_1 can be an end in an $F^{(3)}$-augmenting path.*

Proof. Let $x_1 \in M_1$ be exposed. Suppose by contradiction x_1 is an end of some $F^{(3)}$-augmenting path $P = (x_1 = u_1, u_2, \ldots, u_{2\ell})$. W.l.o.g., P is of minimum length. We will derive a contradiction from the following invariants:

Claim 1. *The following properties hold for every* $0 \leq j \leq 3$*:*

1. $F^{(j)}$ *is a good matching of* G*;*
2. *If* $u_{2i}, u_{2i+1} \in N_G(M_1)$ *and* $\{u_{2i}, u_{2i+1}\} \in F^{(3)}$ *then we also have* $\{u_{2i}, u_{2i+1}\} \in F^{(j)}$*;*
3. $F_1 \subseteq F^{(j)}$*.*

Proof. Since we only increase the successive matchings using augmenting paths, we keep the property that $V(F_1 \cup F_2) \subseteq V(F^{(j)})$. In fact, since we only consider the exposed vertices in M_1 for our operations, we have the stronger Property 3 that $F_1 \subseteq F^{(j)}$. Furthermore, our successive operations do not create any new exposed vertex in M_2 nor any new matched edge in $N_G(M_1) \times N_G(M_1)$, and so, both Properties 1 and 2 also hold. ◇

In what follows, we divide the proof in three claims.

Claim 2. $u_{2\ell} \in M_1 \cup M_2$*.*

Proof. Suppose for the sake of contradiction $u_{2\ell} \notin M_1 \cup M_2$, or equivalently $u_{2\ell} \in N(M_1)$. Then, we could have continued the first step Match$(M_1, F^{(0)})$ by matching $x_1, u_{2\ell}$ together, that is a contradiction. ◇

Next, we derive a contradiction by proving $u_{2\ell} \notin M_1 \cup M_2$.

Claim 3. $u_{2\ell} \notin M_1$*.*

Proof. Suppose for the sake of contradiction $u_{2\ell} \in M_1$. There are two cases.

1. Case $u_2, u_3 \in N(M_1)$. Since $\{u_2, u_3\} \in F^{(3)}$ we have by Claim 1 $\{u_2, u_3\} \in F^{(1)}$. In particular, we could have replaced $\{u_2, u_3\}$ by $\{u_2, x_1\}, \{u_3, u_{2\ell}\}$ during the second step of the main phase (*i.e.*, Split$(M_1, F^{(1)})$), that is a contradiction.
2. Thus, let us now assume $u_2 \in N(M_1)$ (necessarily) but $u_3 \notin N(M_1)$. By minimality of P, we have $u_4 \notin N(M_1)$ (otherwise, $P' = (x_1, u_4, u_5, \ldots, u_{2\ell})$ would be a shorter augmenting path than P). We claim that it implies $u_3 \notin M_1$. Indeed, otherwise we should also have $u_4 \in M_1$, and so, $\{u_3, u_4\} \in F_1$ since we assume that M_1 induces a matching. However, $\{u_3, u_4\} \notin F^{(3)}$, whereas we have by Claim 1 that $F_1 \subseteq F^{(3)}$. A contradiction. Therefore, as claimed, $u_3 \notin M_1$. We deduce from the above that $u_3 \in M_2$. Similarly, since $u_4 \notin N(M_1) \supseteq N(M_2)$ we get $u_4 \in M_2$. Altogether combined (and since M_2 induces a matching), $\{u_3, u_4\} \in F_2 \setminus F^{(3)}$. Then, we could have continued the step Unbreak with $x_1 \in M_1$ and $u_{2\ell} \in M_1$ exposed, and $\{u_3, u_4\} \in F_2 \setminus F^{(3)}$, that is a contradiction.

As a result, $u_{2\ell} \notin M_1$. ◇

Claim 4. $u_{2\ell} \notin M_2$*.*

Proof. Suppose for the sake of contradiction $u_{2\ell} \in M_2$. First we prove $u_2, u_3 \in N(M_1)$. Indeed, since u_1 is exposed and $F_1 \subseteq F^{(3)}$ by Claim 1 we have that $u_2 \in N(M_1)$. Furthermore, if $u_3 \notin N(M_1)$ then we could prove as before (Claim 3, Case 2) $\{u_3, u_4\} \in F_2 \setminus F^{(3)}$; it implies that we could have continued the step Unbreak with $x_1 \in M_1$ and $u_{2\ell} \in M_2$ exposed, and $\{u_3, u_4\} \in F_2 \setminus F^{(3)}$, that is a contradiction. Therefore, as claimed, $u_2, u_3 \in N(M_1)$.

Now, consider the edge $\{u_{2\ell-2}, u_{2\ell-1}\} \in F^{(3)}$. Since $u_{2\ell} \in M_2$ is exposed and by Claim 1 we have $V(F_2) \subseteq V(F^{(3)})$, $u_{2\ell} \notin V(F_2)$. Furthermore, since $E(H_2) = F_2$ we have $u_{2\ell-1} \notin M_2$. There are two cases.

1. Suppose $u_{2\ell-2} \in M_1$. Since $u_{2\ell-2}$ is matched to $u_{2\ell-1} \notin M_1$ and $F_1 \subseteq F^{(3)}$ by Claim 1, we have that $u_{2\ell-2} \notin V(F_1)$. Furthermore, we claim that the edge $\{u_{2\ell-2}, u_{2\ell-1}\}$ was added to the matching during the second step of the main phase (*i.e.*, $\textsc{Split}(M_1, F^{(1)})$).
   In order to prove this subclaim, we only need to decide the first step where $u_{2\ell-2}$ was matched to any vertex; indeed, our above operations can only consider vertices in M_1 that are exposed. We observe that $u_{2\ell-2}, u_{2\ell-1}$ could not possibly be matched together during the first step since otherwise, we could have also matched $u_{2\ell-1}$ with $u_{2\ell}$, thereby contradicting that F'' is a maximum-cardinality matching of $G \setminus M_1$. In addition, recall that we proved above $u_2, u_3 \in N(M_1)$. By Claim 1, $\{u_2, u_3\} \in F^{(1)}$. It implies that $u_{2\ell-2}, u_{2\ell-1}$ were matched together during the second step since, otherwise, this second step could have continued with $x_1, u_{2\ell-2} \in M_1$ exposed and $\{u_2, u_3\} \in F^{(1)}$. Therefore, the subclaim is proved.
   Then, before the second step of the main phase happened, vertex $u_{2\ell-1}$ was matched to some other vertex in $N_G(M_1)$. However, since $u_{2\ell-1} \in N_G(M_2)$ and $u_{2\ell} \in M_2$ is exposed the latter contradicts that $F^{(1)}$ is good, and so, Claim 1.
2. Thus from now on assume $u_{2\ell-2} \notin M_1$. By Claim 1 we have that $F^{(3)}$ is good, and so, since $u_{2\ell-1} \in N_G(M_2)$ and $u_{2\ell} \in M_2$ is exposed we have $u_{2\ell-2} \in M_2$. Furthermore, $u_{2\ell-3} \notin M_2$ since, otherwise, the final step of the main phase (UNBREAK) could have continued with $x_1 \in M_1$ and $u_{2\ell} \in M_2$ exposed, and $\{u_{2\ell-3}, u_{2\ell-2}\} \in F_2 \setminus F^{(3)}$, that is a contradiction. However, it implies that $P' = (x_1 = u_1, u_2, u_3, \ldots, u_{2\ell-3}, u_{2\ell})$ is a shorter augmenting path than P, thereby leading to another contradiction.

As a result, $u_{2\ell} \notin M_2$. ◇

Overall since $u_{2\ell} \in M_1 \cup M_2$ by Claim 2 but $u_{2\ell} \notin M_1 \cup M_2$ by Claims 3 and 4, the above proves that x_1 cannot be an end in any $F^{(3)}$-augmenting path. □

Finalization phase: breaking some edges in F_1. Intuitively, the matching $F^{(3)}$ may not be maximum because we sometimes need to borrow some edges of F_1 in augmenting paths. So, we complete our procedure by performing the following two operations: Let U_1 contain all the exposed vertices in $N(M_1)$. Consider the subgraph $G[M_1 \cup U_1] = G[M_1] \oplus G[U_1]$. The set U_1 is a module of this subgraph. We apply $\textsc{Split}(U_1, F^{(3)})$ in $G[M_1 \cup U_1]$. Doing so, we obtain a larger good matching $F^{(4)}$. Then, we apply LOCALAUG, defined next (see also Fig. 5 for an illustration):

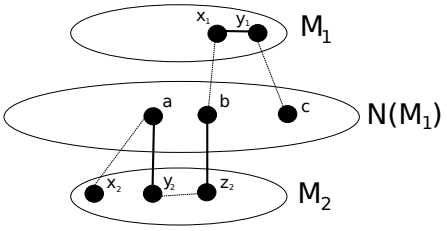


Figure 5: An augmenting path of length 7 with ends x_2, c. Matched edges are drawn in bold.

Operation 6 (LOCALAUG)**.** While there exist $x_2 \in M_2$ and $c \in N(M_1)$ exposed, and there also exist $\{x_1, y_1\} \in F_1 \cap F^{(4)}$ and $\{y_2, z_2\} \in F_2 \setminus F^{(4)}$, we do the following:

- we remove $\{x_1, y_1\}$ and any edge $\{a, y_2\}, \{b, z_2\}$ from $F^{(4)}$;
- we add $\{x_2, a\}, \{y_2, z_2\}, \{b, x_1\}$ and $\{y_1, c\}$ in $F^{(4)}$.

We will observe below that $F^{(4)}$ is a good matching of G (Fact 2). In particular, the two edges $\{y_2, a\}, \{z_2, b\} \in F^{(4)}$ that are required for the operation LocalAug always exist. Furthermore doing so, we obtain a larger matching $F^{(5)}$.

Lemma 10. *$F^{(5)}$ is a maximum-cardinality matching of G.*

Proof. Suppose for the sake of contradiction that there exists an $F^{(5)}$-augmenting path $P = (u_1, u_2, \ldots, u_{2\ell})$. W.l.o.g., P is of minimum size. We start with two useful facts.

Fact 2. *$F^{(4)}$ is a good matching of G.*

Fact 3. *$F^{(5)}$ is a good matching of G.*

Indeed, since we obtain $F^{(4)}$, then $F^{(5)}$, from the good matching $F^{(3)}$ by using augmenting paths, necessarily $V(F_2) \subseteq V(F^{(4)}) \subseteq V(F^{(5)})$ (first condition of Def. 2). Furthermore, neither the Split operations between M_1 and U_1, resp. Operation 6, can create a new matched edge within $N_G(M_2) \times N_G(M_1)$; finally, we cannot create a new exposed vertex in M_2 (second condition of Def. 2). Then, we divide the proof into the following claims.

Claim 5. $u_1, u_{2\ell} \notin M_1$

Proof. Suppose for the sake of contradiction $u_1 \in M_1$ (the case $u_{2\ell} \in M_1$ is symmetrical to this one). Since $F^{(3)}$ and $F^{(5)}\Delta P$ are matchings, the symmetric difference $F^{(3)}\Delta(F^{(5)}\Delta P)$ is a disjoint union of alternating cycles, alternating paths and isolated vertices. In particular, since $F^{(5)}\Delta P$ can be obtained from $F^{(3)}$ by using augmenting paths, the symmetric difference $F^{(3)}\Delta(F^{(5)}\Delta P)$ is exactly a disjoint union of isolated vertices and of augmenting paths that can be used for obtaining $F^{(5)}\Delta P$ from $F^{(3)}$. One of these paths must contain u_1. As a result, there is also an $F^{(3)}$-augmenting path with an end in M_1, thereby contradicting Lemma 9. The latter proves, as claimed, $u_1, u_{2\ell} \notin M_1$. $\diamond$

Claim 6. *There is no exposed vertex in $N(M_1)$.*

Proof. Suppose for the sake of contradiction $N(M_1) \not\subseteq V(F^{(5)})$. In particular $N(M_1) \not\subseteq V(F^{(4)})$. We will prove in this situation there can be only one vertex in $N(M_1)$ that is left exposed by $F^{(4)}$. Then, we will derive a contradiction by proving that we can apply the LocalAug operation.

First, we observe that the main phase of our procedure must terminate after its very first step Match$(M_1, F^{(0)})$. Indeed, after this step there can be no more exposed vertex in M_1 (*i.e.*, because otherwise we could have continued the Match operations using the exposed vertices of $N(M_1)$). Hence, the other rules of our main step cannot be applied. Then, we apply the operation Split$(U_1, F^{(3)})$ in $G[M_1 \cup U_1]$ in order to further match some vertices in $N(M_1)$ to the vertices in M_1. Doing so, we get the following two important properties for $F^{(4)}$:

1. if a vertex of $N(M_1)$ is matched to a vertex of M_1, this vertex was left exposed by F'';
2. $F'' \subseteq F^{(4)}$.

Let $Q = (w_1, w_2, \ldots, w_{2q})$ be a minimum-length $F^{(4)}$-augmenting path. Such path exist since $F^{(5)}$, and so, $F^{(4)}$, is not maximum. Let i_0 be the minimum index i such that $w_i \in M_1$. The latter is well-defined since otherwise, by the above Property 2 Q would be an F''-augmenting path in $G \setminus M_1$, thereby contradicting the maximality of F''. Furthermore, $w_1, w_{2q} \notin M_1$, and so, $i_0 > 1$ (the proof is the same as for Claim 5). More generally, we have that i_0 is even since, if it were not the case, by the above Property 1, we would have that $(w_1, w_2, \ldots, w_{i_0-1})$ is an F''-augmenting path in $G \setminus M_1$, thereby again contradicting the maximality of F''. There are two cases:

1. Case there exists an edge $\{x_1, y_1\} \in F_1 \cap F^{(4)}$. Then, there is exactly one exposed vertex $c \in N_G(M_1)$ (otherwise, the step $\textsc{Split}(U_1, F^{(3)})$ in $G[M_1 \cup U_1]$ could have been continued). W.l.o.g., $w_1 \neq c$ (otherwise, we start the augmenting path from w_{2q}). Since $w_1 \notin M_1$, it implies $w_1 \in M_2$. We consider the alternating subpath (w_1, w_2, w_3) in Q. Since $w_2 \in N_G(M_2)$, we get $w_2 \notin M_1$. Furthermore, since i_0 is even, we have $w_3 \notin M_1$ by minimality of i_0. Finally, since we have: $w_1 \in M_2$ is exposed, $w_2 \in N(M_2)$ is matched to w_3 and, by Fact 2, $F^{(4)}$ is good, we get that $w_3 \notin N(M_1)$. Altogether combined, $w_3 \in M_2$. We also have $w_4 \in M_2$ since otherwise, $Q' = (w_1, w_4, w_5, \ldots, w_{2q})$ would be a shorter augmenting path than Q, thereby contradicting the minimality of Q. As a result, $\{w_3, w_4\} \in F_2 \setminus F^{(4)}$. However, in this case there is at least one possibility for applying the operation LocalAug, namely: with $w_1 \in M_2$ and $c \in N(M_1)$ exposed, $\{x_1, y_1\} \in F_1 \cap F^{(4)}$ and $\{w_3, w_4\} \in F_2 \setminus F^{(4)}$. It implies that we do apply the operation LocalAug at least once, *i.e.*, $F^{(4)} \neq F^{(5)}$. In particular, c is matched by $F^{(5)}$ (because it is the only exposed vertex of $N(M_1)$ that we can use for Operation 6), that is a contradiction.
2. Case $F_1 \cap F^{(4)} = \emptyset$. In particular, $w_{i_0+1} \notin M_1$. Let j_0 be the maximum $j \geq i_0 + 1$ such that $w_{i_0+1}, w_{i_0+2}, \ldots, w_j \notin M_1$. We have $j_0 < 2q$ since otherwise, by the above Property 1, $(w_{i_0+1}, \ldots, w_{2q})$ would be an F''-augmenting path in $G \setminus M_1$, thereby contradicting the maximality of F''. Thus, $w_{j_0+1} \in M_1$. Furthermore, j_0 is even since otherwise, $Q' = (w_1, \ldots, w_{i_0-1}, w_{j_0+1}, \ldots, w_{2q})$ would be a shorter $F^{(4)}$-augmenting path than Q, thereby contradicting the minimality of Q. However, then we have by the above Property 1, $(w_{i_0+1}, \ldots, w_{j_0})$ that is an F''-augmenting path in $G \setminus M_1$, thereby contradicting the maximality of F''.

Overall, the above proves as claimed that there is no exposed vertex in $N(M_1)$. ◇

It follows from Claims 5 and 6 that $u_1, u_{2\ell} \in M_2$. Furthermore we have $u_1 \in M_2$ is exposed, $\{u_2, u_3\}$ is matched and $u_2 \in N_G(M_2)$. Since, by Fact 3, $F^{(5)}$ is good, we have $u_3 \notin N(M_1)$. Equivalently, $u_3 \in M_1 \cup M_2$. The following claim will be instrumental in deriving a contradiction.

Claim 7. $F_2 \subseteq F^{(5)}$.

Proof. Suppose for the sake of contradiction there exists $\{x_2, y_2\} \in F_2 \setminus F^{(5)}$. We prove that $F_2 \setminus F^{(5)} \subseteq F_2 \setminus F''$. Indeed, after the two first steps of the main phase (*i.e.*, the Match and Split operations between M_1 and $N(M_1)$) we have $F_2 \setminus F^{(2)} = F_2 \setminus F''$. The operation Unbreak adds edges of F_2 into the matching, hence $F_2 \setminus F^{(3)} \subseteq F_2 \setminus F''$. Then, after the operation $\textsc{Split}(U_1, F^{(3)})$ in $G[M_1 \cup U_1]$ we have $F_2 \setminus F^{(4)} = F_2 \setminus F^{(3)} \subseteq F_2 \setminus F''$. Finally, the operation LocalAug adds edges of F_2 into the matching, hence $F_2 \setminus F^{(5)} \subseteq F_2 \setminus F''$. However, since $F^{(0)}$ is good, we have $V(F_2) \subseteq V(F'')$. It implies there exist $w, w' \in N(M_2)$ such that $\{x_2, w\}, \{y_2, w'\} \in F''$. In particular we have that $(u_1, w, x_2, y_2, w', u_{2\ell})$ is an F''-augmenting path in $G \setminus M_1$, thereby contradicting the maximality of F''. ◇

Now, there are two cases.

- Case $u_3 \in M_2$. We have $u_4 \notin N(M_2)$ since otherwise, $P' = (u_1, u_4, u_5, \ldots, u_{2\ell})$ would be a shorter augmenting path than P, thereby contradicting the minimality of P. Therefore, $\{u_3, u_4\} \in F_2 \setminus F^{(5)}$. The latter contradicts Claim 7.
- Case $u_3 \in M_1$. By maximality of F'', u_2 was matched in F'' (otherwise, we could have added $\{u_1, u_2\}$ in F''). Therefore, the edge $\{u_2, u_3\}$ was not matched during the operation

MATCH$(M_1, F^{(0)})$ nor during the operation SPLIT$(U_1, F^{(3)})$ in $G[M_1 \cup U_1]$. Furthermore, this edge was not matched during the operation SPLIT$(M_1, F^{(1)})$ either since otherwise, u_2 would have been matched in $F^{(1)}$ with some other vertex in $N(M_1)$; since $u_1 \in M_2$ is exposed and $u_2 \in N(M_2)$, the latter would contradict that $F^{(1)}$ is good (Claim 1). As a result, the edge $\{u_2, u_3\}$ was matched during the UNBREAK operation or the LOCALAUG operation. Both subcases imply the existence of some edge $\{x_2, y_2\} \in F_2 \setminus F''$. As in the previous case, the latter contradicts the maximality of F''.

□

4.2.1. Complexity analysis

Each step of our procedure is corresponding to a while loop. We briefly review each step, and we explain how we can implement them in total $\mathcal{O}(\Delta m(G', G^*))$ time.

- Operation 3 (REPAIR): We scan all the vertices $w \in N_G(M_2)$. By Lemma 3 (or using the adjacency list of G' and the canonical orderings), this can be done in $\mathcal{O}(|N_G(M_2)|)$ time. If w is matched in F^*, then we access in constant-time to the vertex x_2 to which it is paired (Property 1). Furthermore, if $x_2 \in M_2$, then we check whether it is matched in F_2, and if it is the case, then we can access in constant-time to the vertex y_2 to which it is paired (Property 2). Note that in order to decide whether $x_2 \in M_2$, it is sufficient to mark all the vertices of $N_G(M_2)$ during a first scan of this neighbour set (we could also use the least-common ancestor approach of Lemma 2). Finally, if y_2 is exposed in F^*, then we perform a REPAIR operation, with x_2, y_2. Overall, this takes total time $\mathcal{O}(|N_G(M_2)|)$.

- Operation 4 (ATTRACT): Recall that we have access to a canonical ordering for F_2. In particular, we can decide in constant-time whether there exists an exposed vertex $x_2 \in M_2$ w.r.t. F_2, and if so we can output one also in constant time (Property 2). We scan all the vertices $w \in N_G(M_2)$ (in $\mathcal{O}(|N_G(M_2)|)$ time, using Lemma 3). If w is matched in F', then we access in constant-time to the vertex u to which it is paired (Property 1). Furthermore, if $u \notin M_2$ (that can be checked by marking all the vertices of $N_G(M_2)$ during a first scan), then we want to check whether there exists an exposed vertex of M_2, but w.r.t. F' (not F_2). For that, we scan the exposed vertices of M_2, w.r.t. F_2, until there is no more or we found one which is also exposed in F'. Note that it can only take $\mathcal{O}(|N_G(M_2)|)$ total time throughout the whole phase: indeed, if a vertex is exposed in F_2, but not in F', then it is matched with a vertex of $N_G(M_2)$. Overall, this takes total time in $\mathcal{O}(|N_G(M_2)|)$.

- Operations 1 and 2 (MATCH$(M_1, F^{(0)})$, SPLIT$(M_1, F^{(1)})$, SPLIT$(U_1, F^{(3)})$ in $G[M_1 \cup U_1]$): We explained in Sec. 4.1 how we can implement them in $\mathcal{O}(\Delta m(G', G^*))$ time.

- Operation 5 (UNBREAK): We explained before how, in *total* $\mathcal{O}(|N_G(M_2)|)$ time, we can have access to an exposed vertex of M_2 (if any). We can proceed similarly with M_1, *i.e.*, in total $\mathcal{O}(|N_G(M_1)|)$ time, we can have access to an exposed vertex of M_1. Therefore, up to an $\mathcal{O}(|N_G(M_1)|)$-time additional processing, the running time of this phase is the one for finding *unmatched* edges of F_2. As we cannot say much about the size of M_2, we need a trickier method in order to enumerate these edges. Namely, we scan all the vertices $w \in N_G(M_2)$ (again, this can be done in $\mathcal{O}(|N_G(M_2)|)$ time, using Lemma 3). If w is matched in F'', then we access in constant-time to the vertex x_2 to which it is paired (Property 1). Furthermore,

if $x_2 \in M_2$ (checkable in $\mathcal{O}(1)$ time if we first mark all the vertices of $N_G[M_1]$ during a first scan), then we check whether it is matched in F_2, and if it is the case, then we can access in constant-time to the vertex y_2 to which it is paired (Property 2). Finally, if y_2 is matched in F'', then we can access in constant time to the vertex u to which it is paired. Overall, this takes total time $\mathcal{O}(|N_G[M_1]|)$.

- Operation 6 (LocalAug): We can access, at any moment, to an exposed vertex of M_2, in *total* $\mathcal{O}(|N_G(M_2)|)$ time for the whole phase. In the same way, if we scan $N_G(M_1)$ once (in $\mathcal{O}(|N_G(M_1)|)$ time, using Lemma 3), then it becomes possible to have access in constant-time to an exposed vertex of $N_G(M_1)$. Now, in order to have access to an edge of $F_1 \cap F^{(4)}$ (if any), we use the canonical ordering of F_1. Note that from this point on in the algorithm, we *cannot* add an edge of F_1 in the matching if it were not already present. Therefore, it is sufficient to scan each edge of F_1 at most once, and so this takes total $\mathcal{O}(|M_1|)$ time for the whole phase. We are left with scanning the unmatched edges of F_2, which we explained how to do in total time $\mathcal{O}(|N_G(M_2)|)$.

Overall, the total running-time of the procedure is in $\mathcal{O}(|N_G[M_1]|)$, that is in $\mathcal{O}(\Delta m(G', G^*))$.

4.3. Pendant

Suppose v_1 is pendant in G'. W.l.o.g., v_2 is the unique vertex that is adjacent to v_1 in G'. This last case is arguably more complex than the others since it requires both a pre-processing and a post-processing treatment on the matching.

First phase: greedy matching. We apply the "Match and Split" technique to M_1. If there remains an exposed vertex $x_1 \in M_1$, and an edge $\{x_2, y_2\} \in F_2$, then we also add an edge $\{x_1, x_2\}$ to the current matching (thus, removing $\{x_2, y_2\}$ from F_2). We call the latter a "replacement operation", that is quite similar to Attract in Sec. 4.2. Doing so, we obtain a set $F_{1,2}$ of matched edges between M_1 and M_2. Note that this matching $F_{1,2}$ results from the 'Match and Split" technique applied to M_1, to which we possibly added one more edge $\{x_1, x_2\}$.

Our first result in this section is that there always exists an optimal solution that contains $F_{1,2}$.

Lemma 11. *There is a maximum matching of G that contains all edges in $F_{1,2}$.*

Proof. Let $M_1 = (u_1, u_2, \ldots, u_{n_1})$ and $M_2 = (w_1, w_2, \ldots, w_{n_2})$ be canonically ordered w.r.t. F_1, F_2 (cf. Sec. 2). Furthermore, let $u_1, u_2, \ldots, u_k$ be the maximal sequence of exposed vertices in M_1 with $k \leq n_2$. We observe that $F_{1,2}$ is obtained by greedily matching u_i with w_i. Then, let F be any maximum-cardinality matching of G that can be obtained from $F_{1,2}$ using augmenting paths. By construction, $u_1, u_2, \ldots, u_k$ are matched by F. In particular, since every u_i is isolated in $H_1 = G[M_1]$, it is matched by F to some vertex in M_2. So, let $A_2 \subseteq M_2$ be the vertices matched by F with a vertex in $V \setminus M_2$ (possibly, in M_1). Since M_2 is a module, we can always assume that A_2 induces a prefix $(w_1, w_2, \ldots, w_j)$ of the canonical ordering (*i.e.*, see [12, Lemma 5.2]). Finally, let $B_2 \subseteq V \setminus M_2$, $|B_2| = |A_2|$, be the set of vertices matched by F with a vertex of A_2. Note that we have $u_1, u_2, \ldots, u_k \in B_2$. Since M_2 is a module, there are all possible edges between A_2 and B_2. As a result, we can always replace the matched edges between A_2, B_2 by any perfect matching between these two sets without changing the cardinality of F. It implies that we can assume w.l.o.g. every u_i is matched to w_i. □

By Lemma 11, we can remove $V(F_{1,2})$, the set of vertices incident to an edge of $F_{1,2}$, from G. We stress that during this phase, all the operations except maybe the last one increase the cardinality of the matching. Furthermore, the only possible operation that does not increase the cardinality of the matching is the replacement of an edge in F_2 by an edge in $F_{1,2}$. Doing so, there are three cases. If $M_2 \subseteq V(F_{1,2})$ then $M_1 \setminus V(F_{1,2})$ is isolated. We apply Reduction rule 1. If $M_1 \subseteq V(F_{1,2})$ then M_1 is already eliminated. We call these two first cases pathological. The interesting case is when both $M_1 \setminus V(F_{1,2})$ and $M_2 \setminus V(F_{1,2})$ are nonempty. In particular, we obtain through the replacement operation the following stronger property:

Property 3. All vertices in M_1 are matched by F_1.

We will assume Property 3 to be true after First Phase.

Second phase: virtual split edges. We complete the previous phase by performing a Split between M_2, M_1 (Operation 2). That is, while there exist two exposed vertices $x_2, y_2 \in M_2$ and a matched edge $\{x_1, y_1\} \in F_1$ we replace $\{x_1, y_1\}$ by $\{x_1, x_2\}, \{y_1, y_2\}$ in the current matching. However, we encode the Split operation using virtual edges in H_2. Formally, we add a virtual edge $\{x_2, y_2\}$ in H_2 that is labeled by the corresponding edge $\{x_1, y_1\} \in F_1$. Let H_2^* and F_2^* be obtained from H_2 and F_2 by adding all the virtual edges. We set $G^* = G' \setminus v_1$, $\mathcal{P}^* = \{H_2^*\} \cup (\mathcal{P} \setminus \{H_1, H_2\})$ and $\mathcal{F}^* = \{F_2^*\} \cup (\mathcal{F} \setminus \{F_1, F_2\})$.

Intuitively, virtual edges are used in order to shorten the augmenting paths crossing M_1.

Third phase: post-processing. Let F^* be a maximum-cardinality matching of the substitution $G^*(\mathcal{P}^*)$ (*i.e.*, obtained by applying our reduction rules to the new instance). We construct a matching F for G as follows. We add in F all the non virtual edges in F^*. For every virtual edge $\{x_2, y_2\}$, let $\{x_1, y_1\} \in F_1$ be its label. If $\{x_2, y_2\} \in F^*$ then we add $\{x_1, y_2\}, \{x_2, y_1\}$ in F, otherwise we add $\{x_1, y_1\}$ in F. In the first case, we say that we confirm the Split operation, whereas in the second case we say that we cancel it. Finally, we complete F with all the edges of F_1 that do not label any virtual edge (*i.e.*, unused during the second phase).

Lemma 12. *F is a maximum-cardinality matching of G.*

The above result is proved by contrapositive. More precisely, we prove intricate properties on the intersection of *shortest* augmenting paths with pendant modules. Using these properties and the virtual edges, we could transform any shortest F-augmenting path into an F^*-augmenting path, that is a contradiction.

Proof. Suppose for the sake of contradiction that F is not maximum. Let $P = (u_1, u_2, \ldots, u_{2\ell})$ be a shortest F-augmenting path. In order to derive a contradiction, we will transform P into an F^*-augmenting path in $G^*(\mathcal{P}^*)$. For that, we essentially need to avoid passing by M_1, using instead the virtual edges. In the first part of the proof, we show that P intersects M_1 in at most one edge (Claim 11). We need a few preparatory claims in order to prove this result.

First we observe that no end of P can be in M_1:

Claim 8. *$M_1 \subseteq V(F)$. In particular, $u_1, u_{2\ell} \notin M_1$.*

Proof. According to Property 3, all vertices in M_1 are matched by F_1. Our procedure during the third phase ensures that $V(F_1) \subseteq V(F)$, and so, $M_1 \subseteq V(F)$. ◇

Then, we prove that for every $\{x_1, y_1\} \in F_1$ we have either $x_1, y_1 \notin V(P)$ or $\{x_1, y_1\} \in E(P)$. This result follows from the combination of Claims 9 and 10.

Claim 9. *Let $\{x_1, y_1\} \in F_1$. Either $x_1, y_1 \in V(P)$ or $x_1, y_1 \notin V(P)$.*

Proof. Suppose for the sake of contradiction $x_1 \in V(P)$ but $y_1 \notin V(P)$. Up to reverting the path P we have $x_1 = u_{2i+1}$ for some i. Then, since we have $y_1 \notin V(P)$ and M_1 induces a matching, $u_{2i+2} \notin M_1$. It implies $u_{2i+2} \in M_2$. Furthermore, our construction ensures that u_{2i} (the vertex matched with x_1) was left exposed by F_2. Indeed, u_{2i} must be an end of a virtual edge (cf. Second phase). Since $E(H_2) = F_2$ it implies $u_{2i-1} \notin M_2$. Finally, since $u_{2i-1} \in N_G(M_2)$ and M_2 is a module, $P' = (u_1, u_2, \ldots, u_{2i-1}, u_{2i+2}, \ldots, u_{2\ell})$ is a shorter augmenting path than P, thereby contradicting the minimality of P. Therefore, as claimed, either $x_1, y_1 \in V(P)$ or $x_1, y_1 \notin V(P)$. $\diamond$

Claim 10. *Let $u_i, u_j \in V(P) \cap M_1$, $j > i$, such that $\{u_i, u_j\} \in F_1$. Then, $j = i + 1$.*

Proof. The result trivially holds if $\{u_i, u_j\} \in F$. Thus, we assume from now on $\{u_i, u_j\} \notin F$. We need to consider the following cases:

- Case i odd, j even. Since $P' = (u_1, u_2, \ldots, u_{i-1}, u_i, u_j, u_{j+1}, \ldots, u_{2\ell})$ is also an augmenting path, we get $j = i + 1$ by minimality of P.
- Case i odd, j odd. Note that $u_{j+1} \notin M_1$ since we assume $\{u_i, u_j\} \in F_1$ and M_1 induces a matching. Then, since $u_{j+1} \in N_G(M_1)$ and M_1 is a module we have that $P' = (u_1, u_2, \ldots, u_{i-1}, u_i, u_{j+1}, \ldots, u_{2\ell})$ is a shorter augmenting path than P, thereby contradicting the minimality of P.
- Case i even, j even (obtained from the previous case by reversing the augmenting path). Note that $u_{i-1} \notin M_1$ since we assume $\{u_i, u_j\} \in F_1$ and M_1 induces a matching. Then, since $u_{i-1} \in N_G(M_1)$ and M_1 is a module we have that $P' = (u_1, u_2, \ldots, u_{i-1}, u_j, u_{j+1}, \ldots, u_{2\ell})$ is a shorter augmenting path than P, thereby contradicting the minimality of P.
- Case i even, j odd. As before, we have $u_{i-1}, u_{j+1} \notin M_1$, that implies $u_{i-1}, u_{j+1} \in M_2$. We observe that $\{u_{i+1}, u_{j-1}\}$ is a virtual edge labeled by $\{u_i, u_j\}$. In particular, u_{i+1}, u_{j-1} are isolated in M_2, and so, $u_{i+2}, u_{j-2} \notin M_2$. It implies, since $u_{i+2}, u_{j-2} \in N_G(M_2)$ and M_2 is a module, $P' = (u_1, u_2, \ldots, u_{i-1}, u_{i+2}, \ldots, u_{j-2}, u_{j+1}, \ldots, u_{2\ell})$ is shorter augmenting path than P, thereby contradicting the minimality of P.

Overall the first case implies, as claimed, $j = i + 1$, whereas all other cases lead to a contradiction. Therefore, $j = i + 1$. $\diamond$

Finally, our last preparatory claim is that P can cross the module M_1 in at most one edge.

Claim 11. $|E(P) \cap F_1| \leq 1$.

Proof. Suppose by contradiction there exist $\{u_i, u_{i+1}\}, \{u_j, u_{j+1}\} \in F_1 \cap E(P)$, for some $i < j$. Since M_1 induces a matching, $u_{i-1}, u_{j-1} \notin M_1$. There are three cases.

- Case i, j even. Then, $P' = (u_1, \ldots, u_{i-1}, u_j, u_{j+1}, \ldots, u_{2\ell})$ is a shorter augmenting path than P, thereby contradicting the minimality of P.
- Case i, j odd. Then, $P' = (u_1, \ldots, u_i, u_{j-1}, u_j, \ldots, u_{2\ell})$ is a shorter augmenting path than P, thereby contradicting the minimality of P.

- Case i even, j odd (Case i odd, j even is symmetrical to this one). Then, $P' = (u_1, \ldots, u_{i-1}, u_{j+1}, \ldots, u_{2\ell})$ is a shorter augmenting path than P, thereby contradicting the minimality of P.

As a result, $|E(P) \cap F_1| \leq 1$. We note that in order to prove this result, we did not use the fact that M_1 is pendant. $\diamond$

Let $\{u_{i_0}, u_{i_0+1}\}$ be the unique edge in $E(P) \cap F_1$. Such edge must exist since otherwise, P would also be an F^*-augmenting path. In order to derive a contradiction, we are left to replace $\{u_{i_0}, u_{i_0+1}\}$ with a virtual edge. We prove next that it can be easily done if i_0 is odd, *i.e.*, $\{u_{i_0}, u_{i_0+1}\} \notin F$. Indeed, in such case we observe that $\{u_{i_0-1}, u_{i_0+2}\}$ is the virtual edge that is labeled by $\{u_{i_0}, u_{i_0+1}\}$. Furthermore, $\{u_{i_0-1}, u_{i_0+2}\} \in F^*$ since we confirmed the Split. Therefore, we will assume from now on that i_0 is even, *i.e.*, $\{u_{i_0}, u_{i_0+1}\} \in F$.

We will need the following observation:

Claim 12. *The vertices u_{i_0-1}, u_{i_0+2} are the only vertices in $M_2 \cap V(P)$.*

Proof. Suppose for the sake of contradiction this is not the case. By symmetry, we can assume the existence of an index $j < i_0 - 1$ such that $u_j \in M_2$. Furthermore, j is even since otherwise, $P' = (u_1, \ldots, u_j, u_{i_0}, u_{i_0+1}, \ldots, u_{2\ell})$ would be a shorter augmenting path than P, thereby contradicting the minimality of P. For the same reason as above, we also have $u_{j-1} \notin M_2$. However, since $u_{j-1} \in N_G(M_2)$ and M_2 is a module, it implies that $P' = (u_1, \ldots, u_{j-1}, u_{i_0+2}, \ldots, u_{2\ell})$ would be a shorter augmenting path than P, thereby contradicting the minimality of P. $\diamond$

There are three cases.

1. Case $u_{i_0-1}, u_{i_0+2} \notin V(F_2^*)$ (left exposed by F_2^*). There exists a virtual edge $\{x_2, y_2\}$ that is labeled by $\{u_{i_0}, u_{i_0+1}\}$ (otherwise, the second phase could have continued with u_{i_0-1}, u_{i_0+2} and $\{u_{i_0}, u_{i_0+1}\}$). The two of x_2, y_2 cannot be matched together in F^* since we have $\{u_{i_0}, u_{i_0+1}\} \in F$. Nevertheless, since x_2, y_2 are adjacent in the substitution $G^*(\mathcal{P}^*)$, at least one of the two vertices, say x_2, is matched by F^*. There are two subcases.
   (a) Subcase y_2 is exposed. Let $\{w, x_2\} \in F^*$. Since $w \neq y_2$ we have $w \notin M_2$. Then, $P^* = (u_1, u_2, \ldots, u_{i_0-1}, w, x_2, y_2)$ is an F^*-augmenting path, thereby contradicting the maximality of F^*.
   (b) Subcase y_2 is matched. Let $\{w, x_2\}, \{w', y_2\} \in F^*$. As before, $w, w' \notin M_2$. Then, $P^* = (u_1, u_2, \ldots, u_{i_0-1}, w, x_2, y_2, w', u_{i_0+2}, \ldots, u_{2\ell})$ is an F^*-augmenting path, thereby contradicting the maximality of F^*.
2. Case $\{u_{i_0-1}, u_{i_0+2}\} \in F_2^*$. We have $\{u_{i_0-1}, u_{i_0+2}\} \notin F^*$ (otherwise, since u_{i_0-1}, u_{i_0+2} are non-adjacent in P, this edge $\{u_{i_0-1}, u_{i_0+2}\}$ should be virtual; but then, its label should also be contained in $V(P)$ and, since it cannot be u_{i_0}, u_{i_0+1}, the latter would contradict Claims 9, 10, 11). Hence, we have that $P^* = (u_1, u_2, \ldots, u_{i_0-1}, u_{i_0+2}, \ldots, u_{2\ell})$ is an F^*-augmenting path, thereby contradicting the maximality of F^*.
3. Case $\{u_{i_0-1}, w\} \in F_2^*$ for some $w \neq u_{i_0+2}$ (Case $\{u_{i_0+2}, w\} \in F_2^*$ for some $w \neq u_{i_0-1}$ is symmetrical to this one). By Claim 12, $w \notin V(P)$. There are two subcases.
   (a) Subcase w is exposed. Then, $P^* = (u_1, u_2, \ldots, u_{i_0-1}, w)$ is an F^*-augmenting path, thereby contradicting the maximality of F^*.
   (b) Subcase w is matched. Let $\{w, w'\} \in F$. As before, $w' \notin M_2$. Furthermore, $w' \notin M_1$ (otherwise, the edge $\{u_{i_0-1}, w\}$ should be virtual; but then, $u_{i_0-2} \in M_1$, thus contradicting Claims 9, 10, 11). Then, $P^* = (u_1, u_2, \ldots, u_{i_0-1}, w, w', u_{i_0+2}, \ldots, u_{2\ell})$ is an F^*-augmenting path, thereby contradicting the maximality of F^*.

Summarizing, by contrapositive we get F^* maximum for $G^*(\mathcal{P}^*) \Longrightarrow F$ maximum for G. □

4.3.1. Complexity analysis

We address the time complexity of our reduction rule phase by phase.

- **First Phase.** The desired matching $F_{1,2}$ can be computed in $\mathcal{O}(|F_{1,2}|)$ time (and so, in $\mathcal{O}(n_1)$ time), using the canonical orderings at M_1 and M_2. On our way, we can update the canonical ordering of M_1 by choosing as a new head for the list the cell where we stopped after having computed $F_{1,2}$ (if we do not fall in a pathological case, then this is the first vertex that is matched by F_1). We proceed similarly in order to update the canonical ordering of M_2. – Note that in doing so, we can easily retrieve the original canonical orderings of M_1, M_2 after Third phase (post-processing). –

- **Second Phase.** Recall that we assume that all vertices of M_1 are matched by F_1 (Property 3). Therefore, using the canonical ordering of M_1, we can compute in $\mathcal{O}(1)$ time an edge $\{x_1, y_1\}$ of F_1. In order to create virtual edges, we scan the canonical ordering of M_2 until we reach the last vertex left exposed by F_2. It can be done in $\mathcal{O}(n_2)$ time, and so, in $\mathcal{O}(\Delta m(G', G^*))$ time. Let $x_2, y_2 \in M_2$ be the two last exposed vertices in this canonical ordering, if any. We add a virtual edge $\{x_2, y_2\}$ simply by modifying the auxiliary integer variables of these two vertices in our implementation of the canonical ordering (*i.e.*, see Sec. 2.2.1); on our way, we also modify our internal representation of the matching of G (*i.e.*, as a global linear-size array) in order to insert in it the virtual edge $\{x_2, y_2\}$. Note that if we know how many virtual edges there are, then finding their respective labels is easy: indeed, the label of the last virtual edge – in the canonical ordering of M_2 – is the first edge of F_1 – in the canonical ordering of M_1 –, and so forth. Therefore, we only need to keep track of the number of virtual edges, say in an auxiliary integer variable.

- **Third phase.** Our above construction ensures that the number k'_1 of virtual edges is known, and that the virtual edges are exactly the k'_1 first matched edges in the (modified) canonical ordering of M_2. We can output this set of virtual edges by scanning the canonical ordering of M_2, in $\mathcal{O}(n_2)$ time, and so, in $\mathcal{O}(\Delta m(G', G^*))$ time. On our way, we modify the auxiliary integer variables in our implementation of this canonical ordering (*i.e.*, see Sec. 2.2.1) in order to remove the virtual edges from the matching F_2^*. Recall that our construction of virtual edges also allows us to retrieve their labels, in $\mathcal{O}(1)$ time per virtual edge, using the canonical ordering of M_1. Now, we just need to scan these virtual edges and their labels, using Property 1, in order to either confirm or cancel each split.

Therefore, the total running time is in $\mathcal{O}(\Delta m(G', G^*))$.

4.4. Main result

Our framework consists in applying any reduction rule presented in this section until it can no more be done. Then, we rely on the following result:

Theorem 2 (Theorem 5.7 in [12])**.** Module Matching *can be solved, for any* $\langle G', \mathcal{P}, \mathcal{F} \rangle$ *in* $\mathcal{O}(\Delta\mu \cdot p^4)$*-time.*

We are now ready to state our main result in this paper (the proof of which directly follows from all the previous results in this section).

Theorem 3. *Let $G = (V, E)$ be a graph. Suppose that, for every prime subgraph H' in the modular decomposition of G, its pruned subgraph has order at most k. Then,* Maximum Matching *can be solved for G in $\mathcal{O}(k^4 \cdot n + m \log n)$-time.*

Proof. By Lemma 5, it suffices to solve Module Matching for any $\langle H', \mathcal{P}, \mathcal{F} \rangle$, with H' in the modular decomposition of G, in time $\mathcal{O}(p + \Delta m \cdot \log p + k^4 \cdot \Delta\mu)$. For that, we start computing the pruned subgraph H^{pr} of H', and a corresponding pruning sequence. By Proposition 1, it can be done in $\mathcal{O}(p + \Delta m \cdot \log p)$-time. We assume in what follows that H^{pr} has at least two vertices (if it is not the case, then we abort the pruning sequence as soon as there remain exactly two vertices). This is a technical assumption needed in order to apply some of our intermediate operations on the modular decomposition tree without having to consider explicitly some degenerate situations (*i.e.*, see Sec. 2.1.1). Then, we follow the pruning sequence and at each step, we apply the reduction rule that corresponds to the current one-vertex extension. Specifically, let $(v_1, v_2, \ldots, v_q)$ be the pruning sequence. Let $H^0 := H'$.

1. For every $1 \leq j \leq q$, we transform the instance $\langle H^{j-1}, \mathcal{P}^{j-1}, \mathcal{F}^{j-1} \rangle$ that we are currently considering for a new instance $\langle H^j, \mathcal{P}^j, \mathcal{F}^j \rangle$, where $H^j := H^{j-1} \setminus v_j$. On our way, we apply some pre-processing rule, that is specified by the Reduction rule corresponding to v_j.
2. We solve Module Matching on the reduced instance $\langle H^{pr}, \mathcal{P}^{pr}, \mathcal{F}^{pr} \rangle$.
3. Finally, for every $1 \leq j \leq q$, we compute a solution for $\langle H^{j-1}, \mathcal{P}^{j-1}, \mathcal{F}^{j-1} \rangle$ from a solution for $\langle H^j, \mathcal{P}^j, \mathcal{F}^j \rangle$. For that, we apply a post-processing rule, that is specified by the Reduction rule corresponding to v_j.

In Sec. 3.1, we explained how, after ordering the adjacency list of H', we can compute H^j from H^{j-1} and vice versa, for any $1 \leq j \leq q$. It takes total time in $\mathcal{O}(\Delta m)$. The application of all our reduction rules (both the pre-processing rules and the post-processing rules) also takes total $\mathcal{O}(\Delta m)$ time; see Sec. 4.1, 4.2 and 4.3 for a detailed analysis. Finally, we stress that if H' is degenerate (complete or edgeless) then H^{pr} is trivial, otherwise by the hypothesis H^{pr} has order at most k. As a result, by Theorem 2 we can solve Module Matching on the reduced instance in $\mathcal{O}(\Delta\mu \cdot k^4)$-time. □

5. Applications

We conclude this paper presenting applications and refinements of our main result to some graph classes. Recall that cographs are exactly the graphs that are totally decomposable by modular decomposition [11]. We start showing that several distinct generalizations of cographs in the literature are totally decomposable by the pruned modular decomposition.

Distance-hereditary graphs. A graph $G = (V, E)$ is distance-hereditary if it can be reduced to a singleton by pruning sequentially the pendant vertices and twin vertices [2]. Conversely, G is co-distance hereditary if it is the complement of a distance-hereditary graph, *i.e.*, it can be reduced to a singleton by pruning sequentially the anti-pendant vertices and twin vertices. In both cases, the corresponding pruning sequence can be computed in linear-time [17, 20]. Furthermore, since the algorithm from [17] fails in linear time if the input is *not* distance-hereditary, we may also decide in linear time in which case we are (*i.e.*, either distance-hereditary or co-distance hereditary). Therefore, we can derive the following result from our framework:

Proposition 2. Maximum Matching *can be solved in linear-time on graphs that can be modularly decomposed into distance-hereditary graphs and co-distance hereditary graphs.*

We stress that even for distance-hereditary graphs, we may need to use the reduction rule of Section 4.3 for pendant modules. Indeed, as we follow the pruning sequence, we may encounter twin vertices and merge them into a single module. It means that, even in the simpler case of distance-hereditary graphs, we need to handle with modules instead of just handling with vertices. In the same way, even for co-distance hereditary graphs, we may need to use the reduction rule of Section 4.2 for anti-pendant modules.

Trees are a special subclass of distance-hereditary graphs. We say that a graph has *modular treewidth* at most k if every prime quotient subgraph in its modular decomposition has treewidth at most k. In particular, graphs with modular treewidth at most one are exactly the graphs that can be modularly decomposed into trees[5]. We stress the following consequence of Proposition 2:

Corollary 2. MAXIMUM MATCHING *can be solved in linear-time on graphs with modular-treewidth at most one.*

The case of graphs with modular treewidth $k \geq 2$ is left as an intriguing open question.

Tree-perfect graphs. Two graphs G_1, G_2 are P_4-isomorphic if there exists a bijection from G_1 to G_2 such that a 4-tuple induces a P_4 in G_1 if and only if its image in G_2 also induces a P_4 [9]. The notion of P_4-isomorphism plays an important role in the study of perfect graphs (*e.g.*, see [9, 46]). A graph is *tree-perfect* if it is P_4-isomorphic to a tree [6]. We prove the following result:

Proposition 3. *Tree-perfect graphs are totally decomposable by the pruned modular decomposition. In particular,* MAXIMUM MATCHING *can be solved in linear-time on tree-perfect graphs.*

Our proof is based on a deep structural characterization of tree-perfect graphs [6]. First we need to introduce a few additional graph classes. Given a vertex-ordering $(v_1, v_2, \ldots, v_n)$ let $N_{<i}(v_i) =$

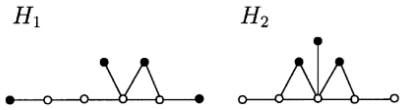


Fig. 3. The graphs H_1 and H_2.

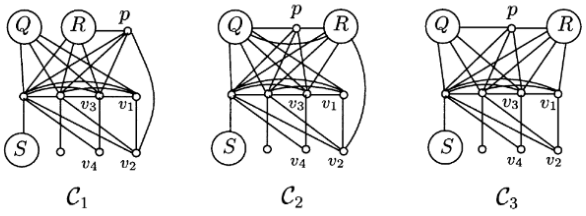


Figure 6: Examples of tree-perfect graphs [6]. The sets Q, R, S represent modules substituting the vertices q, r, v_n.

$N(v_i) \cap \{v_1, v_2, \ldots, v_{i-1}\}$. A graph is termed *elementary* if it admits a vertex-ordering $(v_1, v_2, \ldots, v_n)$ such that, for every i:

$$N_{<i}(v_i) = \begin{cases} \{v_1, v_2, \ldots, v_{i-2}\} \text{ if } i \text{ is odd} \\ \{v_{i-1}\} \text{ otherwise.} \end{cases}$$

[5]Our definition is more restricted than the one in [43] since they only impose the quotient subgraph G' to have bounded treewidth.

Note that such ordering as above is a pruning sequence by pendant and anti-pendant vertices. Let us define the classes $\mathcal{C}_j,\ j = 1, 2, 3$ as all the graphs that can be obtained from an elementary graph, with ordering $(v_1, v_2, \ldots, v_n)$, by adding the three new vertices p, q, r and the following set of edges:

- (for all classes) $\{p, v_i\}, \{q, v_i\}, \{r, v_i\}$ for every $i > 1$ odd;
- (only for $\mathcal{C}_1$) $\{v_1, q\},\ \{p, r\}$ and $\{v_2, p\}$;
- (only for $\mathcal{C}_2$) $\{p, q\},\ \{p, r\}, \{q, r\},\ \{v_1, q\}$ and $\{v_2, r\}$;
- (only for $\mathcal{C}_3$) $\{p, q\},\ \{p, r\}$ and $\{v_1, r\}$.

The graphs H_1, H_2 are illustrated in Fig. 6

Tree-perfect graphs are fully characterized in [6], and a linear-time recognition algorithm can be derived from this characterization. We will only use a weaker form of this result:

Theorem 4 ([6])**.** *A graph $G = (V, E)$ is a tree-perfect graph* only if *every nontrivial module induces a cograph and the quotient graph G' is in one of the following classes or their complements: trees; elementary graphs; $\mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3$; H_1 or H_2.*

Proof of Proposition 3. Let $G = (V, E)$ be a tree-perfect graph. By Theorem 4 every nontrivial module induces a cograph. It implies that all the subgraphs in the modular decomposition of G, except maybe its quotient graph G', are cographs, and so, totally decomposable by the modular decomposition. We are left with proving that G' is totally decomposable by the pruned modular decomposition. The latter is immediate whenever G is a tree, H_1, H_2 or a complement of one of these graphs. Furthermore, we already observed that elementary graphs can be reduced to a singleton by pruning pendant and anti-pendant vertices sequentially. Therefore elementary graphs and their complements are also totally decomposable by the pruned modular decomposition.

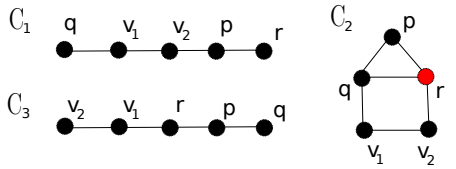


Figure 7: Small tree-perfect graphs with 5 vertices.

Finally, we prove that graphs in $\mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3$ are totally decomposable (this will prove the same for their complements). Recall that every graph $G' \in \mathcal{C}_j,\ j = 1, 2, 3$ can be obtained from an elementary graph H with ordering $(v_1, v_2, \ldots, v_n)$ by adding three new vertices p, q, r and a set of specified edges. Furthermore, for every odd i, resp. for every even i, we have that v_i is anti-pendant, resp. pendant, in $H \setminus \{v_{i+1}, \ldots, v_n\}$. Since p, q, r are made adjacent to every v_i for $i > 1$ odd, and nonadjacent to every v_i for $i > 2$ even, this above property stays true in $G' \setminus \{v_{i+1}, \ldots, v_n\}$. As a result, we can remove the vertices $v_n, v_{n-1}, \ldots, v_3$ sequentially. We are left with studying the subgraph induced by p, q, r, v_1, v_2. The latter subgraph is a path if $G' \in \mathcal{C}_1 \cup \mathcal{C}_3$, otherwise it is a house (cf. Fig. 7). In both cases, such subgraph can be totally decomposed by pruning pendant and anti-pendant vertices sequentially. □

The case of unicycles. We end up this section with a refinement of our framework for the special case of unicyclic quotient graphs (*a.k.a.*, graphs with exactly one cycle). We stress that unicycles are a special case of graphs of treewidth two.

Proposition 4. MAXIMUM MATCHING *can be solved in linear-time on the graphs that can be modularly decomposed into unicycles.*

For that, we reduce the case of unicycles to the case of cycles (iteratively removing pendant modules). Then, we test for all possible numbers of matched edges between two adjacent modules. Doing so, we reduce the case of cycles to the case of paths.

Proof. By Lemma 5, it suffices to show that on every instance $\langle G', \mathcal{P}, \mathcal{F} \rangle$ such that G' is a unicycle, we can solve MODULE MATCHING in $\mathcal{O}(\Delta m)$-time. Recall that G' is a unicycle if it can be reduced to a cycle by pruning the pendant vertices sequentially. Therefore, in order to prove the result, we only need to prove it when G' is a cycle.
Given an edge $e = \{v_i, v_j\} \in E(G')$, our strategy consists in fixing the number $\mu_{i,j}$ of matched edges with one end in M_i and the other end in M_j. By [12, Lemma 5.2], we can always assume that the ends of these $\mu_{i,j}$ edges are the $\mu_{i,j}$ first vertices in a canonical ordering of M_i (w.r.t. F_i), and in the same way, the $\mu_{i,j}$ first vertices in a canonical ordering of M_j (w.r.t. F_j). We can remove these above vertices from M_i, M_j and update the matchings F_i, F_j accordingly. Doing so, we can remove the edge $\{v_i, v_j\}$ from G'. Then, since $G' \setminus e$ is a path, we can systematically apply the reduction rule for pendant modules (Section 4.3). Overall, we test for all possible number of matched edges between M_i and M_j and we keep any one possibility that gives the largest matching.
In order to apply our strategy, we choose any edge e such that $|M_i|$ is minimized. Doing so, there can only be at most $\mathcal{O}(n_i) = \mathcal{O}(\Delta m/p)$ possibilities for $\mu_{i,j}$, where $p = |V(G')|$. However, we are not done yet as we now need to test for every possibility in $\mathcal{O}(p)$-time. A naive implementation of this test, using the reduction rule of Section 4.3, would run in $\mathcal{O}(\Delta m)$-time. We propose a faster implementation that only computes the *cardinality* of the solution (*i.e.*, not the matching itself). The latter is enough in order to compute the optimum value for $\mu_{i,j}$. Then, once this value is fixed, we can run the naive implementation in order to compute a maximum-cardinality matching.

W.l.o.g., a smallest-cardinality module is M_1, *i.e.*, we have $i = 1$, $j = p$ and $e = \{v_1, v_p\}$. For every t let $n_t = |M_t|$. Furthermore, let $\mu_t = |F_t|$. Note that there are exactly $n_t - 2\mu_t$ vertices in M_t that are left exposed by F_t. We also maintain a counter μ representing the cardinality of the current matching. Initially $\mu = \mu_{1,p}$. Then, we proceed as follows:

- We decrease n_1 by $\mu_{1,p}$. If $\mu_{1,p} > n_1 - 2\mu_1$, then we also decrease μ_1 by $\lceil (\mu_{1,p} - n_1 + 2\mu_1)/2 \rceil$. We proceed similarly for M_p. After that, we can remove e from G'. We have that $G' \setminus e$ is isomorphic to the path $(v_1, v_2, \ldots, v_p)$. This first step takes constant-time.

  Note that we decrease n_1 by $\mu_{1,p}$ in order to simulate the removal of the $\mu_{1,p}$ first vertices in a canonical ordering of M_1 w.r.t. F_1. If $\mu_{1,p} \leq n_1 - 2\mu_1$ then we only removed exposed vertices and there is nothing else to change. Otherwise, we also need to update μ_1.

- Then, for every $1 \leq t < p$, we simulate the reduction rule of Section 4.3 sequentially. More precisely:

  1. Let $k_t = \min\{n_t - 2\mu_t, n_{t+1}\}$. We decrease n_t, n_{t+1} by k_t, while we *increase* μ by k_t. Furthermore, if $k_t > n_{t+1} - 2\mu_{t+1}$, then we decrease μ_{t+1} by $\lceil (k_t - n_{t+1} + 2\mu_{t+1})/2 \rceil$. Three cases might occur:
     - If $k_t = n_t$, then we continue directly to the next vertex v_{t+1}.
     - If $k_t = n_{t+1}$, then we increase μ by μ_t. We continue directly to the next vertex v_{t+1}.
     - Otherwise, we go to Step 2.

Let us justify a posteriori these above operations that we perform on integer variables (in total $\mathcal{O}(1)$ time). For that, we observe that k_t is the maximum number of exposed vertices in M_t that can be matched with a vertex of M_{t+1} in the first phase of the reduction rule, *i.e.*, the cardinality of the matching $F_{t,t+1}$. Since during this phase, we remove $V(F_{t,t+1})$ from $M_t \cup M_{t+1}$, we need to update n_t, n_{t+1} accordingly. In the same way, since we proved that there always exists a maximum matching containing $F_{t,t+1}$ (Lemma 11), we also need to update the size μ of the current matching. Then, as before, if $k_t \leq n_{t+1} - 2\mu_{t+1}$ then we only remove exposed vertices from M_{t+1} and so, there is nothing else to be done. Otherwise, we also need to update μ_{t+1}. We fall in a degenerate case if $k_t = n_t$ or $k_t = n_{t+1}$. In the former case, we do not modify the value of μ, however in the latter case (M_t is now an isolated module) we can increase this value by μ_t. For both degenerate cases, we ignore the second and third phases of the rule, hence we continue directly to the next vertex v_{t+1}.

2. Let $k'_t = \min\{\lfloor (n_{t+1} - 2\mu_{t+1})/2 \rfloor, \mu_t\}$. We increase μ_{t+1} by exactly k'_t.
   Indeed, k'_t is the number of virtual edges that we create during the second phase.
3. Finally, in order to simulate the third phase, we claim that we only need to increase μ by exactly μ_t.
   Indeed, after a solution F_t^* was obtained for $(v_{t+1}, \ldots, v_p)$ the reduction rule proceeds as follows. Either we confirm a Split operation, *i.e.*, we replace a virtual matched edge in F_t^* by two edges between M_t, M_{t+1}; or we cancel the Split operation, *i.e.*, we add an edge of F_t in the current matching. In both cases, the cardinality of the solution increases by one. Then, all the edges of F_t that were not used during the second phase are added to the current matching. Overall, we have as claimed that the cardinality of the solution increases by exactly μ_t.

The procedure ends for $t = p$. In this situation, the quotient subgraph is reduced to a single node, and so, we only need to increase the current size μ of the matching by μ_p. Summarizing, since all the steps of this procedure take constant-time, the total running-time is in $\mathcal{O}(p)$. □

6. Open problems

The pruned modular decomposition happens to be an interesting add up in the study of Maximum Matching algorithms. An exhaustive study of its other algorithmic applications remains to be done. Moreover, another interesting question is to characterize the graphs that are totally decomposable by this new decomposition. We note that our pruning process can be seen as a repeated update of the modular decomposition of a graph after some specified modules (pendant, anti-pendant) are removed. However, we can only detect a restricted family of these new modules (*i.e.*, universal, isolated, twins). A fully dynamic modular decomposition algorithm could be helpful in order to further refine our framework.

Acknowledgements

We wish to thank the anonymous referees for their careful reading of the first version of this manuscript, and their useful comments.

References

[1] F. Abu-Khzam, S. Li, C. Markarian, F. Meyer auf der Heide, and P. Podlipyan. Modular-width: An auxiliary parameter for parameterized parallel complexity. In *International Workshop on Frontiers in Algorithmics*, pages 139–150. Springer, 2017.

[2] H.-J. Bandelt and H. Mulder. Distance-hereditary graphs. *J. of Combinatorial Theory, Series B*, 41(2):182–208, 1986.

[3] C. Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences*, 43(9):842–844, 1957.

[4] J. A. Bondy and U. S. R. Murty. *Graph theory*. Grad. Texts in Math., 2008.

[5] A. Brandstädt, T. Klembt, and S. Mahfud. P_6-and triangle-free graphs revisited: structure and bounded clique-width. *Discrete Mathematics and Theoretical Computer Science*, 8, 2006.

[6] A. Brandstädt and V. Le. Tree-and forest-perfect graphs. *Discrete applied mathematics*, 95(1-3):141–162, 1999.

[7] H. Bunke. Graph matching: Theoretical foundations, algorithms, and applications. In *Proc. Vision Interface*, volume 2000, pages 82–88, 2000.

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

[9] V. Chvátal. A semi-strong perfect graph conjecture. In *North-Holland mathematics studies*, volume 88, pages 279–280. Elsevier, 1984.

[10] O. Cogis and E. Thierry. Computing maximum stable sets for distance-hereditary graphs. *Discrete Optimization*, 2(2):185 – 188, 2005.

[11] D. Corneil, Y. Perl, and L. Stewart. A linear recognition algorithm for cographs. *SIAM J. Comput.*, 14(4):926–934, 1985.

[12] 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.

[13] A. Cournier and M. Habib. A new linear algorithm for modular decomposition. In *Colloquium on Trees in Algebra and Programming*, pages 68–84. Springer, 1994.

[14] E. Dahlhaus. Minimum fill-in and treewidth for graphs modularly decomposable into chordal graphs. In *WG*, pages 351–358. Springer, 1998.

[15] E. Dahlhaus, J. Gustedt, and R. McConnell. Efficient and practical algorithms for sequential modular decomposition. *Journal of Algorithms*, 41(2):360–387, 2001.

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

[17] G. Damiand, M. Habib, and C. Paul. A simple paradigm for graph recognition: application to cographs and distance hereditary graphs. *Theoretical Computer Science*, 263(1-2):99–111, 2001.

[18] E. Dekel and S. Sahni. A parallel matching algorithm for convex bipartite graphs and applications to scheduling. *Journal of Parallel and Distributed Computing*, 1(2):185–205, 1984.

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

[20] S. Dubois, V. Giakoumakis, and C. El Mounir. On co-distance hereditary graphs. In *CTW*, pages 94–97, 2008.

[21] G. Ducoffe and A. Popa. The Use of a Pruned Modular Decomposition for Maximum Matching Algorithms on Some Graph Classes. In *29th International Symposium on Algorithms and Computation (ISAAC 2018)*, volume 123 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

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

[23] F. Fomin, M. Liedloff, P. Montealegre, and I. Todinca. Algorithms parameterized by vertex cover and modular-width, through potential maximal cliques. *Algorithmica*, 80(4):1146–1169, 2018.

[24] 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.

[25] 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.

[26] H. Gabow and R. Tarjan. A linear-time algorithm for a special case of disjoint set union. In *STOC'83*, pages 246–251. ACM, 1983.

[27] J. Gajarskỳ, M. Lampis, and S. Ordyniak. Parameterized Algorithms for Modular-Width. In *IPEC'13*, volume 8246 of *LNCS*, pages 163–176. Springer, 2013.

[28] T. Gallai. Transitiv orientierbare graphen. *Acta Math. Hungarica*, 18(1):25–66, 1967.

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

[30] M. Habib, F. De Montgolfier, and C. Paul. A simple linear-time modular decomposition algorithm for graphs, using order extension. In *SWAT*, pages 187–198. Springer, 2004.

[31] M. Habib, R. McConnell, C. Paul, and L. Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234(1-2):59–84, 2000.

[32] M. Habib and C. Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010.

[33] D. Harel and R. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[34] J. Hopcroft and R. Karp. An n^5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.

[35] R. Karp and M. Sipser. Maximum matching in sparse random graphs. In *FOCS'81*, pages 364–375. IEEE, 1981.

[36] J. Lanlignel, O. Raynaud, and E. Thierry. Pruning graphs with digital search trees. application to distance hereditary graphs. In *STACS*, pages 529–541. Springer, 2000.

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

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

[39] R. McConnell and J. Spinrad. Linear-time modular decomposition and efficient transitive orientation of comparability graphs. In *SODA*, pages 536–545. Society for Industrial and Applied Mathematics, 1994.

[40] 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.

[41] 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.

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

[43] D. Paulusma, F. Slivovsky, and S. Szeider. Model counting for CNF formulas of bounded modular treewidth. *Algorithmica*, 76(1):168–194, 2016.

[44] W. Pulleyblank. Matchings and extensions. *Handbook of combinatorics*, 1:179–232, 1995.

[45] M. Rao. Clique-width of graphs defined by one-vertex extensions. *Discrete Mathematics*, 308(24):6157–6165, 2008.

[46] B. Reed. A semi-strong perfect graph theorem. *Journal of Combinatorial Theory, Series B*, 43(2):223–240, 1987.

[47] M. Tedder, D. Corneil, M. Habib, and C. Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *ICALP*, pages 634–645. Springer, 2008.

[48] 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.

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

[50] 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.