On improving matchings in trees, via bounded-length augmentations

Julien Bensmail, Valentin Garnero, Nicolas Nisse

On improving matchings in trees, via bounded-length augmentations[1]

Julien Bensmail[a], Valentin Garnero[a], Nicolas Nisse[a]

[a] *Université Côte d'Azur, CNRS, Inria, I3S, France*

Abstract

Due to a classical result of Berge, it is known that a matching of any graph can be turned into a maximum matching by repeatedly augmenting alternating paths whose ends are not covered. In a recent work, Nisse, Salch and Weber considered the influence, on this process, of augmenting paths with length at most k only. Given a graph G, an initial matching $M \subseteq E(G)$ and an odd integer k, the problem is to find a longest sequence of augmenting paths of length at most k that can be augmented sequentially from M. They proved that, when only paths of length at most $k = 3$ can be augmented, computing such a longest sequence can be done in polynomial time for any graph, while the same problem for any $k \geq 5$ is NP-hard. Although the latter result remains true for bipartite graphs, the status of the complexity of the same problem for trees is not known.

This work is dedicated to the complexity of this problem for trees. On the positive side, we first show that it can be solved in polynomial time for more classes of trees, namely bounded-degree trees (via a dynamic programming approach), caterpillars and trees where the nodes with degree at least 3 are sufficiently far apart. On the negative side, we show that, when only paths of length *exactly* k can be augmented, the problem becomes NP-hard already for $k = 3$, in the class of planar bipartite graphs with maximum degree 3 and arbitrary large girth. We also show that the latter problem is NP-hard in trees when k is part of the input.

Keywords: maximum matchings, bounded-length augmentations, trees.

1. Introduction

1.1. Matchings and augmentations

A *matching* M of a (simple undirected) graph G is a set of edges that are pairwise disjoint, i.e., no two edges of M share an end. Matchings are rather understood objects of graph theory. A common task, for a given graph G, is to find a matching of G that is maximum (with respect to the number of edges it includes). The maximum cardinality of a matching of G is denoted by $\mu(G)$. From a well-known result of Berge [Ber57], we know that a maximum matching of G can be obtained by *augmenting* paths arbitrarily, while it is possible.

The definition of an augmenting path is as follows. By any matching M of a graph G, every vertex is either *covered* (i.e., incident to an edge in M) or *exposed* (i.e., not incident to any edge in M). A path $P = (u_1, \cdots, u_p)$ of G is said *M-alternating* if no two consecutive edges of P are either both in M, or both not in M. Now, P is said *M-augmenting* if it is M-alternating and both u_1 and u_p are exposed (note that this implies that an augmenting path must have an odd number of edges). In the sequel, we will only call such a path *augmenting*, i.e., omitting M when it is clear from the context. In case P is augmenting, by *augmenting* it we mean removing from M every of its edges being in M, and adding to M all its other edges. Note that this operation, called an *augmentation*, results in another matching $M' = M \Delta E(P)$ (where Δ is the symmetric difference and $E(P)$ is the set of edges of P) such that $|M'| = |M| + 1$.

Precisely, Berge's Theorem states that a matching M in a graph G is maximum if and only if there are no M-augmenting paths [Ber57]. Building on Berge's result, Edmonds, via his Blossom

[1] The results of this paper consist of a part of an extended abstract that have been presented in LAGOS 2017 [BGN+17].

Algorithm [Edm65], later proved that augmenting paths in any graph G can be found in polynomial time, thus that $\mu(G)$ can be computed in polynomial time. One key idea behind Berge and Edmonds' results is that, when converging towards a maximum matching via performing augmentations, the choice of those performed augmentations is not crucial, as they will necessarily lead to a maximum matching. This does not remain true when one is allowed to augment paths with bounded length only, as pointed out in [NSW15].

1.2. Bounded-length augmentations

Throughout this paper, a $(\leq k)$-*augmentation* is an augmentation where the augmented path has (odd) length at most k. For a given graph G and a matching M of G, we denote by $\mu_{\leq k}(G, M)$ the maximum size of a matching that can be reached from M by performing $(\leq k)$-augmentations. These notions lead to the main problem this paper is interested in:

$(\leq k)$-Matching Problem ($MP^{\leq k}$)
Input: A graph G, and a matching M of G.
Question: What is the value of $\mu_{\leq k}(G, M)$?

The problem can be equivalently formulated as follows. Given a graph G and a matching $M = M_0$ of G, the goal is to compute the maximum length r of a sequence $\mathcal{S} = (P_1, \cdots, P_r)$ of paths, each of length at most k, such that: for every $1 \leq i \leq r$, P_i is an M_{i-1}-augmenting path and $M_i = M_{i-1}\Delta E(P_i)$ (i.e., M_i is obtained from M_{i-1} by augmenting P_i). We say that $\mathcal{S}$ *starts* from M and *results* in M_r. Note that $|M_i| = |M_0| + i$ and so $\mu_{\leq k}(G, M) = |M| + r$.

Although $\mu(G)$ can be determined in polynomial time for any graph G, intriguingly determining $\mu_{\leq k}(G, M)$ is an NP-hard problem in general [NSW15]. More precisely, a dichotomy result is provided concerning $MP^{\leq k}$, where the dichotomy is with respect to k. Namely, it is proved that $MP^{\leq 3}$ can be solved in polynomial time, while $MP^{\leq k}$ is NP-hard for every $k \geq 5$ [NSW15].

For the cases where $k \geq 5$, one may naturally wonder whether $MP^{\leq k}$ becomes polynomial-time solvable for particular classes of graphs. [NSW15] also made a first step towards that direction by showing that the NP-hardness result above holds even when the input graph is assumed to be planar, bipartite and of maximum degree 3. One open question, though, is whether this result also extends to trees, or whether $MP^{\leq k}$ (with $k \geq 5$) can be solved in polynomial time for trees. This is precisely the central topic considered in this paper. Note that it is only known that $MP^{\leq k}$ can be solved in polynomial time in the class of paths [NSW15].

Related work. The problem of finding a maximum matching in bipartite graphs has been extensively studied (in particular, because it is a special case of a network flow problem). It is well known that it can be solved in polynomial time (e.g., using the Hungarian method [Kuh55]). The first algorithm for solving the maximum matching problem in polynomial time in general graphs is due to Edmonds [Edm65]. Then, many work has been dedicated to designing more efficient algorithms [HK73, MV80, DP14]. In particular, the algorithms in [HK73, MV80] are based on augmenting paths in the non-decreasing order of their lengths. Such a method gives a good approximation since augmenting only the paths of length at most $2k - 3$ provides a $(1 - 1/k)$-approximation of the maximum matching [HK73].

The problem of finding a matching by augmenting bounded-length paths has also been studied in the context of wireless networks. In particular, it provides simple distributed algorithms to compute the scheduling of transmissions with interference [WS05, BSS09].

1.3. Results in this paper

In this paper, we provide both positive and negative results about the complexity of $MP^{\leq k}$ in trees. On the positive side, we show, in Section 2, that $MP^{\leq k}$ can be solved, for any odd $k \geq 5$, in polynomial time in several classes of trees. Via a dynamic programming approach, we first prove that $MP^{\leq k}$ is linear-time solvable in the class of trees with maximum degree Δ. That is, $MP^{\leq k}$ is FPT for these graphs when parameterized by $k + \Delta$. Generalizing the arguments for the path case, we then provide polynomial-time algorithms for *k-sparse trees*, i.e., trees where the nodes with degree at least 3 are at distance more than k, and caterpillars

Our negative results are related to the following thoughts. While trying to prove some hardness result concerning the tree instances of $MP^{\leq k}$ (for $k \geq 5$), we ran into the issue that allowing

augmentations of length up to k is a very permissive thing, which, at least in the case of trees, makes the design of a consistent NP-hardness proof not obvious. In Section 3, we thus study the behaviour of all those considerations in the context where only augmentations of length **exactly** k are allowed. In other words, we consider the following problem (where the notations and terminology are derived from others above, in the obvious way):

$(= k)$-Matching Problem ($MP^{=k}$)
Input: A graph G, and a matching M of G.
Question: What is the value of $\mu_{=k}(G, M)$?

While $MP^{\leq k}$ is NP-hard for every odd $k \geq 5$, we prove that $MP^{=k}$ is NP-hard for every odd $k \geq 3$, even when restricted to planar bipartite graphs with maximum degree 3 and arbitrarily large girth. We then focus on trees, and show that $MP^{=k}$ is NP-hard in trees when k is part of the input.

2. Augmenting matchings via ($\leq k$)-augmentations

In this section, we prove that, for any odd $k \geq 5$, $MP^{\leq k}$ can be solved in polynomial time for trees with bounded degree, k-sparse trees, and caterpillars. We will (sometimes implicitly) make use of the following three general statements:

Claim 2.1. *When executing a sequence of augmentations, a covered vertex cannot become exposed.*

Claim 2.2. *After augmenting a path P, all vertices of $V(P)$ are covered.*

Claim 2.3. *Let G be a graph, M be a matching of G, and $\mathcal{S} = (P_1, \cdots, P_r)$ be a sequence of $(\leq k)$-augmentations starting from M. Assume P_i and P_{i+1} are disjoint for some $i < r$. Then, $(P_1, \cdots, P_{i-1}, P_{i+1}, P_i, P_{i+2}, \cdots, P_r)$ is a sequence of $(\leq k)$-augmentations resulting in the same matching as $\mathcal{S}$.*

2.1. Bounded-degree trees

Using dynamic programming, we prove, in the following result, that $MP^{\leq k}$ can be solved in linear time for trees with maximum degree Δ, assuming both k and Δ are fixed. In other words, we show that $MP^{\leq k}$ is FPT when parametrized by $k + \Delta$.

Theorem 2.4. *Let k be a fixed odd integer. Let T be a tree with maximum degree Δ and M be a matching of T. Then, $\mu_{\leq k}(T, M)$ can be computed in time $O(f(k+\Delta) \cdot |V(T)|)$ for some decidable function f.*

Proof. Let T be any n-node tree with maximum degree Δ. Let k be an odd integer and let M be a matching of T. We present an algorithm that computes $\mu_{\leq k}(T, M)$ and the corresponding sequence of paths to be augmented in time $f(\Delta, k) \cdot n$ for some computable function f, i.e., in linear time when both k and Δ are fixed parameters. The algorithm proceeds by dynamic programming from the leaves to an arbitrary root $r \in V(T)$. For any node $v \in V(T)$, let T_v be the subtree of T rooted in v, let X_v be the subtree of T induced by all nodes at distance at most k from v and let P_v be the set of all paths of length at most k with nodes in X_v. Note that X_v has bounded size in Δ and k and so $|P_v|$ is $O(1)$ when Δ and k are fixed parameters.

Precisely, for every node $v \in V(T)$ and for every sequence $\mathcal{S}$ of distinct paths in P_v, the algorithm computes a matching with maximum size in T_v that can be computed from the initial matching M by a sequence of augmenting paths of length at most k in T using the ones in $\mathcal{S}$ (respecting the order of $\mathcal{S}$). In other words, the algorithm computes a matching $M_{v,\mathcal{S}}$ of T_v that is obtained from M by a sequence of augmenting paths $\mathcal{S}'$ (in T) such that $\mathcal{S}$ is a subsequence of $\mathcal{S}'$ (i.e., all paths in $\mathcal{S}$ appear in $\mathcal{S}'$ in the same order, not necessarily consecutively; Note also that some paths of $\mathcal{S}$ may be subpaths of paths in $\mathcal{S}'$) and that $M_{v,\mathcal{S}}$ has maximum size for these properties. By definition, the maximum solution $M_{r,\mathcal{S}}$ obtained for the root r (where the maximum is taken over all possible sequences in P_r) corresponds to an optimal solution for $MP^{\leq k}$.

Note that, for a given node $v \in V(T)$, the number of such partial solutions $M_{v,\mathcal{S}}$ that must be computed for v is $O(1)$ when k and Δ are fixed parameters. It only remains to prove that, for

any node v and any sequence $\mathcal{S}$, the desired solution can be computed in constant time from the solutions (the tables of the dynamic program) of the children of v.

If v is such that T_v is a star (all children of v are leaves) then, for every sequence $\mathcal{S}$ of paths as defined above, it is sufficient to check that the paths in $\mathcal{S}$ are compatible in T_v with the initial matching M and are pairwise compatible in T_v (note that we *a priori* do not assume that the paths in $\mathcal{S}$ are augmenting paths but we have to check that it is the case at the moment when they have to be augmented). This clearly can be done in constant time since the number of these paths is $O(1)$. If the paths in $\mathcal{S}$ are compatible, then $M_{v,\mathcal{S}}$ is the matching obtained by augmenting the paths in $\mathcal{S}$. Otherwise, the returned solution is $\emptyset$ which represents the fact that augmenting the paths in $\mathcal{S}$ is not a feasible solution.

More generally, for $v \in V(T)$ and every sequence $\mathcal{S}$ of paths as defined above, the algorithm proceeds as follows. For every non-leaf child v_i of v, and for every partial solution $M_{v_i,\mathcal{S}_i}$ obtained from a sequence $\mathcal{S}_i$ in the table of v_i, we check the compatibility of the $\mathcal{S}_i$'s with $\mathcal{S}$ in T_v. The algorithm then returns a best solution $M_{v,\mathcal{S}}$ obtained from the combinations of the children's solutions. Since they are $O(1)^\Delta$ such combinations, this still can be done in constant time. □

2.2. Sparse trees

In this section, we provide a polynomial-time algorithm for solving $MP^{\leq k}$ in the cases where the graph is a k-sparse tree. Let us introduce a few terminology beforehand. The nodes with degree at least 3 are called *b-nodes*. A path is called a *b-path* if it contains at least one b-node, while it is called an *a-path* otherwise.

As we need numerous preliminary results in order to present the algorithm, let us give a first intuition of how to get a matching of size $\mu_{\leq k}(T, M)$ in a k-sparse tree T starting from a matching M. Since the b-nodes of T are far apart (i.e., a $(\leq k)$-augmenting path cannot go through two b-nodes), T can be seen as a concatenation of several subdivided stars $T_1, \cdots, T_q$ glued along some of their branches. Hence, a sequence of $(\leq k)$-augmentations in T is made up of subsequences of $(\leq k)$-augmentations in the T_i's, that eventually have to get combined somehow. One of the main goals throughout this section is thus to comprehend how to get, via $(\leq k)$-augmentations, a matching of size $\mu_{\leq k}(S, M)$ for any subdivided star S and initial matching M.

A subdivided star, generally speaking, can be seen as a combination of paths attached at one node. Solving $MP^{\leq k}$ in a path can be done easily, as first shown by Nisse, Salch and Weber [NSW15], as it suffices to go from left to right and augment paths of close exposed nodes as they are encountered. In this section, we prove that, in a subdivided star S, a matching of size $\mu_{\leq k}(S, M)$ can be attained, roughly speaking, by performing at most one $(\leq k)$-augmentation through the b-node, and several $(\leq k)$-augmentations along the branches (i.e., apply the path algorithm onto a forest of paths). In particular, the augmentation through the b-node is proved to be necessary in a very particular context only.

We now start off by proving several lemmas. In what follows, given two paths Q and P, we note $Q \cap P$ for $V(P) \cap V(Q)$, i.e., we consider vertex-intersections.

Lemma 2.5. *Let T be a k-sparse tree, M be a matching, and $\mathcal{S} = (P_1, \cdots, P_r)$ be a sequence of $(\leq k)$-augmentations starting from M. Then, two b-paths in $\mathcal{S}$ intersect only if they contain the same b-node.*

Proof. Let $1 \leq i < j \leq r$, and let P_i and P_j be two b-paths containing the b-nodes u and v, respectively. Because T is k-sparse, no path in $\mathcal{S}$ can contain more than one b-node. For purpose of contradiction, let us assume that $u \neq v$ and $P_i \cap P_j \neq \emptyset$. Since $P_i \cap P_j \neq \emptyset$, there cannot be any b-node on the path from u and v since, otherwise, u and v would be at distance at least $2k + 2$ from each other and P_i and P_j (of length at most k each) could not intersect. Hence, P_i and P_j must intersect on the path Q between u and v, that consists only of nodes of degree 2. In particular, there must be an end x of P_j that is in $Q \cap P_i$. On the one hand, x must be exposed since P_j is an augmenting path. On the other hand, after P_i has been augmented, x is covered. A contradiction. □

The following lemmas are devoted to prove that, in sparse trees, augmentation sequences can be restricted to have some specific structure. First, we show that any sequence of augmentations can be transformed into an equivalent sequence that first augments b-paths, and then a-paths.

Lemma 2.6. *Let T be a k-sparse tree, M be a matching, and $\mathcal{S} = (P_1, \cdots, P_r)$ be a sequence of $(\leq k)$-augmentations starting from M. Let $1 \leq i < r$ be such that P_i is an a-path and P_{i+1} is a b-path. Then, there exist two a-paths A and B such that $A \cap B = \emptyset$, at most one of them, say A, is a b-path and $\mathcal{S}' = (P_1, \cdots, P_{i-1}, A, B, P_{i+2}, \cdots, P_r)$ is a sequence of $(\leq k)$-augmentations resulting in the same matching as $\mathcal{S}$.*

Proof. By Claim 2.3, the lemma holds if $P_i \cap P_{i+1} = \emptyset$. Let us assume that $P_i \cap P_{i+1} \neq \emptyset$. If P_i is included in P_{i+1}, then we can augment the two paths of $P_i \Delta P_{i+1}$ (where Δ is the symmetric difference) instead of P_i and P_{i+1}. Clearly at most one of these paths is a b-path and it can be augmented first. Otherwise (if P_i not included in P_{i+1}) then let x be the end of P_{i+1} belonging to $V(P_i)$ (this x must exist since P_i is an a-path not included in P_{i+1}). Then x must be covered after the augmentation of P_i, contradicting the fact that P_{i+1} is augmenting. □

Note that repeating the argument of Lemma 2.6, we can ensure that there exists an optimal sequence of augmentations that augments the b-paths first. We now show that any sequence of augmentations can be transformed into an equivalent sequence where all a-paths are vertex-disjoint.

Lemma 2.7. *Let T be a tree, M be a matching, and $\mathcal{S} = (P_1, \cdots, P_r)$ be a sequence of $(\leq k)$-augmentations starting from M. Let $1 \leq i < r$ be such that P_i and P_{i+1} are a-paths. Then, there exist two a-paths A and B such that $A \cap B = \emptyset$ and $\mathcal{S}' = (P_1, \cdots, P_{i-1}, A, B, P_{i+2}, \cdots, P_r)$ is a sequence of $(\leq k)$-augmentations resulting in the same matching as $\mathcal{S}$.*

Proof. If $P_i \cap P_{i+1} = \emptyset$, then $A = P_i$ and $B = P_{i+1}$ satisfy the lemma. Hence, let us assume that $P_i \cap P_{i+1} \neq \emptyset$. Let $P_i = (v_a, \cdots, v_b)$ and $P_{i+1} = (v_1, \cdots, v_c)$. After augmenting P_i, the nodes v_a and v_b are covered, and before augmenting P_{i+1}, the nodes v_1 and v_c were exposed. Now, since P_i and P_{i+1} are a-paths, it implies that $V(P_i) \subset V(P_{i+1})$ and, thus, $P_{i+1} = (v_1, \cdots, v_a, \cdots, v_b, \cdots, v_c)$. Then, setting $A = (v_1, \cdots, v_a)$ and $B = (v_b, \cdots, v_c)$ proves the lemma. □

From previous lemmas and claims, we get:

Corollary 2.8. *Let T be a k-sparse tree with b-nodes $\{v_1, \cdots, v_o\}$, M be a matching, and $\mathcal{S} = (P_1, \cdots, P_r)$ be a sequence of $(\leq k)$-augmentations starting from M. Then, there is a sequence $\mathcal{S}'$ of $(\leq k)$-augmentations starting from M, resulting in the same matching as $\mathcal{S}$, that consists of, in order:*

- *a sequence of b-paths each containing one of the v_i's, such that two b-paths containing different v_i's do not intersect, and moreover the b-paths containing a same v_i are consecutive, and*
- *a sequence of a-paths that are pairwise vertex-disjoint and vertex-disjoint from the previous augmented b-paths.*

A sequence satisfying the properties of Corollary 2.8 is called *partially-well structured*. Next, we characterize the "structure" of the b-paths augmented in a sparse tree. First, let us consider the case when a b-node is not initially covered by the matching.

Lemma 2.9. *Let T be a k-sparse tree, M be a matching, and $\mathcal{S} = (P_1, \cdots, P_r)$ be a sequence of $(\leq k)$-augmentations starting from M. Let v be any b-node of T. If v is exposed by M, then there exists a partially-well structured sequence $\mathcal{S}'$ of $(\leq k)$-augmentations starting from M, resulting in the same matching as $\mathcal{S}$, and such that at most one path of $\mathcal{S}'$ contains v. Moreover, this sequence can be obtained by modifying only the b-paths of $\mathcal{S}$ that contain v.*

Proof. By Corollary 2.8, we may assume that $\mathcal{S}$ is partially-well structured. If at most one path of $\mathcal{S}$ contains v, then we are done. Otherwise, let P and Q be the first two paths of $\mathcal{S}$ that contain v. Since $\mathcal{S}$ is partially-well structured, P and Q are consecutive in $\mathcal{S}$; let $P = P_i$ and $Q = P_{i+1}$ for some $i < r$. Since v is exposed by M, necessarily P must be a path that ends in v and, because T is k-sparse and P has length at most k, all nodes of P (but v) have degree at most 2 in T. Let us set $P = (v_1, \cdots, v_j = v)$. After having augmented P, the edge $v_{j-1}v_j$ belongs to the matching and all nodes of P are covered. Therefore, the only way for Q to be an augmenting path containing v is to fully contain P. So $Q = (w_1, \cdots, w_s, v_1, \cdots, v_j = v, u_1, \cdots, u_h)$ where all nodes of Q but v have degree at most 2. Hence, the sequence $\mathcal{S}' = (P_1, \cdots, P_{i-1}, (w_1, \cdots, w_s, v_1), (v, u_1, \cdots, u_h), P_{i+2}, \cdots, P_r)$

is a sequence of ($\leq k$)-augmenting paths resulting in the same matching as $\mathcal{S}$ and containing strictly less paths containing v. Reordering $\mathcal{S}'$ using Lemma 2.6, we obtain a partially-well structured sequence containing the same paths as in $\mathcal{S}'$. Repeating this process until at most one path contains v allows us to obtain a sequence satisfying the lemma. □

Now, we characterize the "structure" of the b-paths containing a b-node that is covered by the initial matching. Let T be a k-sparse tree with an initial matching M and let v be a b-node that is covered by M. Let $\mathcal{S} = (P_1, \cdots, P_r)$ be a sequence of ($\leq k$)-augmentations all of which pass through v. The path P_1 must go from T_{i_1}, the component of $T - \{v\}$ that contains the node matched with v by M, to some other component T_{i_2} of $T - \{v\}$. After having augmented P_1, the node matched with v must be in T_{i_2}. Then, P_2 must go from T_{i_2} to another component T_{i_3} of $T - \{v\}$, i.e., $i_3 \neq i_2$ but i_3 may be equal to i_1. Going on that way, $\mathcal{S}$ is fully characterized by the sequence $(T_{i_1}, \cdots, T_{i_{r+1}})$ of components of $T - \{v\}$ such that, for every $1 \leq j \leq r$, we have $i_j \neq i_{j+1}$. Precisely, for any $j \leq r$, after having augmented $P_1, \cdots, P_j$, the path P_i goes from the exposed node of T_{i_j} that is the closest to v, to the exposed node of T_{j+1} that is the closest to v. We say that such a sequence *starts* in T_{i_1} and *finishes* in $T_{i_{r+1}}$. Let us call the sequence $\mathcal{S}$ to be *unlooping* if the indices in $\{i_1, \cdots, i_{r+1}\}$ are pairwise distinct. Note that, in particular, the length of such a sequence is bounded by the degree of v minus 1.

Lemma 2.10. *Let T be a k-sparse tree, M be a matching, and $\mathcal{S} = (P_1, \cdots, P_r)$ be a sequence of ($\leq k$)-augmentations starting from M. Let v be any b-node of T. If v is covered by M, then there exists a partially-well structured sequence $\mathcal{S}'$ of ($\leq k$)-augmentations starting from M, resulting in the same matching as $\mathcal{S}$, and such that the subsequence of b-paths containing v is unlooping. Moreover, this sequence can be obtained by modifying only the b-paths of $\mathcal{S}$ that contain v.*

Proof. By Corollary 2.8, we may assume that $\mathcal{S}$ is partially-well structured. Let $\mathcal{S}' = (P'_1, \cdots, P'_s)$ be the subsequence of (consecutive) b-paths of $\mathcal{S}$ that contain v. If $\mathcal{S}'$ is unlooping, then we are done. Otherwise, let $(T_{i_1}, \cdots, T_{i_{s+1}})$ be the sequence of components of $T - \{v\}$ that characterizes $\mathcal{S}'$ (as in the paragraph above). Let $y < x \leq s$ be such that the indices in $\{i_y, \cdots, i_x\}$ are pairwise distinct and $i_y = i_{x+1}$. We show that, in $\mathcal{S}'$ (and so in $\mathcal{S}$), we can replace the paths $P_y, \cdots, P_x$ by as many a-paths (i.e., not passing through v). For every $y \leq j \leq x$, let d_j be the end of P_j in the component T_{i_j} and let f_j be its end in the component $T_{i_{j+1}}$. For every $y \leq j < x$, let Q_j be the path between f_j and d_{j+1}, and let Q_x be the path between d_y and f_x. Then, the sequence $\mathcal{S}'' = (P'_1, \cdots, P'_{y-1}, Q_y, \cdots, Q_x, P'_{x+1}, \cdots, P'_s)$ results in the same matching as $\mathcal{S}'$. Moreover, the paths $Q_y, \cdots, Q_x$ are a-paths, and according to Lemma 2.6, they can be re-ordered to obtain a partially-well structured sequence. In this latter sequence, the number of times that the b-paths containing v are "looping around v" has been reduced by 1. Hence, repeating this process leads to the desired sequence of ($\leq k$)-augmentations. □

In what follows, we deal with sequences of augmentations having a particular shape. Let T be a k-sparse tree and M be a matching. Denote by $K = \{c_1, \cdots, c_p\}$ and $U = \{u_1, \cdots, u_q\}$ the sets of b-nodes respectively covered and exposed by M. A sequence $\mathcal{S}$ of ($\leq k$)-augmentations starting from M is said *well-structured* if it consists of, in order:

- a sequence of b-paths containing the u_i's, where each u_i is contained in at most one b-path,
- for every $i \leq p$, there is one unlooping (possibly empty) sequence of b-paths containing c_i (in particular, the b-paths containing c_i are consecutive), and
- a sequence of a-paths.

Moreover, every two paths of the whole sequence intersect if and only if they contain the same b-node. Clearly, a well-structured sequence is also partially-well structured.

All results proved so far allow to easily derive the following theorem.

Theorem 2.11. *Let T be a k-sparse tree, M be a matching, and $\mathcal{S}$ be a sequence of ($\leq k$)-augmentations starting from M. Then, there exists a well-structured sequence of ($\leq k$)-augmentations starting from M and resulting in the same matching as $\mathcal{S}$.*

Lemma 2.12. *Let T be a k-sparse tree, M be a matching, and $\mathcal{S}$ be a sequence of $(\leq k)$-augmentations starting from M and resulting in a matching M'. Let v be any node that is contained in a path of $\mathcal{S}$. Finally, if v is not a b-node and v is exposed by M, let us assume that it belongs to a unique path of $\mathcal{S}$ (in particular, it is the case if v is an exposed leaf of T). Then, there exists a sequence of $(\leq k)$-augmentations starting from M such that none of its paths contains v, and resulting in a matching M'' of size at least $|M'| - 1$. Moreover, if v is a b-node, then M and M'' only differ in some edges of the paths of $\mathcal{S}$ containing v.*

Proof. By Theorem 2.11, we may assume that $\mathcal{S}$ is well-structured.

- Let us first assume that v is a b-node. If at most one path of $\mathcal{S}$ contains v, then removing it (if it exists) leads to the desired result. Hence, we may assume that there is a unique subsequence $\mathcal{S}' = (P_1, \cdots, P_r)$ of b-paths of $\mathcal{S}$ containing v. Moreover, this subsequence is unlooping. Let $(T_1, \cdots, T_{r+1})$ be the sequence of components of $T - \{v\}$ that characterizes $\mathcal{S}'$. For every $1 \leq i \leq r$, let d_i be the end of P_i in the component T_i and let f_i be its end in the component T_{i+1}. For every $1 \leq i < r$, let Q_i denote the path between f_i and d_{i+1}. Then, replacing the sequence $\mathcal{S}'$ by $(Q_1, \cdots, Q_{r-1})$ in $\mathcal{S}$ results in a matching of size $|M'| - 1$ and that differs from M only on the path between d_1 and f_r. Moreover, no paths of the resulting sequence contain v.

- If v is not a b-node, then we consider several cases. If v is contained in an a-path of $\mathcal{S}$, then it is sufficient to remove this path from $\mathcal{S}$ (because it is disjoint from any other path, since $\mathcal{S}$ is well-structured). Otherwise, v belongs to some path of an unlooping subsequence $\mathcal{S}' = (P_1, \cdots, P_r)$ of $\mathcal{S}$. For every $1 \leq i \leq r$, let d_i be the starting node of P_i and let f_i be its end. Let $j \leq r$ be the first index such that $v \in V(P_j)$.

  If $v \neq f_j$ (or $j = r$), it is sufficient to replace $\mathcal{S}'$ by $(P_1, \cdots, P_{j-1}, P'_{j+1}, \cdots, P'_r)$ where, for every $j < i \leq r$, we denote by P'_i the path between d_i and f_{i-1}.

  Finally, it is not possible that $v = f_j$ and $j < r$, since otherwise v would be exposed by M and contained in two paths (P_j and P_{j+1}) of $\mathcal{S}'$ (and so of $\mathcal{S}$), contradicting the hypothesis. □

Lemma 2.13. *Let T be a k-sparse tree, M be a matching, and v be a b-node covered by M. Let $\mathcal{B} = \{T_1, \cdots, T_r\}$ be any subset of at least two components of $T - \{v\}$. It can be decided in polynomial time (in $V(T)$) whether it exists an unlooping sequence of $(\leq k)$-augmentations intersecting v, starting in T_1, finishing in T_r and only passing through components of $\mathcal{B}$.*

Proof. A trivial necessary condition is that v is matched by M with a node in T_1 (otherwise no sequence of augmenting paths containing v can start in T_1). Hence, we may assume that it is the case.

Consider the following auxiliary digraph with vertex-set $\{v_1, \cdots, v_r\}$. For every $2 \leq j \leq r$, add an arc from v_1 to v_j if the first exposed node of T_1 (the one that is closest to v) is at distance at most k from the first exposed node of T_j (in which case, the path between these two nodes is augmenting and has length at most k). Then, for every $2 \leq i < j \leq r$, add an arc from v_i to v_j if the second exposed node of T_i is at distance at most k from the first exposed node of T_j. It is easy to show that a desired sequence exists if and only if there is a directed (simple) path from v_1 to v_r in that digraph, which can be checked in polynomial time by a BFS algorithm. □

Before going on, we present the following lemma that leads to a much simpler proof than that in [NSW15] of the linear-time algorithm to compute $\mu_{\leq k}(P, M)$ in paths P. Let P be a path graph and M be a matching. Let $v_1, \cdots, v_r$ be the exposed nodes in order, say, from "left to right". First, as in [NSW15], if two consecutive exposed nodes (i.e., v_i and v_{i+1} for some $i < r$) are at distance strictly more than k, then all edges between them can be removed. Repeating this deletion process while possible, we are then left with a set of disjoint paths (subpaths of P), whose every two consecutive exposed nodes are at distance at most k. The following lemma expresses a simple algorithm to deal with this kind of instances.

Lemma 2.14. *Let P be a path, and M be matching such that any two consecutive exposed nodes are at distance at most k. Let $v_1, \cdots, v_r$ denote the exposed nodes in order, say, from "left to right". Then, the sequence of paths $(P_i)_{1 \leq i \leq \lfloor r/2 \rfloor}$, where P_i denotes the path going from v_{2i-1} to v_{2i}, is a sequence of $(\leq k)$-augmentations starting from M and resulting in a matching of size $\mu_{\leq k}(P, M)$.*

Proof. We only need to show that the sequence is optimal. Each time a path is augmented in P, two exposed nodes become covered. Therefore, any sequence of augmentations contains at most $\lfloor r/2 \rfloor$ paths. Since the size of the final matching is $|M|$ plus the number of augmented paths, the proposed sequence is optimal. □

In the context of sparse trees, the previous proof motivates us to consider the parity of the number of exposed nodes in the "branches". Let T be a tree rooted in some node r. A b-node of T is called a *lowest b-node* if it has no other b-nodes as descendants. Any component of $T - \{v\}$ that consists of descendants of v is called a *child-branch* at v. The component of $T - \{v\}$ that contains the parent of v is called the *parent-branch* at v. For a matching M of T, a child-branch B at v is called *even* (resp., *odd*) if the number of exposed nodes in B is even (resp., odd). Finally, the pair (T, M) is said *clean* if there are no two exposed nodes at distance strictly more than k such that all nodes on the path between them are covered and have degree 2.

Lemma 2.15. *Let T be a k-sparse tree, M be a matching such that (T, M) is clean, and $\mathcal{S}$ be a well-structured sequence of $(\leq k)$-augmentations starting from M and resulting in a matching M'. Let v be any lowest b-node. Let $\mathcal{S}'$ be the subsequence of paths of $\mathcal{S}$ containing v. If $\mathcal{S}'$ starts or ends in an even child-branch at v, then there exists a sequence of $(\leq k)$-augmentations starting from M such that none of its paths contains v, and resulting in a matching M'' of size at least $|M'|$.*

Proof. Let $\mathcal{S}$ be a well-structured optimal sequence for (T, M) having the properties described in Theorem 2.11. If no path of $\mathcal{S}$ contains both v and a node in some even child-branch, then the result clearly holds.

Now assume that at least one path intersects v. Since $\mathcal{S}$ is well-structured, there is a subsequence $\mathcal{S}' = (P_1, \cdots, P_r)$ of b-paths of $\mathcal{S}$ containing v (possibly $r = 1$). Moreover, this subsequence is unlooping. Let $(T_1, \cdots, T_{r+1})$ be the sequence of components of $T - \{v\}$ that characterizes $\mathcal{S}'$. Moreover, either T_1 or T_{r+1} is an even child-branch. Let us assume it is T_1, the other case being symmetric, and let $v_1, \cdots, v_{2h}$ be the exposed nodes (by M) on T_1, with v_1 being the closest to v. For every $1 \leq i \leq r$, let d_i be the end of P_i in the component T_i and let f_i be its end in the component T_{i+1}. For every $1 \leq i < r$, let Q_i denote the path (of T_{i+1}) between f_i and d_{i+1}. Replacing, in $\mathcal{S}$, the paths $(P_1, \cdots, P_r)$ and all paths of $\mathcal{S}$ strictly included in T_1, by the paths $(Q_1, \cdots, Q_{r-1})$ and by the paths from v_{2i-1} to v_{2i}, for every $1 \leq i \leq h$, we obtain a sequence of $(\leq k)$-augmentations none of which contains v, and resulting in a matching of same size as M. Hence, we are back to the first case. □

Lemma 2.16. *Let T be a rooted k-sparse tree, M be a matching such that (T, M) is clean, and $\mathcal{S}$ be an optimal well-structured sequence of $(\leq k)$-augmentations starting from M. Let v be any lowest b-node with child-branches $B_1, \cdots, B_h$. For any child-branch B_i at v, let x_i be its number of exposed nodes and let $\delta_i = 1$ if B_i is an even branch and $\delta_i = 0$ otherwise. Assume that $\mathcal{S}$ contains an unlooping sequence $(P_1, \cdots, P_s)$ of paths containing v, characterized by a sequence $\mathcal{B}$ of components of $T - \{v\}$. Let Z be the number of paths augmented by $\mathcal{S}$ that intersect the subtree T_v of T rooted at v. Then:*

1. *If $\mathcal{B}$ contains only child-branches at v, w.l.o.g., $\mathcal{B} = (B_1, \cdots, B_{s+1})$, then, $Z = 1 - \delta_1 - \delta_{s+1} + \sum_{1 \leq i \leq h} \lfloor (x_i)/2 \rfloor$.*
2. *If $\mathcal{B}$ contains (but neither starts nor ends by) the parent-branch at v, w.l.o.g., $\mathcal{B} = (B_1, \cdots, B_{s+1})$ and the parent-branch is B_q $(1 < q < s+1)$, then $Z = 2 - \delta_1 - \delta_{s+1} + \sum_{1 \leq i \leq h, i \neq q} \lfloor (x_i)/2 \rfloor$.*
3. *If $\mathcal{B}$ starts or finishes by the parent-branch at v, w.l.o.g., $\mathcal{B} = (B_1, \cdots, B_{s+1})$ and the parent-branch is B_1, then $Z = 1 - \delta_{s+1} + \sum_{1 < i \leq h} \lfloor (x_i)/2 \rfloor$.*

In particular, Z depends only on the parity of the first and final child-branches. Furthermore, if no paths of $\mathcal{S}$ contain v, then $Z = \sum_{1 \leq i \leq h} \lfloor (x_i)/2 \rfloor$.

Proof. We prove the result for the first item, the proof for the second and third items being similar. The number of paths augmented in T_v by $\mathcal{S}$ is

$$s + \lfloor (x_1 - 1)/2 \rfloor + \lfloor (x_{s+1} - 1)/2 \rfloor + \sum_{1<i<s+1} \lfloor (x_i - 2)/2 \rfloor + \sum_{s+2 \leq i \leq h} \lfloor x_i/2 \rfloor.$$

Indeed, augmenting the s paths $P_1, \cdots, P_s$ "consumes" one exposed node in both B_1 and B_{s+1} and two exposed nodes in every branch $B_2, \cdots, B_s$. Furthermore, since $\mathcal{S}$ is an optimal well-structured sequence, and according to Lemma 2.14, the number of augmented paths in each branch is the number of remaining exposed nodes divided by 2 (floor). Reorganizing the sum gives the result.

The last sentence of the statement is trivial, and follows e.g., from Lemma 2.14. □

We are now ready to introduce a polynomial-time algorithm for solving $MP^{\leq k}$ in the case of k-sparse trees. For purpose of simplification, the algorithm we introduce only computes a matching of size $\mu_{\leq k}(G, M)$. However, it can be modified easily, by memorizing the paths that have been augmented, so that it also provides a corresponding sequence of $(\leq k)$-augmentations.

Theorem 2.17. *For any k-sparse tree T and matching M,* SparseTreeAlgorithm*(k, T, M) computes in polynomial time (in $|V(T)|$) a matching of size $\mu_{\leq k}(T, M)$ by performing $(\leq k)$-augmentations starting from M.*

Proof. The algorithm PathAlgorithm mentioned in Case 1 is the linear-time algorithm for solving $MP^{\leq k}$ in the case of paths mentioned in Lemma 2.14. SparseTreeAlgorithm is a recursive algorithm. The fact that it runs under polynomial time is obvious since all tests can indeed be performed in polynomial time (in Cases 6, 7 and 8, this is true by Lemma 2.13), and, in any of the cases, the algorithm is recursively applied on vertex-disjoint (strictly smaller) subtrees of T.

Let us prove the correctness of SparseTreeAlgorithm, i.e., the fact that we eventually obtain a matching of size $\mu_{\leq k}(T, M)$ from M, by induction on the size of T. The result clearly holds in Cases $1, 2, 3$. In all remaining cases, the proof follows the same scheme: we start from a well-structured sequence $\mathcal{S}$ of $(\leq k)$-augmentations starting from M and resulting in a matching of size $\mu_{\leq k}(T, M)$ (it exists by Theorem 2.11), and we show that the algorithm achieves a matching of (at least) the same size.

Case 4. In that case, v has only even child-branches. We prove the correctness when v is exposed, the case when v is covered being similar. We prove that there is a sequence of $(\leq k)$-augmentations starting from M and resulting in a matching of size $\mu_{\leq k}(T, M)$ that can be obtained by considering independently the child-branches $B_1, \cdots, B_s$ at v and $T - (\bigcup_{i \leq s} V(B_i))$. Since it is what is done by SparseTreeAlgorithm, the result holds by induction.

If $\mathcal{S}$ does not contain any path containing v and some node of a child-branch, then the result clearly holds. Otherwise, since all child-branches at v are even, by Lemma 2.15 there exists an optimal sequence not intersecting v.

Case 5. In that case, v is exposed and has at least one odd child-branch. Since $\mathcal{S}$ is well-structured, there is at most one path P of it that contains (and ends in) v. Let B be any odd child-branch at v, and let $v_1, \cdots, v_{2h+1}$ be the exposed nodes (by M) on B, with v_1 being the closest to v. Because $\mathcal{S}$ is well-structured, by Lemma 2.14 there are, in $\mathcal{S}$, at most h augmenting paths strictly included in B. Set $v = v_0$. We simply replace P and these paths in B by the paths P_i from v_{2i} to v_{2i+1} for every $0 \leq i \leq h$. The obtained sequence can clearly be obtained by first augmenting P_0 and then considering all components of $T - P_0$ independently. This is what SparseTreeAlgorithm does. Hence, the result holds by induction.

Case 6. In that case, there is an unlooping sequence $\mathcal{S}' = (P_1, \cdots, P_s)$ of paths containing v, characterized by a sequence $(T_1, \cdots, T_{s+1})$ of components of $T - \{v\}$. Moreover, T_1 and T_{s+1} are odd child-branches at v, and $T_2, \cdots, T_s$ are even child-branches at v.

In that case, SparseTreeAlgorithm first augments the paths in $\mathcal{S}'$ and then in the components of $T - \bigcup_{i \leq s} V(P_i)$ independently. Let $B_1, \cdots, B_h$ be the child-branches at v and, for every $i \leq h$, let x_i be the number of exposed nodes of B_i. Let Q be the parent-branch at v. Said differently, SparseTreeAlgorithm computes an optimal solution for Q (by induction) and, by Lemma 2.16, augments $1 + \sum_{i \leq h} \lfloor x_i/2 \rfloor$ paths in T_v. Let us compare this number of augmentations with the one obtained by the optimal sequence $\mathcal{S}$. There are several cases to consider.

SparseTreeAlgorithm(k, T, M)

Require: A k-sparse tree T rooted in any arbitrary node and a matching M

// *Case 1*
if T is a path **then**
  Return PathAlgorithm(k, T, M)
// *Case 2*
if there exists a leaf edge $uv \in M$ **then**
  Return SparseTreeAlgorithm(k, $T - \{u, v\}$, M)
// *Case 3*
if there are two exposed nodes at distance strictly more than k such that all nodes on the path P between them are covered and have degree 2 **then**
  Let T_1, T_2 denote the trees obtained when removing the internal nodes of P from T
  Return SparseTreeAlgorithm(k, T_1, M) and SparseTreeAlgorithm(k, T_2, M)
Let v be a lowest b-node of T
// *Case 4*
if all child-branches at v are even **then**
  if v is exposed **then**
    Let $T_1, \cdots, T_x$ denote the trees obtained when removing the edges between v and its children
  if v is covered by $\{v, w\} \in M$ **then**
    Let $T_1, \cdots, T_x$ denote the trees obtained when removing v and w
  Return SparseTreeAlgorithm(k, T_i, M) for every $i = 1, \cdots, x$
// *Case 5*
if v is exposed **then**
  Let w be the exposed node closest to v on some odd child-branch at v
  Augment the path P from v to w
  Let T_1, T_2 denote the trees of $T - V(P)$
  Return SparseTreeAlgorithm(k, T_1, M) and SparseTreeAlgorithm(k, T_2, M)
// *Case 6*
if there exists an unlooping sequence $\mathcal{S} = (P_1, \cdots, P_s)$ of $(\leq k)$-augmentations intersecting v, starting from an odd child-branch at v, finishing in an odd child-branch at v, without passing through the parent-branch at v **then**
  // *this can be checked in polynomial time by Lemma 2.13*
  Augment the paths in $\mathcal{S}$
  Let $T_1, \cdots, T_x$ denote the trees of $T - (\bigcup_{i \leq s} V(P_i))$
  Return SparseTreeAlgorithm(k, T_i, M) for every $i = 1, \cdots, x$
// *Case 7*
if there exists an unlooping sequence $\mathcal{S} = (P_1, \cdots, P_s)$ of $(\leq k)$-augmentations intersecting v, starting from an odd child-branch at v, finishing in an odd child-branch at v (passing through the parent-branch at v) **then**
  // *this can be checked in polynomial time by Lemma 2.13*
  Augment the paths in $\mathcal{S}$
  Let $T_1, \cdots, T_x$ denote the trees of $T - (\bigcup_{i \leq s} V(P_i))$
  Return SparseTreeAlgorithm(k, T_i, M) for every $i = 1, \cdots, x$
// *Case 8*
if there exists an unlooping sequence $\mathcal{S} = (P_1, \cdots, P_s)$ of $(\leq k)$-augmentations intersecting v, starting from an odd child-branch at v, finishing in the parent-branch at v, OR starting from the parent-branch at v and finishing in an odd child-branch at v **then**
  // *this can be checked in polynomial time by Lemma 2.13*
  Augment the paths in $\mathcal{S}$
  Let $T_1, \cdots, T_x$ denote the trees of $T - (\bigcup_{i \leq s} V(P_i))$
  Return SparseTreeAlgorithm(k, T_i, M) for every $i = 1, \cdots, x$
// *Case 9*
Let $T_1, \cdots, T_x$ denote the trees obtained when removing the edges between v and its children
Return SparseTreeAlgorithm(k, T_i, M) for every $i = 1, \cdots, x$

- No path of $\mathcal{S}$ contains v and some node of a child-branch at v. Then $\mathcal{S}$ deals with $Q \cup \{v\}$ and $T_v - \{v\}$ independently. By Lemma 2.16, at most $\sum_{i \leq h} \lfloor x_i/2 \rfloor$ paths may be augmented in T_v. Moreover, by Lemma 2.12 in $Q \cup \{v\}$ (v being a leaf of the tree induced by $Q \cup \{v\}$, it satisfies the hypothesis of Lemma 2.12), the number of paths that may be augmented is at most the maximum number of paths that may be augmented in Q plus 1. Overall, $\mathcal{S}$ augments at most the same number of paths as SparseTreeAlgorithm.
- No path of $\mathcal{S}$ contains v and some node of the parent-branch at v. Then $\mathcal{S}$ deals with Q and T_v independently (as SparseTreeAlgorithm). By Lemma 2.16, at most $1 + \sum_{i \leq h} \lfloor x_i/2 \rfloor$ paths may be augmented. So $\mathcal{S}$ does the same as SparseTreeAlgorithm (possibly augmenting a different unlooping sequence around v).
- $\mathcal{S}$ contains an unlooping sequence around v that starts or finishes in the parent-branch at v. Then, by Lemma 2.16, $\mathcal{S}$ augments at most $1 + \sum_{1 < i \leq h} \lfloor (x_i)/2 \rfloor$ paths intersecting T_v and then computes an optimal solution in a subtree of Q (precisely an optimal solution of Q minus the nodes of the paths of the unlooping sequence). Therefore, the solution computed by SparseTreeAlgorithm has at least the same size.
- $\mathcal{S}$ contains an unlooping sequence around v that starts and finishes in some child-branches at v, and passes through the parent-branch Q at v. Let $(P_1, \cdots, P_s)$ be the unlooping sequence, let $i < s$ such that P_i and P_{i+1} intersect Q, and let x (resp., y) be the end of P_i (resp., P_{i+1}) in Q. Finally, let Q' be the component of $T - \{y\}$ that does not contain v and let $Y = T - Q$.

  By Lemma 2.16, $\mathcal{S}$ augments at most $2 + \sum_{1 \leq i \leq h} \lfloor (x_i)/2 \rfloor$ paths in Y and then computes independently an optimal solution of Q' augmenting z paths. Since combining an optimal solution of Q' with the augmentation of the path between x and y provides a solution for Q, SparseTreeAlgorithm augments at least $z + 1$ paths in Q. Overall, SparseTreeAlgorithm augments at least $2 + z + \sum_{1 \leq i \leq h} \lfloor (x_i)/2 \rfloor$ paths in T. Hence, the solution computed by the algorithm has at least the same size as the one of $\mathcal{S}$.

Case 7. In that case, there is an unlooping sequence $\mathcal{S}' = (P_1, \cdots, P_s)$ of paths containing v, characterized by a sequence $(T_1, \cdots, T_{s+1})$ of components of $T - \{v\}$. Moreover, T_1 and T_{s+1} are odd child-branches at v, there exists $j \leq s$ such that T_j is the parent-branch at v, and T_i is an even child-branch at v for every $1 < i \leq s$, $i \neq j$. Let x be the end of P_{j-1} in T_j and y be the end of P_j in T_j. Note that because T is k-sparse, neither x nor y is a b-node.

Also, there exists no unlooping sequence starting and finishing in some odd child-branches and avoiding the parent-branch. In the present case, SparseTreeAlgorithm first augments the paths in $\mathcal{S}'$ and then deals with the components of $T - \bigcup_{i \leq s} V(P_i)$ independently. Let $B_1, \cdots, B_h$ be the child-branches at v and, for every $i \leq h$, let x_i be the number of exposed nodes of B_i. Let Q be the parent-branch at v. Said differently, SparseTreeAlgorithm computes an optimal solution for $Q \setminus P_j$ (by induction) and, by Lemma 2.16, augments $2 + \sum_{i \leq h} \lfloor x_i/2 \rfloor$ paths intersecting T_v. Let us compare this number of augmentations with the one obtained by the optimal sequence $\mathcal{S}$. There are several cases to be considered.

- No path of $\mathcal{S}$ contains v and some node of a child-branch at v. Note that, since $\mathcal{S}'$ exists, v was initially matched by M with a node in one of its child-branches. Therefore, $\mathcal{S}$ cannot contain either a path that contains v and some node of the parent-branch at v. Hence, $\mathcal{S}$ deals with Q and $T_v - \{v\}$ independently. By Lemma 2.16, at most $\sum_{i \leq h} \lfloor x_i/2 \rfloor$ paths may be augmented.
  Because y is not a b-node, $Q_y = T \setminus P_j$ is the parent-branch of y and, moreover, y is a leaf of the tree induced by $Q_y \cup \{y\}$. Hence, Lemma 2.12 applies to the tree induced by $Q_y \cup \{y\}$ (with initial matching $M \cap E(Q_y \cup \{y\})$) and node y and so, the number of paths that may be augmented in $Q_y \cup \{y\}$ is at most the maximum number of paths that may be augmented in Q_y plus 1.
  Note then that all nodes (but x and y) on the path from x to y are covered by M and that x is not a b-node. Therefore, the number of paths that may be augmented in the

parent-branch Q_x of x (with initial matching $M \cap E(Q_x)$) is the same as the number of paths that may be augmented in $Q_j \cup \{y\}$.

Since x is a leaf of the tree induced by $Q_x \cup \{x\}$, Lemma 2.12 applies to the tree induced by $Q_x \cup \{x\}$ (with initial matching $M \cap E(Q_x \cup \{x\})$) and node x and so, the number of paths that may be augmented in $Q_x \cup \{x\}$ is at most the maximum number of paths that may be augmented in Q_x plus 1.

Let $p \in V(Q)$ be the parent of v and note that all nodes (except x but including p if $p \neq x$) on the path from p to x are covered by M. Therefore, the number of paths that may be augmented in the parent-branch Q (with initial matching $M \cap E(Q)$) is the same as the number of paths that may be augmented in $Q_x \cup \{x\}$.

Altogether, the number of paths that may be augmented in Q is at most the maximum number of paths that may be augmented in $Q \setminus P_j$ plus 2. Overall, $\mathcal{S}$ augments at most the same number of paths as SparseTreeAlgorithm

So, because no unlooping sequence avoiding the parent-branch exists, if $\mathcal{S}$ contains paths intersecting v, it must be via unlooping sequences intersecting the parent-branch.

- $\mathcal{S}$ contains an unlooping sequence around v that starts or finishes in the parent-branch at v (in particular, it starts or finishes in x). Then, by Lemma 2.16, $\mathcal{S}$ augments at most $1+\sum_{1<i\leq h}\lfloor(x_i)/2\rfloor$ paths intersecting T_v and then computes an optimal solution in the subtree Q' of Q (precisely the component of $Q \setminus \{x\}$ containing y). By Lemma 2.12 (applied to y as in the previous subcase), an optimal solution of Q' is at most 1 plus an optimal solution for $Q \setminus P_j$. Therefore, the solution computed by SparseTreeAlgorithm has at least the same size.
- Finally, let us assume that $\mathcal{S}$ contains an unlooping sequence around v that starts and finishes in some child-branches at v, and passes through the parent-branch Q at v. By Lemma 2.16, $\mathcal{S}$ augments at most $2+\sum_{1\leq i\leq h}\lfloor(x_i)/2\rfloor$ paths and, independently, computes an optimal solution in $Q \setminus P_j$. So $\mathcal{S}$ does the same as SparseTreeAlgorithm (possibly augmenting a different unlooping sequence around v). Hence, SparseTreeAlgorithm obtains a solution of same size.

Case 8. In that case, there is an unlooping sequence $\mathcal{S}' = (P_1, \cdots, P_s)$ of paths containing v, characterized by a sequence $(T_1, \cdots, T_{s+1})$ of components of $T - \{v\}$. Moreover, T_1 is an odd child-branch at v, $T_2, \cdots, T_s$ are even child-branches at v, and T_{s+1} is the parent-branch at v (the case when $\mathcal{S}'$ starts by the parent-branch is similar). Let x be the end of P_s in T_{s+1}.

There exists no unlooping sequence starting or finishing in some odd child-branch and that does not finish or start in the parent-branch. SparseTreeAlgorithm first augments the paths in $\mathcal{S}'$ and then the components of $T - \bigcup_{i\leq s} V(P_i)$ independently. Let $B_1, \cdots, B_h$ be the child-branches at v and, for every $i \leq h$, let x_i be the number of exposed nodes of B_i. Let Q be the parent-branch at v. Said differently, SparseTreeAlgorithm computes an optimal solution for $Q \setminus P_s$ (by induction) and, by Lemma 2.16, augments $1+\sum_{i\leq h}\lfloor x_i/2\rfloor$ paths intersecting T_v. Let us compare this number of augmentations with the one obtained by the optimal sequence $\mathcal{S}$. There are several cases to be considered.

- No path of $\mathcal{S}$ contains v and some node of a child-branch at v. Note that, since $\mathcal{S}'$ exists, v was initially matched by M with a node in one of its child-branches. Therefore, $\mathcal{S}$ cannot contain a path that contains v and some node of the parent-branch at v. Hence, $\mathcal{S}$ deals with Q and $T_v - \{v\}$ independently. By Lemma 2.16, at most $\sum_{i\leq h}\lfloor x_i/2\rfloor$ paths may be augmented. Moreover, by Lemma 2.12 applied for x (x satisfies the hypothesis of Lemma 2.12 since it is the last node of the last path P_s of $\mathcal{S}'$), in Q, the number of paths that may be augmented is at most the maximum number of paths that may be augmented in $Q \setminus P_s$ plus 1. Overall, $\mathcal{S}$ augments at most the same number of paths as SparseTreeAlgorithm.
- Now, because of the hypothesis, if $\mathcal{S}$ contains paths intersecting v, then we may assume that $\mathcal{S}$ contains an unlooping sequence around v that starts or finishes in the parent-branch at v (in particular, it starts or finishes in x). Then, by Lemma 2.16, $\mathcal{S}$ augments

at most $1+\sum_{1<i\leq h}\lfloor(x_i)/2\rfloor$ paths intersecting T_v and then computes an optimal solution in the subtree $Q \setminus P_s$. Thus, SparseTreeAlgorithm obtains a solution of same size.

Case 9. In the last case, no unlooping sequence starting or finishing by an odd child-branch at v exists. In that case, SparseTreeAlgorithm deals independently with $T_v - \{v\}$, in which case, by Lemma 2.16, it augments $\sum_{i\leq h}\lfloor x_i/2\rfloor$ paths, and then computes an optimal (by induction) solution for $(T - T_v) \cup \{v\}$. As previously, it can be checked that no optimal sequence can augment strictly more paths. □

2.3. Caterpillars

Recall that a *caterpillar* is a tree consisting of one main path (which we call *spine* throughout), such that every node either belongs to that path, or is adjacent to a node of that path. Prior to exhibiting a polynomial-time algorithm to solve $MP^{\leq k}$ for caterpillars, we first need to show that a maximum matching can be attained by performing specific augmentations.

We start by pointing out that, in our context, we may consider caterpillars with maximum degree at most 3. This follows from the following more general statement, which actually concerns all trees.

Lemma 2.18. *Let T be a tree, M be a matching, and v be a node of T adjacent to $d \geq 2$ leaves $u_1, \cdots, u_d$. Then, for any sequence of $(\leq k)$-augmentations starting from M, at most one of the u_i's can be an end of an augmented path.*

Proof. This follows from the fact that once a leaf gets covered by a matching, it cannot loose this property, recall Claim 2.1. Furthermore, once one of the leaves u_i is covered, there is no more augmenting path containing v. □

Let C be a caterpillar, and M be a matching of C. We denote by $\tilde{C}$ a caterpillar obtained from C by removing some leaves, as follows. For every non-leaf node v of the spine of C:

- if v is adjacent to a covered leaf u, then we remove all leaves adjacent to v, but u;
- otherwise, we remove all leaves adjacent to v, but one.

From Lemma 2.18, we deduce the following.

Corollary 2.19. *Let C be a caterpillar, and M be a matching. Let M' be the restriction of M to $\tilde{C}$. Then $\mu_{\leq k}(C, M) = \mu_{\leq k}(\tilde{C}, M')$.*

As a consequence of Corollary 2.19, we may narrow down our attention on caterpillars with maximum degree 3, i.e., we may assume that every node of the spine is adjacent to at most one leaf. Therefore, we may denote the nodes of any considered caterpillar C in the following way. Let $(u_1, \cdots, u_\ell)$ be the spine of C (where u_1 and u_ℓ are leaves of T). For any $i \in \{2, \cdots, \ell - 1\}$ we denote by u'_i the leaf (not belonging to the spine) adjacent to u_i, if it exists. Note that u'_2 and $u'_{\ell-1}$ do not exist. Note that this labelling of the nodes is not consistent for the subcaterpillars of C. That is, when removing a node u_i of C, the node u'_{i+1} might belong to the spine of one connected component of $C - \{u_i\}$. Hence, in what follows, for every caterpillar we consider, we implicitly assume that the labelling of its nodes respects the convention above.

We start by pointing out that, in a caterpillar C with maximum degree 3, starting from a matching M, a matching with size $\mu_{\leq k}(C, M)$ can be obtained by performing non-intersecting $(\leq k)$-augmentations.

Lemma 2.20. *Let C be a caterpillar with maximum degree 3, M be a matching, and $\mathcal{S} = (P_1, \cdots, P_r)$ be a sequence of $(\leq k)$-augmentations starting from M. Assume P_i and P_{i+1} intersect, for some $i < r$. Then, there exist two disjoint paths A, B such that $(P_1, \cdots, P_{i-1}, A, B, P_{i+2}, \cdots, P_r)$ is a sequence of $(\leq k)$-augmentations resulting in the same matching as $\mathcal{S}$.*

Proof. This follows from arguments that are similar to that we used to prove Lemma 2.7 (with the exception that, here, it might be that none of P_i and P_{i+1} strictly includes the other). Consider, as A and B, the exactly two paths of $(P_i \cup P_{i+1}) - E(P_i \cap P_{i+1})$. Note that if P_i and P_{i+1} are vertex-disjoint, then $\{A, B\} = \{P_i, P_{i+1}\}$. It can be checked that A and B are disjoint, and, because C is a caterpillar, that they have length at most k (because P_i and P_{i+1} do). More precisely, each of these paths A and B is either:

- a subpath of P_{i+1}, or
- obtained from P_i or P_{i+1} by replacing a subpath of P_i (hence of length at least 1) and a leaf edge (in C) of P_{i+1}.

Furthermore, executing $(P_1, \cdots, P_{i-1}, A, B, P_{i+2}, \cdots, P_r)$ from M results in the same matching as $\mathcal{S}$. □

Let C be a caterpillar with maximum degree 3 and M be a matching. If u_1 is covered by M then no augmenting path can contain u_1 nor u_2 (which is also obviously covered). Hence, removing u_1 and u_2 (recall that u_2' does not exist) from C and removing u_1u_2 from M results in a new instance (C', M') such that solving the problem in (C', M') is equivalent to solving the problem in (C, M). Therefore, we may assume that u_1, which is a leaf, is exposed. When referring to the *closest exposed node* of u_1 in C, we mean the exposed node (different from u_1) that is at shortest distance from u_1. Note that there may be two candidates u_i' and u_{i+1} (for some $i > 1$) as such closest exposed nodes. In that case, we consider u_i' as the closest exposed node from u_1. By definition, the path from u_1 to its closest exposed node is augmenting. Furthermore, we may suppose that u_1 and its closest exposed node are at distance at most k. Otherwise, no augmenting path of length at most k can contain nodes in $\{u_1, \cdots, u_i, u_i'\}$. In such case (if u_i' is at distance more than k from u_1), C and M could be simplified by removing all nodes in $\{u_1, \cdots, u_i, u_i'\}$, and applying the algorithm on the remaining caterpillar. Now, by the *left-most augmentation* in C, we refer to the augmentation joining u_1 and its closest exposed node.

From Lemma 2.20, we now prove that, for a caterpillar C and a matching M, we can obtain, via $(\leq k)$-augmentations, a matching of size $\mu_{\leq k}(C, M)$ by repeatedly performing left-most augmentations.

Lemma 2.21. *Let C be a caterpillar with maximum degree 3 and M be a matching. Then, a matching of size $\mu_{\leq k}(C, M)$ can be obtained by repeatedly performing left-most $(\leq k)$-augmentations starting from M.*

Proof. According to Lemma 2.20 and Claim 2.3, a matching of size $\mu_{\leq k}(C, M)$ can be obtained, starting from M, by a sequence $\mathcal{S} = (P_1, \cdots, P_r)$ of pairwise disjoint $(\leq k)$-augmentations. Furthermore, by Claim 2.3, we may assume that $P_1 < \cdots < P_r$, i.e., that the right-most end of P_i is located on the left of the left-most end of P_{i+1}, for every two consecutive P_i, P_{i+1}. To prove the lemma, it suffices to show that a matching of size $\mu_{\leq k}(C, M)$ can be obtained via a sequence $(P_1', P_2, \ldots, P_r)$ of $(\leq k)$-augmentations, where $P_1' < P_2 < \cdots < P_r$ and P_1' is the left-most augmentation. If P_1 is already the left-most augmentation, then we set $P_1' = P_1$. Otherwise, let P_1' be the left-most $(\leq k)$-augmentation in the subcaterpillar containing all nodes located on the left of the right-most end of P_1. Then we just repeat, by induction, the same arguments with the subcaterpillar containing all nodes located on the right of the right-most end of P_1', and with the sequence $(P_2, \cdots, P_r)$ of $(\leq k)$-augmentations. □

Corollary 2.19 and Lemma 2.21 directly yield a linear-time algorithm for solving $MP^{\leq k}$ in a caterpillar C. First, we may suppose that $C = \tilde{C}$ as otherwise the instance could be simplified. Then, a matching of size $\mu_{\leq k}(C, M)$ can be obtained by repeatedly applying the left-most $(\leq k)$-augmentation, for the notion of left and right we have used throughout this section.

Theorem 2.22. *Let C be a caterpillar, and M be a matching. Then, $\mu_{\leq k}(C, M)$ can be computed in linear time (in $|V(C)|$).*

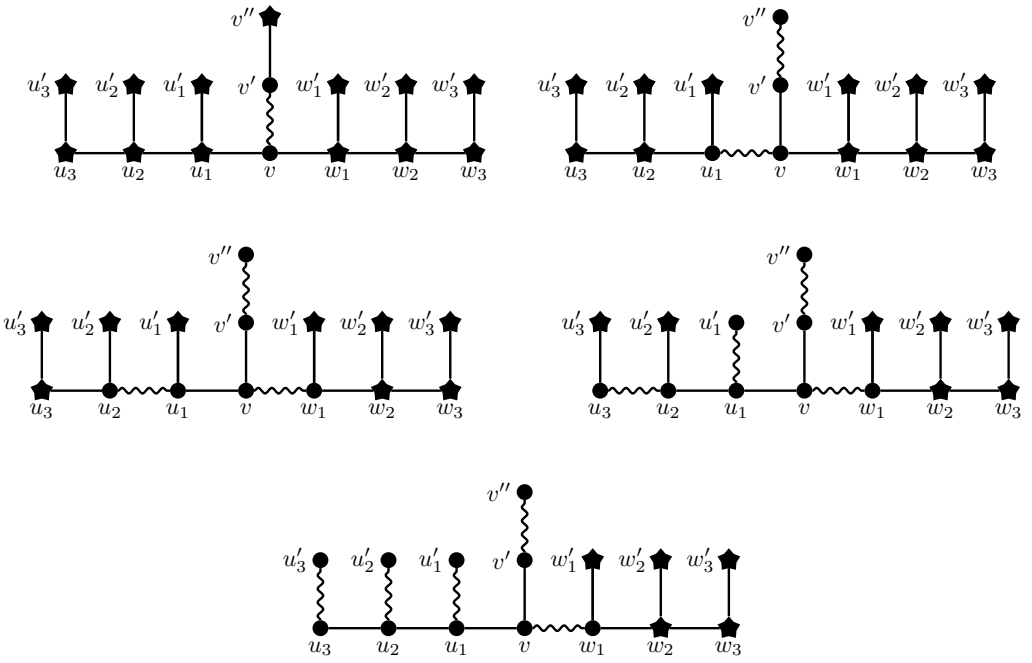

Figure 1: The 3-choice gadget H, and the way the initial matching $M = \{vv'\}$ is being $(= 3)$-augmented until a matching of size $\mu_{=3}(H, M)$ is attained (from top to bottom, and left to right). Star vertices are exposed vertices. Wiggly edges are edges of the matching.

3. Augmenting matchings via $(= k)$-augmentations

We now investigate the consequences, on $MP^{\leq k}$, of restricting ourselves to augmentations of paths with length exactly k. In Section 3.1, we start by showing that $MP^{=k}$ is, in general, NP-hard for every $k \geq 3$. The case of trees is considered in Section 3.2, wherein we prove that, in this context, the problem is NP-hard when k is part of the instance.

3.1. Complexity of $MP^{=k}$ in general graphs (for fixed k)

First, we would like to point out that, in the reduction of [NSW15] for showing that $MP^{\leq k}$ is NP-hard for every $k \geq 5$, only $(= k)$-augmentations can actually be performed in the reduced graphs, which can be checked by considering the pairs of exposed vertices. Therefore, from this reduction we directly get the following.

Theorem 3.1. *[NSW15] $MP^{=k}$ is NP-hard for every $k \geq 5$, even when restricted to instances where G is a planar bipartite graph with maximum degree* 3*.*

Since $MP^{\leq 3}$ was shown to be polynomial-time solvable [NSW15], we do not get the NP-hardness of $MP^{=3}$ right away, in a similar way, and instead have to provide a proof.

Theorem 3.2. *$MP^{=3}$ is NP-hard, even when restricted to instances where G is a planar bipartite graph with maximum degree* 3 *and arbitrarily large girth.*

Before going to the proof of Theorem 3.2, we first describe the gadgets to be used, as well as some of their behaviours. We start off with *choice gadgets* (see Figure 1 for an illustration). For any $\ell \geq 2$, the *ℓ-choice gadget* is obtained from a path $(u_\ell, u_{\ell-1}, \cdots, u_1, v, w_1, \cdots, w_{\ell-1}, w_\ell)$ on $2\ell + 1$ vertices by 1) joining a new pendant vertex u'_i to u_i, for every $i = 1, \cdots, \ell$, 2) joining a new pendant vertex w'_i to w_i, for every $i = 1, \cdots, \ell$, then 3) joining a new pendant vertex v' to v, and 4) joining a new pendant vertex v'' to v'. We call vv' the *middle-edge* of the choice gadget. Throughout this section, assuming it is clear which choice gadget we are dealing with, we refer to its vertices and edges using the terminology we have just introduced.

One property of interest of choice gadgets is the following:

Observation 3.3. *Let H be an ℓ-choice gadget, and $M = \{vv'\}$ be a matching of H. Then, for any matching M' of size $\mu_{=3}(H, M)$ obtained by performing $(= 3)$-augmentations starting from M, we have either:*

1. $M' = \{v'v''\} \cup \{vw_1\} \cup \bigcup_{i=1}^{\ell} \{u_i u_i'\}$, *or*
2. $M' = \{v'v''\} \cup \{vu_1\} \cup \bigcup_{i=1}^{\ell} \{w_i w_i'\}$.

In particular, we have $\mu_{=3}(H, M) = \ell + 2$.

Proof. Consider a maximum sequence $\mathcal{S} = (P_1, \cdots, P_q)$ of $(= 3)$-augmentations that can be performed starting from M, and denote by M' the resulting matching. Throughout this proof, for every $1 \leq i \leq q$, we denote by $\mathcal{S}(M, i)$ the matching obtained from M by augmenting sequentially the paths $P_1, \cdots, P_i$. Note that $\mathcal{S}(M, q) = M'$.

Recall that a $(= 3)$-augmentation can only be performed on a path of length 3 in which only the middle-edge belongs to the matching. For this reason, since $M = \{vv'\}$, we have $P_1 = (v'', v', v, u_1)$ or $P_1 = (v'', v', v, w_1)$. Assume $P_1 = (v'', v', v, u_1)$ without loss of generality (the other case is symmetric). Because $v''v' \in \mathcal{S}(M, 1)$ and v'' has degree 1, no augmenting path can contain any of v'' and v'. So P_2 necessarily contains $w_1 v$ and vu_1. More precisely, P_2 may be either (w_1, v, u_1, u_2) or (w_1, v, u_1, u_1'). At this point, it can be noted that, since vw_1 and vu_1 belong to one of P_1 and P_2, they cannot belong to any P_i with $i \geq 3$. Indeed, whatever be the choice of P_2, after its augmentation, all the neighbours of v get covered, and, thus, no edge incident to v may belong to an augmenting path of length 3 from this point. This implies that none of w_2 and w_1' belongs to one of the P_i's, thus that the matching cannot be spread further towards the w_i's. We also note that if $P_2 = (w_1, v, u_1, u_1')$, then, in H and $\mathcal{S}(M, 2)$, there is no further $(= 3)$-augmentation, which means that the matching cannot propagate further towards the u_i's. So we necessarily have $P_2 = (w_1, v, u_1, u_2)$.

We thus have $\mathcal{S}(M, 2) = \{v'v'', vw_1, u_1u_2\}$, and the only augmenting $(= 3)$-paths are (u_1', u_1, u_2, u_2') and (u_1', u_1, u_2, u_3) (if $\ell \geq 3$). If $\ell = 2$, then necessarily $P_3 = (u_1', u_1, u_2, u_2')$, and we are done. If $\ell > 3$, we note that P_3 cannot be (u_1', u_1, u_2, u_2') as otherwise $\mathcal{S}(M, 3)$ would have no augmenting $(= 3)$-paths in H. So, in that case, $P_3 = (u_1', u_1, u_2, u_3)$. These arguments generalize as follows by induction on i (see Figure 1 for an illustration). For every $i = 2, \cdots, \ell + 1$, the only $(= 3)$-augmenting paths in H, assuming the current matching is $\mathcal{S}(M, i-1)$, are $(u_{i-1}', u_{i-1}, u_i, u_i')$ and $(u_{i-1}', u_{i-1}, u_i, u_{i+1})$. In case $i = \ell + 1$, actually only the first of these two paths exists, so $P_i = (u_{i-1}', u_{i-1}, u_i, u_i')$. Otherwise, i.e., $i = 2, \cdots, \ell$, we note that having $P_i = (u_{i-1}', u_{i-1}, u_i, u_i')$ would make $\mathcal{S}(M, i)$ have no augmenting $(= 3)$-paths in H. So we have $P_i = (u_{i-1}', u_{i-1}, u_i, u_{i+1})$ for every $i = 2, \cdots, \ell$, so that the matching can be spread further.

Under the assumption that $P_1 = (v'', v', v, u_1)$, we eventually get, assuming that $\mathcal{S}$ is maximum, that $M' = \{v'v''\} \cup \{vw_1\} \cup \bigcup_{i=1}^{\ell} \{u_i u_i'\}$. By symmetry, we note that having $P_1 = (v'', v', v, w_1)$ results in $M' = \{v'v''\} \cup \{vu_1\} \cup \bigcup_{i=1}^{\ell} \{w_i w_i'\}$. In both cases, we have $\mu_{=3}(H, M) = \ell + 2$, as claimed. □

In order to introduce the next type of gadgets, we need some additional terminology to deal with choice gadgets. Let H be a choice gadget. The vertices u_i' of H are called the *spike vertices* of H, while the edges $u_i u_i'$ incident to the spike vertices are called the *spike edges*. Each spike vertex or edge is numbered accordingly to the index i of the vertex u_i' (or w_i') it intersects. The vertices $u_1, u_1', u_2, u_2', \cdots$ form the *positive branch* of H, while the vertices $w_1, w_1', w_2, w_2', \cdots$ form the *negative branch*. Rephrased differently, Observation 3.3 says that, in a choice gadget, the "optimal" way to propagate the original matching is towards the spike edges of the positive branch only, or towards the spike edges of the negative branch only. In the first case, we say that the original matching has been propagated *positively*, while we say it has been propagated *negatively* otherwise.

We now introduce *variable gadgets*, that are combinations of choice gadgets connected in a cyclic fashion (see Figure 2 for an illustration). For any $m \geq 1$, the *m-variable gadget* is constructed as follows. The 1-variable gadget is the ℓ-choice gadget (the length ℓ of the choice gadget will be fixed later). Then, for any $m \geq 2$, the m-variable gadget is constructed by taking m ℓ-choice gadgets $H_0, \cdots, H_{\ell-1}$ and, for every $i = 0, \cdots, \ell - 1$, by identifying the first negative spike vertex of H_i

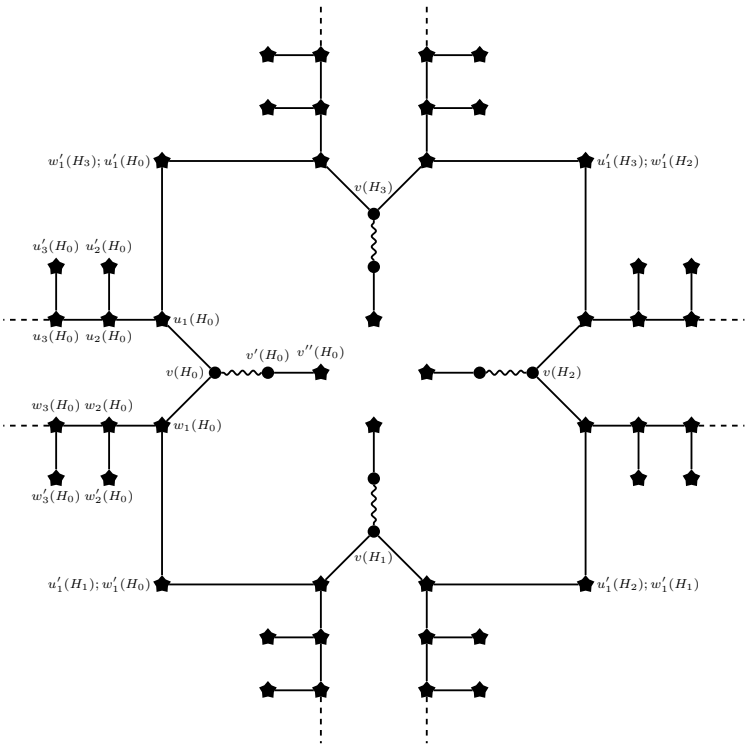

Figure 2: The 4-variable gadget. Star vertices are exposed vertices. Wiggly edges are edges of the matching.

and the first positive spike vertex of H_{i+1}, where the indexes are understood modulo m. Precisely, for any $j < m$ and any $i \leq \ell$, let, here and further, $u_i(H_j)$ denote the vertex u_i of the j^{th} copy H_j of the choice gadget $(v(H_j), v'(H_j), v''(H_j), u'_i(H_j), w_i(H_j), w'_i(H_j)$ are defined analogously). Hence, the variable gadget is obtained by identifying $w'_1(H_j)$ with $u'_1(H_{j+1})$ (modulo m) for every $j = 0, \cdots, m-1$.

The original matching of any variable gadget is the union of the original matchings of the choice gadgets constituting it (i.e., their middle-edges). By the *positive branches* (resp. *negative branches*) of H, we mean the m positive (resp. negative) branches of its underlying choice gadgets. Analogously, by referring to the *spike vertices* and *spike edges* of a variable gadget, we mean the spike vertices and edges of its underlying choice gadgets.

Note that we have not explicited the lengths of the $2m$ branches composing an m-variable gadget, i.e., ℓ still must be defined. However, we must now ensure that the behaviour described in Observation 3.3 remains valid after having combined the choice gadgets to form a variable gadget. Assuming these branches are "long enough", we prove it is the case.

Observation 3.4. *Let H be an m-variable gadget, and M be the original matching of H as described above. If ℓ (the length of the choice gadgets composing H) is at least 2, then, for any matching M' of size $\mu_{=3}(H, M)$ obtained by performing $(=3)$-augmentations starting from M, we have either:*

1. *all positive spike edges in M' and no negative spike edges in M', or*
2. *all negative spike edges in M' and no positive spike edges in M'.*

Proof. Let $H_0, \cdots, H_{m-1}$ be the m ℓ-choice gadgets composing H. We first prove that, because $\ell \geq 2$, any maximum sequence of $(=3)$-augmentations has no interest to perform $(=3)$-augmentations that intersect different choice gadgets. That is, a matching of size $\mu_{=3}(H, M)$ cannot be obtained by augmenting a $(=3)$-path having edges in two consecutive H_i's.

For purpose of contradiction, let us assume that there is an augmenting $(=3)$-path Q that intersects two choice gadgets, w.l.o.g., say H_0 and H_1 and that Q has two edges in H_0. Let us

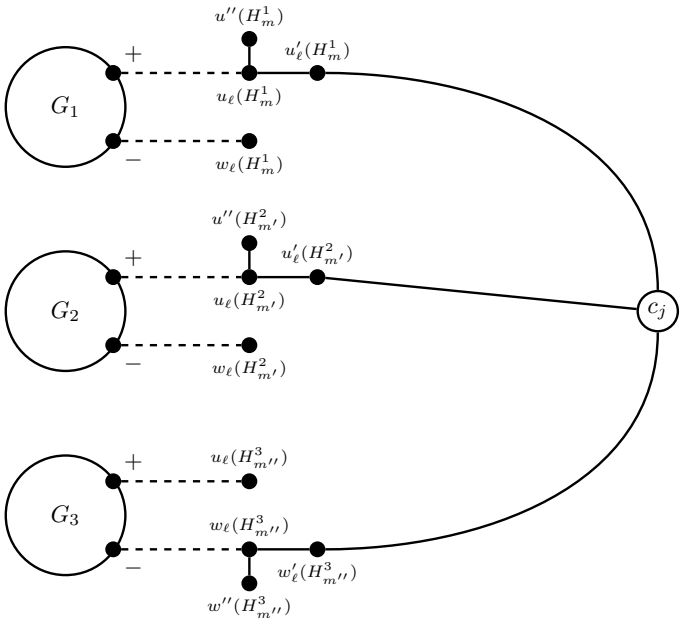

Figure 3: Illustration of the reduction in the proof of Theorem 3.2, for a formula Φ having a clause $C = (x_1 \vee x_2 \vee \overline{x_3})$, and C is the m^{th} (resp. m'^{th}, m''^{th}) clause containing x_1 (resp. $x_2, \overline{x_3}$).

assume that Q is the first (to be augmented) such path intersecting distinct choice gadgets. When Q is about to be augmented, $w_1(H_0)w'_1(H_0)$ must be in the current matching, a neighbour of $w_1(H_0)$ must be exposed, and $u_1(H_1)$ must be exposed. The only way to have reached such a situation is when, in H_0, the paths $(v''(H_0), v'(H_0), v(H_0), w_1(H_0))$ and $(u_1(H_0), v(H_0), w_1(H_0), w'_1(H_0))$ have been augmented, and only them. Moreover, only the path $(v''(H_1), v'(H_1), v(H_1), w_1(H_1))$ can have been augmented in H_1.

Therefore, Q must be $(w_2(H_0), w_1(H_0), w'_1(H_0), u_1(H_1))$. After having augmented Q, we note that, because all of $v''(H_0), v'(H_0), v(H_0), u_1(H_0), w_1(H_0)w_2(H_0), w'_1(H_0)$ are covered, there is, in H, no further $(= 3)$-augmentation including an edge of H_0. In H_1, we note that, since $u'_1(H_1)$ and $u_1(H_1)$ are covered, the only $(= 3)$-augmentation that can potentially be performed (if not already) in H_1 is $(v''(H_1), v'(H_1), v(H_1), w_1(H_1))$. According to these arguments, augmenting a $(= 3)$-path intersecting H_0 and H_1 leads to a final matching with at most three edges in H_0 and three edges in H_1, while by Observation 3.3 we could have achieved a matching with $\ell + 2 \geq 4$ edges in each of H_0 and H_1 by propagating the matching positively or negatively.

Following these observations, since $\ell \geq 2$, a matching of size $\mu_{=3}(H, M)$ can only be obtained, from M, by propagating the original matching of each H_i positively or negatively. The claim now follows from the fact that if, say, H_i has propagated its matching negatively, then H_{i+1} cannot propagate its matching positively, since the first negative spike edge of H_i and the first positive spike edge of H_{i+1} are adjacent. □

According to Observation 3.4, we can thus derive the notion of positive and negative propagations to variable gadgets with long branches: by propagating its matching *positively* (resp. *negatively*), we mean propagating the matching positively (resp. negatively) in all of its underlying choice gadgets.

We now have all ingredients in hand for proving Theorem 3.2.

Proof of Theorem 3.2. The proof is by reduction from 3-SAT. Namely, from a 3CNF formula Φ, we construct, in polynomial time, a graph G with an initial matching M such that, from a satisfiable truth assignment of the variables of Φ, we can deduce a matching of size $\mu_{=3}(G, M)$ to which M can be $(= 3)$-augmented, and vice versa.

The construction of G (and M) is mainly achieved by connecting several variable gadgets together. For every variable x_i of Φ, we add an m_i-variable gadget G_i to G, where

$$m_i = \max\{\#\text{ of distinct clauses that contain } x_i, \#\text{ of distinct clauses that contain } \overline{x_i}\},$$

and the choice gadgets underling G_i have length ℓ at least 2 (so that Observation 3.4 can apply). Let $H^i_0, \cdots, H^i_{m_i-1}$ be the choice gadgets composing G_i. For every last (i.e., the farthest from the middle-edge) positive (resp. negative) spike edge $u_\ell(H^i_j)u'_\ell(H^i_j)$ (resp. $w_\ell(H^i_j)w'_\ell(H^i_j)$) of every choice gadget H^i_j underlying G_i, we add a pendant vertex $u''(H^i_j)$ (resp. $w''(H^i_j)$) that we join to $u_\ell(H^i_j)$ (resp. $w_\ell(H^i_j)$).

For every clause C_j of Φ, we add a clause vertex c_j to G. Finally, we connect the variable gadgets and clause vertices in the following way (see Figure 3): for every variable x_i (resp. negated variable $\overline{x_i}$) and m^{th} distinct clause C_j containing x_i (resp. $\overline{x_i}$), we join c_j and the spike vertex $u'_\ell(H^i_m)$ (resp. $w'_\ell(H^i_m)$) from the m^{th} choice gadget H^i_m of G_i. Note that the order in which the clauses containing x_i are taken is not relevant.

We eventually define the original matching M of G as the union of the original matchings of the choice gadgets composing the variable gadgets. Clearly, the construction of G is achieved in polynomial time.

A matching M' of size $\mu_{=3}(G, M)$ of G that can be obtained starting from M via $(= 3)$-augmentations, is obtained as follows. Recall that the choice gadgets constituting the variable gadgets are assumed long, so that Observation 3.4 applies. First, for every variable gadget G_i, we have to propagate the original matching of G_i either positively or negatively. Thus, at this point, the maximum number of such $(= 3)$-augmentations that can be performed does not depend on the connexion with the clause vertices, but only on the lengths of the choice gadgets we used. So, from now on, we denote by α_M this number of $(= 3)$-augmentations. That is, by Observations 3.3 and 3.4, we have

$$\alpha_M = \sum_{i=1}^{n} m_i \cdot (\ell + 2)$$

where n is the number of variables in Φ. Now, for every clause C_j of Φ, we can potentially augment one of the at most three $(= 3)$-paths of the form $(c_j, u'_\ell(H^i_m), u_\ell(H^i_m), u''(H^i_m))$ or $(c_j, w'_\ell(H^i_m), w_\ell(H^i_m), w''(H^i_m))$ joined to c_j in G (where m is the index of the choice gadget of G_i that is associated to clause C_j), but this is only possible if one of the three variable gadgets incident to c_j propagated its matching positively (first case), or negatively (second case). The matching then cannot be propagated further through c_j: if, say, $u'_\ell(H^i_m)c_j$ belongs to the matching at some point, then it means that the $(= 3)$-path $(c_j, u'_\ell(H^i_m), u_\ell(H^i_m), u''(H^i_m))$ was augmented, hence that $u_\ell(H^i_m)$ is covered while $u'_\ell(H^i_m)$ has degree 2 (so no $(= 3)$-augmenting path including $u'_\ell(H^i_{m'})c_j$ nor $w'_\ell(H^i_{m'})c_j$ exists for every $m' \neq m$). So we have

$$\alpha_M \leq \mu_{=3}(G, M) \leq \alpha_M + \gamma,$$

where γ denotes the number of clauses appearing in Φ. In particular, the upper bound is attained whenever, for every variable gadget G_i of G, we can propagate its original matching appropriately, so that, for every clause vertex of G, an augmentation of an incident $(= 3)$-path can be performed.

We claim that $\mu_{=3}(G, M) = \alpha_M + \gamma$ if and only if Φ admits a satisfying truth assignment. To see this holds, consider that, in G, propagating the original matching of any G_i positively (resp. negatively) simulates the fact that variable x_i of Φ is set to *true* (resp. *false*) by some truth assignment. So that one of the final γ augmentations including the clause vertices of G can be performed, it needs one of its at most three incident variable gadgets to have propagated its matching the good way. By this, we mean positively $(=$ *true*$)$ if the clause contains the positive version of that variable, of negatively $(=$ *false*$)$ otherwise. Hence, the equivalence holds.

We conclude the proof by pointing out some possible modifications of the reduction above, which maintain the equivalence with 3-SAT. Let us first point out that G has maximum degree 3. Furthermore, since 3-SAT remains NP-hard for planar formulas Φ, we may assume that Φ is planar, in which case the reduced graph G is clearly planar as well, since the variable gadgets are planar (as illustrated in Figure 2).

We can also modify the reduction above so that G fulfils additional properties:

- It can be checked that all cycles of the reduced graph G go through variable gadgets, i.e., at some point, they enter a variable gadget via a spike edge, and exit it using another spike edge (possibly from the other branch). Since two variable gadgets G_i and G_{i+1} are connected by spike edges or by clause vertices (hence by paths of length 2), and the positive and negative first (or last) spike edges of any G_i are at even distance in G_i, the only way for a cycle of G to be of odd length is when, in the construction, we use a choice gadget with odd length (so that, in its branches, its first and last spike edges are at odd distance). So, by using choice gadgets of even length only, we can make sure that G is bipartite.

- Now, by increasing 1) the number of choice gadgets used to construct the G_i's, and 2) the length of these choice gadgets, we also increase the lengths of the cycles in G. So we can also make sure that G has arbitrarily large girth. □

3.2. Complexity of $MP^{=k}$ in trees (for non-fixed k)

In this section, we show that $MP^{=k}$ is NP-hard for trees when k is part of the input. That is, we show that

(=)-Matching Problem ($MP^{=}$)
Input: A graph G, a matching M of G, and an odd integer $k \geq 1$.
Question: What is the value of $\mu_{=k}(G, M)$?

is NP-hard, even for instances where G is a tree.

Theorem 3.5. *$MP^{=}$ is NP-hard, even when restricted to instances where G is a tree.*

Proof. The proof is by reduction from 3-SAT. From a 3CNF formula Φ, we construct a tree T with a matching M, such that Φ is satisfiable if and only if $\mu_{=k}(T, M)$ is equal to a specific value, where k depends on the number of variables and clauses of Φ. For the sake of understanding, we here describe the proof for $k = 100 \cdot (n + m) + 1$, where n is the number of distinct variables appearing in Φ and m is the number of distinct clauses of Φ, but the reduction could also be performed for much smaller values of k (but still dependent on $n + m$). Note in particular that $\lfloor \frac{1}{10}k \rfloor$ is even.

The tree T, and its original matching M, are obtained by combining several gadgets together. These gadgets will each be equipped with an initial matching, so that M will be the union of these matchings.

The first type of gadgets we use are *departure gadgets* of odd *length* $\ell \geq 1$. The departure gadget is depicted in Figure 4 (left), where the wiggly edges are the edges of its original matching. The gadget consists of an augmenting path $(u_{\text{forth}}, x, w_1, \cdots, w_{\ell+1})$ plus one node u_{back} adjacent to w_1. Its node $w_{\ell+1}$ is called the *root* of the gadget, as it will be used to attach the gadget to other gadgets. We call the two exposed nodes u_{forth} and u_{back} the *forth-node* and the *back-node* of the gadget, respectively, where the forth-node is the farther from $w_{\ell+1}$. The path from w_1 to $w_{\ell+1}$ might be of arbitrary odd length ℓ.

The second type of gadgets we need are *gate gadgets* of even *length* $\ell \geq 2$. This gadget is depicted in Figure 4 (middle). Its nodes w and $v_{\ell+1}$ are the *first-root* and *second-root*, respectively, of the gadget. We call the exposed nodes u_{in} and u_{out} the *in-node* and the *out-node*, respectively. The path with even length ℓ from v_1 to $v_{\ell+1}$ is meant to be alternating. The initial matching of a gate gadget is the one depicted in Figure 4, namely the path from v_1 to $v_{\ell+1}$ has to be alternating with the second-root being covered, and the neighbours of the in- and out-nodes must be covered (while the in- and out-nodes must be exposed).

The last type of gadgets we need is *arrival gadgets* of even *length* $\ell \geq 2$, as depicted in Figure 4 (right). It is a path of even length $\ell + 2$. The *root* of the gadget is $w_{\ell+1}$. The two exposed nodes u_{in} and u_{out} are called the *in-node* and *out-node*, respectively, of the gadget. The path from w_1 to $w_{\ell+1}$ is alternating so that w_1 is covered and $w_{\ell+1}$ is exposed.

We are now ready to describe how to combine these gadgets to form T (and M); see Figures 5 and 6 for illustrations of the structure of T. We start from an edge uv, which we call the *switch edge* of T, and which belongs to M. We then add two special nodes h_v and h_c to T. We connect h_v and u (an end of the switch edge) via an alternating path with even length $\ell_v = \lfloor \frac{9}{10}k \rfloor$, in such a way that h_v is covered by an edge of that path. Now, for every variable x_i of Φ, we identify h_v and

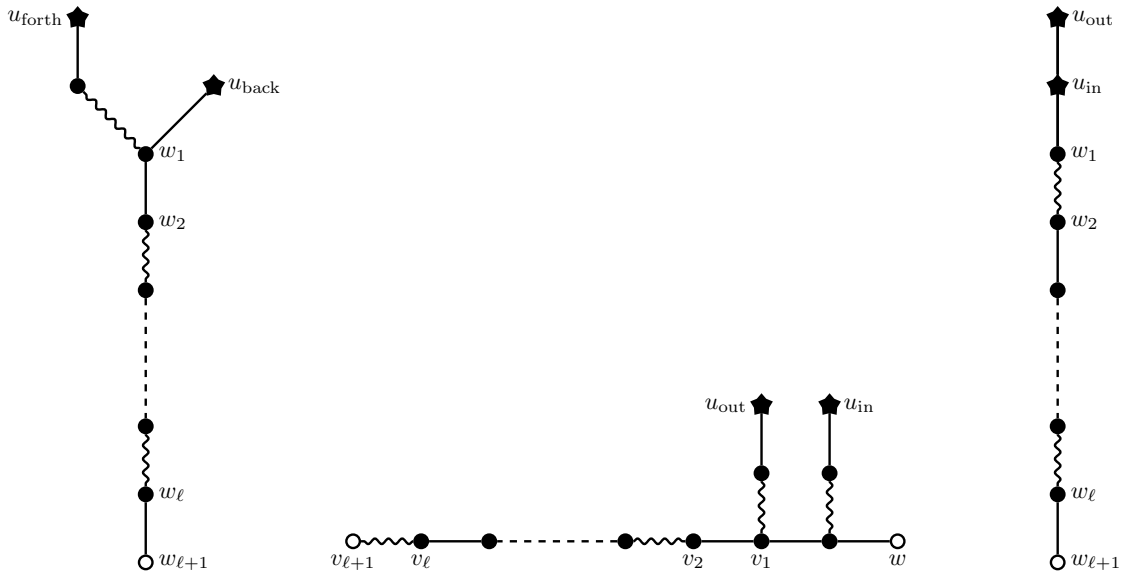

Figure 4: The departure gadget with length ℓ and root $w_{\ell+1}$ (left), the gate gadget with length ℓ, first-root w, and second-root $v_{\ell+1}$ (middle), and the arrival gadget with length ℓ and root $w_{\ell+1}$ (right). Star nodes will be exposed in the reduced graph. White nodes (i.e., root nodes) will be used to attach the gadgets at some nodes. Wiggly edges are edges of the matching.

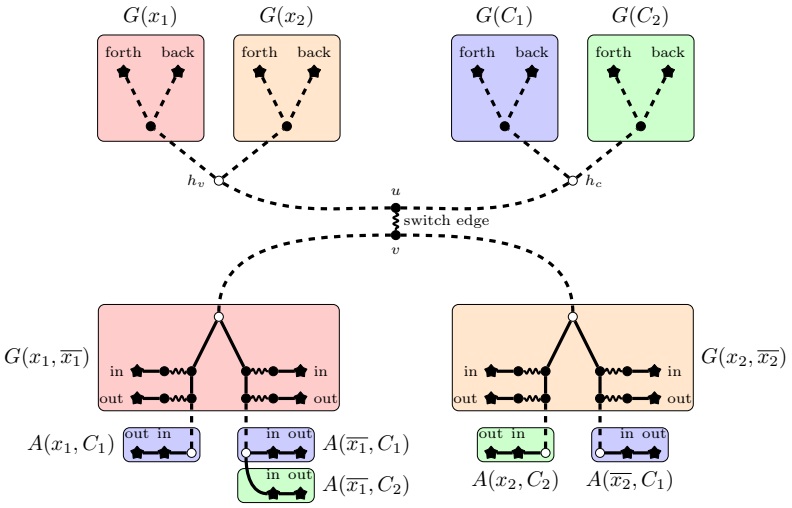

Figure 5: Illustration of the structure of T, as constructed in the proof of Theorem 3.5, for a formula Φ having two variables x_1 and x_2, and two clauses C_1 and C_2 such that $x_1, \overline{x_1}, \overline{x_2} \in C_1$ and $\overline{x_1}, x_2 \in C_2$. Star nodes are exposed nodes. Wiggly edges are edges of the matching. The exposed nodes from an area with a given colour are only joined, via an augmenting ($= k$)-path, to another exposed node of the distinct area with the same colour.

the root of a departure gadget $G(x_i)$ with odd length $\ell_{x_i} = 2i+1$. Note that, for every variable x_i, we have $\ell_{x_i} < \lfloor \frac{1}{10}k \rfloor$ by our choice of k. We do a similar construction for the clauses of Φ. Namely, we connect h_c and u via an alternating path with even length $\ell_c = \lfloor \frac{2}{10}k \rfloor$, so that h_c is covered by an edge of that path (and u is not, since it is already covered by uv). Then, for every clause C_j of Φ, we identify h_c and the root of a departure gadget $G(C_j)$ with odd length $\ell_{C_j} = 2j+1$. Here as well, note that $\ell_{C_j} < \lfloor \frac{1}{10}k \rfloor$ for every clause C_j.

We now consider every variable x_i of Φ in turn. For every such x_i, we add, to T, a gate gadget $S(x_i)$ with even length $\ell'_{x_i} = \lfloor \frac{6}{10}k \rfloor$. Similarly, we add to T a gate gadget $S(\overline{x_i})$ with even length $\ell'_{\overline{x_i}} = \lfloor \frac{6}{10}k \rfloor$. We then identify the first roots of $S(x_i)$ and $S(\overline{x_i})$. The resulting subgraph we have obtained for the couple $\{x_i, \overline{x_i}\}$ is denoted by $G(x_i, \overline{x_i})$, and the node which served for connecting $S(x_i)$ and $S(\overline{x_i})$ is called the *root* of $G(x_i, \overline{x_i})$. We now connect the root of $G(x_i, \overline{x_i})$ and v, the

end of the switch edge we have not used yet, via an alternating path with even length

$$\ell_{x_i,\overline{x_i}} = k - (2 + \ell_{x_i} + \ell_v + 1 + 3) = k - \left(6 + \ell_{x_i} + \left\lfloor \frac{9}{10}k \right\rfloor\right).$$

This ensures that the distance between the forth-node (resp., back-node) of a departure gadget $G(x_i)$ and the in-node (resp., out-node) of the corresponding gate gadget $G(x_i, \overline{x}_i)$ is precisely k.

The initial matching on this path is made in such a way that the root of $G(x_i, \overline{x_i})$ is covered by an edge of that path (and v is not, since it is already covered by uv). Note that $\ell_{x_i,\overline{x_i}} < \lfloor \frac{1}{10}k \rfloor$ by our choice of k.

Now, for every clause C_j of Φ containing x_i, we identify the second-root of $S(x_i)$ and the root of a new arrival gadget $A(x_i, C_j)$ with even length

$$\ell_{x_i,C_j} = k - (2 + \ell_{C_j} + \ell_c + 1 + \ell_{x_i,\overline{x_i}} + 2 + \ell'_{x_i} + 1) = k - \left(6 + \ell_{C_j} + \left\lfloor \frac{8}{10}k \right\rfloor + \ell_{x_i,\overline{x_i}}\right).$$

This ensures that the distance between the forth-node (resp., back-node) of a clause gadget $G(j)$ and the in-node (resp., out-node) of the arrival gadget $A(x_i, C_j)$ is precisely k.

Note that $\ell_{x_i,C_j} < \lfloor \frac{2}{10}k \rfloor$. We repeat the exact same operation with $\overline{x_i}$ and the clauses containing $\overline{x_i}$, resulting in gadgets $S(\overline{x_i})$ with length $\ell'_{\overline{x_i}} = \lfloor \frac{6}{10}k \rfloor$ and $A(\overline{x_i}, C_j)$ with even length

$$\ell_{\overline{x_i},C_j} = k - (2 + \ell_{C_j} + \ell_c + 1 + \ell_{x_i,\overline{x_i}} + 2 + \ell'_{\overline{x_i}} + 1) = k - \left(6 + \ell_{C_j} + \left\lfloor \frac{8}{10}k \right\rfloor + \ell_{x_i,\overline{x_i}}\right),$$

smaller than $\lfloor \frac{2}{10}k \rfloor$.

This ensures that the distance between the forth-node (resp., back-node) of a clause gadget $G(j)$ and the in-node (resp., out-node) of the arrival gadget $A(\overline{x}_i, C_j)$ is precisely k.

Note that the construction of T and M is achieved in polynomial time.

Prior to explain why the reduction is correct, let us point out a few facts. The exposed nodes of T are the following:

- the forth- and back-nodes of all departure gadgets $G(x_i)$;
- the forth- and back-nodes of all departure gadgets $G(C_j)$;
- the in- and out-nodes of all gate gadgets $S(x_i)$ and $S(\overline{x_i})$;
- the in- and out-nodes of all arrival gadgets $A(x_i, C_j)$ and $A(\overline{x_i}, C_j)$.

Among all pairs of exposed nodes, due to our choice of some the gadgets' and paths' lengths, note that only the following ones are at distance exactly k in T (see Figure 6 for an illustration):

1. For every variable x_i of Φ, the forth-node of $G(x_i)$ and the in-nodes of $S(x_i)$ and $S(\overline{x_i})$.
2. For every variable x_i of Φ, the back-node of $G(x_i)$ and the out-nodes of $S(x_i)$ and $S(\overline{x_i})$.
3. For every clause C_j and variable $x_i \in C_j$ (or negated variable $\overline{x_i} \in C_j$) in Φ, the forth-node of $G(C_j)$ and the in-node of $A(x_i, C_j)$ (resp. $A(\overline{x_i}, C_j)$).
4. for every clause C_j and variable $x_i \in C_j$ (or negated variable $\overline{x_i} \in C_j$) in Φ, the back-node of $G(C_j)$ and the out-node of $A(x_i, C_j)$ (resp. $A(\overline{x_i}, C_j)$).

We now explain how M should be $(= k)$-augmented to a matching of size $\mu_{=k}(T, M)$. First of all, it should be noted that, at any point, if a $(= k)$-augmentation is performed from the forth-node of a departure gadget G_1 to the in-node of a gate or arrival gadget G_2, then only the $(= k)$-augmentation from the back-node of G_1 to the out-node of G_2 can be performed. This is because the node u (the end of the switch edge that is the closest to the departure gadgets) is now covered by an incident edge on the path between u and G_1. So, in a sense, $(= k)$-augmentations have to be performed in pairs. Assuming the two $(= k)$-augmentations from $G(x_i)$ to $S(x_i)$ (resp., to $S(\overline{x_i})$) have been performed, we say that the gate gadget $S(x_i)$ (resp. $S(\overline{x_i})$) is *open*.

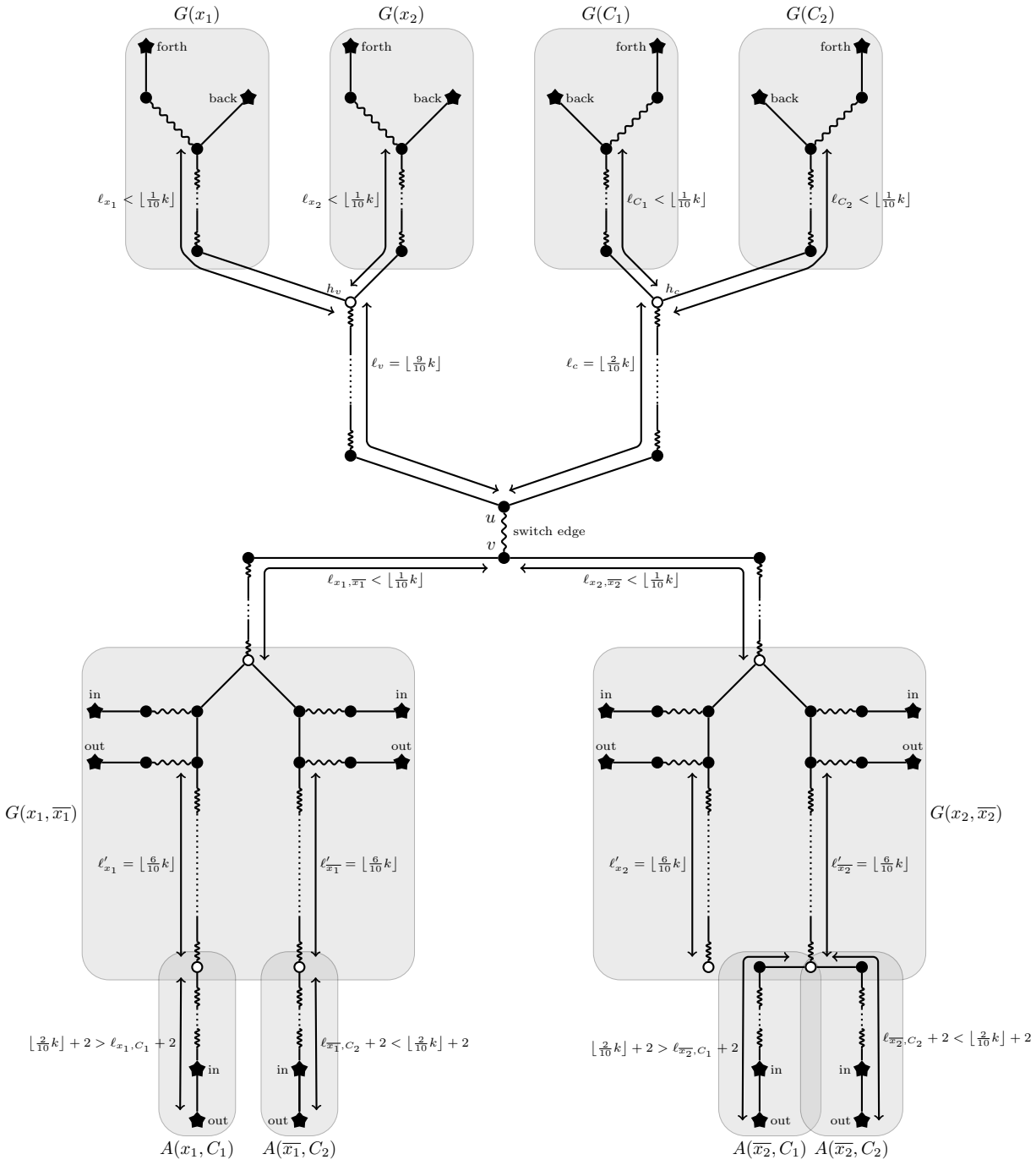

Figure 6: Illustration of the reduction described in the proof of Theorem 3.5, for a formula Φ having two variables x_1 and x_2 and two clauses C_1 and C_2 such that $x_1, \overline{x_2} \in C_1$ and $\overline{x_1}, \overline{x_2} \in C_2$. Star nodes are exposed nodes. Wiggly edges are edges of the matching.

As exposed above, the possible $(= k)$-augmentations go from a departure gadget $G(x_i)$ or $G(C_j)$ towards either a gate gadget (case where the departure gadget is a $G(x_i)$), or an arrival gadget (otherwise). Assuming the clause C_j of Φ contains the variable x_i (resp. negated variable $\overline{x_i}$), we note that augmenting a $(= k)$-path from $G(C_j)$ to $A(x_i, C_j)$ (resp. $A(\overline{x_i}, C_j)$) cannot be done until $S(x_i)$ (resp. $S(\overline{x_i})$) is open. Furthermore, we note that once a gate, say $S(x_i)$, is open, assuming x_i is contained in distinct clauses $C_{j_1}, \cdots, C_{j_{n_i}}$ of Φ, all pairs of $(= k)$-augmentations from the departure gadgets $G(C_{j_1}), \cdots, G(C_{j_{n_i}})$ to the arrival gadgets $A(x_i, C_{j_1}), \cdots, A(x_i, C_{j_{n_i}})$ can be performed in turns.

To sum up, a matching of size $\mu_{=k}(T, M)$ can be obtained, via $(= k)$-augmentations starting from M, as follows:

1. Consider all of the variables $x_1, \cdots, x_n$ of Φ in turn, and, for every considered variable x_i, either perform the two augmentations from $G(x_i)$ to $S(x_i)$ (i.e., open $S(x_i)$), or perform the two augmentations from $G(x_i)$ to $S(\overline{x_i})$ (i.e., open $S(\overline{x_i})$),

2. Then consider all of the clauses $C_1, \cdots, C_m$ of Φ in turn, and, for every considered clause C_j, if C_j contains one variable x_i (resp. negated variable $\overline{x_i}$) such that $S(x_i)$ (resp. $S(\overline{x_i})$) is open, then perform the two augmentations from $G(C_j)$ to $A(x_i, C_j)$ (resp. $A(\overline{x_i}, C_j)$).

So $2n + |M| \leq \mu_{=k}(T, M) \leq 2n + 2m + |M|$, and the upper bound is attained when, for every variable x_i, we can open either $S(x_i)$ or $S(\overline{x_i})$ so that, for every clause gadget C_j, there is an open gate gadget creating a way from the departure gadget $G(C_j)$ towards an arrival gadget $A(x_i, C_j)$ or $A(\overline{x_i}, C_j)$. By considering that opening $S(x_i)$ (resp. $S(\overline{x_i})$) simulates the affectation of value *true* (resp. *false*) to x_i, and that performing a pair of $(= k)$-augmentations from $G(C_j)$ to $A(x_i, C_j)$ (or $A(\overline{x_i}, C_j)$) simulates the fact that x_i (resp. $\overline{x_i}$) brings value *true* to C_j, the equivalence with satisfying Φ follows. □

4. Conclusion

Following the work of Nisse, Salch and Weber [NSW15], we have, in this paper, studied the algorithmic complexity of $MP^{\leq k}$, the problem of augmenting an initial matching as much as possible via $(\leq k)$-augmentations. As this case is far from being well understood, we gave a special focus to the case of trees. On the positive side, we have provided polynomial-time algorithms for solving the problem in bounded-degree trees (assuming k is also fixed), k-sparse trees, and caterpillars.

Seeking for the complexity of $MP^{\leq k}$ in trees, we have introduced a more restricted version of the problem, $MP^{=k}$. On the negative side, we have proved that this problem is indeed NP-complete in trees, when k is part of the input.

One point to raise, is that the polynomial-time algorithm we have proposed for $MP^{\leq k}$ in caterpillars also applies for $MP^{=k}$. To see this is true, one should keep in mind that our algorithm heavily relies on the point that, when given a sequence of augmentations, we can disentangle the augmented paths (i.e., make them disjoint) and get an equivalent sequence. When considering the $MP^{=k}$ problem, such a property is not needed, as it can easily be checked that, for any two augmenting paths with the same length, in a caterpillar, one of the two paths cannot completely include the other.

In the rest of this section, we summarize a number of directions for further work on this topic.

Complexity of $MP^{\leq k}$

Our perspectives for future work are mainly about the complexity of $MP^{\leq k}$ in more classes of trees, towards a full understanding of the problem in that class of graphs. In the current paper, so that the arguments in our proofs work, we needed strong assumptions on the tree's structure, such as long distances between the b-nodes (k-sparse trees), or short branches attached to the b-nodes (caterpillars). The next step would hence be to consider classes of trees without those properties.

Our main interest for focusing on caterpillars in this paper, is that their structure is close to that of paths, for which it exists a polynomial-time solving algorithm for $MP^{\leq k}$. We have the feeling that, from an algorithm solving $MP^{\leq k}$ for a tree T, one could deduce one for any subdivision of T, as the subdivision operation introduces new degree-2 nodes only. Hence, we believe the following could be considered:

Question 4.1. *What is the complexity of $MP^{\leq k}$ when restricted to subdivided caterpillars?*

Remind that, in the case of caterpillars, a key fact is that the maximum degree can be assumed to be at most 3 (due to Corollary 2.19). Obviously, this does not remain true in subdivided caterpillars. Towards Question 4.1, a simpler generalization of our result for caterpillars could

hence be to consider *combs*, i.e., subdivided caterpillars with maximum degree 3. In particular, a quick investigation shows that, already in that class of trees, performing augmentations from left to right might not be an optimal strategy.

Question 4.2. *What is the complexity of $MP^{\leq k}$ when restricted to combs?*

Our result on k-sparse trees stands as a generalization of arguments for solving $MP^{\leq k}$ in subdivided stars. In that light, perhaps another class of trees that could be interesting considering is the one of subdivided bistars (i.e., trees with exactly two b-nodes). Note that, in a subdivided bistar S, if the two b-nodes are at distance more than k, then S is a k-sparse tree, in which case we know how to solve $MP^{\leq k}$. Hence, in order to understand how ($\leq k$)-augmentations should be performed through b-nodes, the following case could be considered:

Question 4.3. *What is the complexity of $MP^{\leq k}$ when restricted to subdivided bistars?*

Complexity of $MP^{=k}$

We believe directions concerning $MP^{=k}$ are also worth considering. In particular, although we proved that $MP^{=}$ is NP-complete in trees, we have no clue about whether this is true for $MP^{=k}$, for some fixed k.

Question 4.4. *Is there an odd $k \geq 3$ such that $MP^{=k}$ is NP-complete when restricted to trees?*

In particular, the case $k = 3$ sounds intriguing:

Question 4.5. *What is the complexity of $MP^{=3}$ when restricted to trees?*

Other classes of graphs

Although $MP^{\leq k}$ and $MP^{=k}$ remain NP-complete when restricted to rather restricted classes of graphs (with maximum degree 3, degeneracy 2, arbitrarily large girth, planarity, bounded treewidth), the polynomial-time solvable cases should concern particular classes of graphs with convenient properties. As such classes, one could consider Θ-graphs (graphs with two vertices joined by arbitrarily many disjoint paths with arbitrary lengths), cacti (graphs made up of cycles connected in a tree-like fashion), or interval graphs. The latter class sounds rather interesting, as perhaps, in that case, augmentations should be performed from left to right, just as in the path and caterpillar cases.

References

[Ber57] C. Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences of the United States of America*, 43(9):842–844, September 1957.

[BGN+17] Julien Bensmail, Valentin Garnero, Nicolas Nisse, Alexandre Salch, and Valentin Weber. Recovery of disrupted airline operations using k-maximum matching in graphs. *Electronic Notes in Discrete Mathematics, proceedings of IX Latin and American Algorithms, Graphs and Optimization (LAGOS'17)*, 62:3–8, 2017.

[BSS09] L. Bui, S. Sanghavi, and R. Srikant. Distributed link scheduling with constant overhead. *IEEE/ACM Trans. Netw.*, 17(5):1467–1480, 2009.

[DP14] R. Duan and S. Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 61(1), 2014.

[Edm65] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[HK73] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[Kuh55] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[MV80] S. Micali and V. V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *21st Symp. on Foundations of Comp. Sc. (FOCS)*, pages 17–27. IEEE, 1980.

[NSW15] N. Nisse, A. Salch, and V. Weber. Recovery of disrupted airline operations, 2015. INRIA-RR-8679, `https://hal.inria.fr/hal-01116487`, to appear in European Journal of Operational Research.

[WS05] X. Wu and R. Srikant. Regulated maximal matching: A distributed scheduling algorithm for multi-hop wireless networks with nodeexclusive spectrum sharing. In *IEEE Conf. on Decision and Control*, 2005.