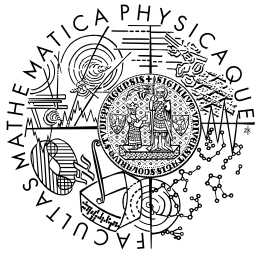

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Jan Soukup

Parity vertex colorings

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the bachelor thesis:	Mgr. Petr Gregor, Ph.D.
Study programme:	Computer Science
Study branch:	General Computer Science

Prague 2018

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

Title: Parity vertex colorings

Author: Jan Soukup

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Petr Gregor, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: A parity path in a vertex colouring of a graph G is a path in which every colour is used even number of times. A parity vertex colouring is a vertex colouring having no parity path. Let $\chi_p(G)$ be the minimal number of colours in a parity vertex colouring of G. It is known that $\chi_p(B_n) \geq \sqrt{n}$ where B_n is the complete binary tree with n layers. We show that the sharp inequality holds. We use this result to obtain a new bound $\chi_p(T) > \sqrt[3]{\log n}$ where T is any binary tree with n vertices.

We study the complexity of computing the parity chromatic number $\chi_p(G)$. We show that checking whether a vertex colouring is a parity vertex colouring is coNP-complete and we design an exponential algorithm to compute it. Then we use Courcelle's theorem to prove the existence of a FPT algorithm checking whether $\chi_p(G) \leq k$ parametrized by k and the treewidth of G. Moreover, we design our own FPT algorithm solving the problem. This algorithm runs in polynomial time whenever k and the treewidth of G is bounded. Finally, we discuss the relation of this colouring to other types of colourings, specifically unique maximum, conflict free, and parity edge colourings.

Keywords: parity vertex colouring, conflict free colouring, unique maximum colouring, binary tree, treewidth, FPT

I would like to thank my supervisor, Mgr. Petr Gregor, Ph.D., for his advice and the support he gave me.

Contents

Introduction

In this thesis we study parity vertex colourings of graphs. A colouring of vertices of a graph G is a *parity vertex colouring* if for every subpath P of G there exists at least one colour used odd number of times on P. A *parity vertex chromatic number* $\chi_p(G)$ is the minimal number of colours used in a parity vertex colouring of G.

This colouring was independently introduced by Cheilaris and Tóth [5] and Borowiecki et al. [3]. Cheilaris and Tóth [5] introduced it as a relaxation of conflict free and unique maximum colourings that is useful for proving lower bounds on the minimal number of colours in these colourings. A *conflict free colouring* (in the literature also known as a conflict free colouring of hypegraphs with respect to paths) is a colouring of vertices of G such that for every path P in G there exists a colour used exactly once on P. The main application of conflict free colourings is in frequency assignment for cellular networks. The conflict free colouring was studied in [6, 5].

This colouring is itself a relaxation of a unique maximum colouring. A *unique maximum colouring* (in the literature also known as a unique maximum colouring of hypergraphs with respect to paths, or alternatively as a vertex ranking) is a colouring of vertices of G with integers such that for every path P in G the maximum colour used on P is used exactly once on P. This colouring has many applications including sparse Cholesky factorization [13] or VLSI layout [14]. Theoretical and algorithmic properties of this colouring were studied in many papers, see e.g. [2, 6, 3].

Borowiecki et al. [3] began the study of the parity vertex colouring inspired by the work on the edge variant of this problem. The study of the parity edge colouring was initiated by Bunde et al. [4] and continued by Hsu and Chang [11]. It was motivated by the fact that this colouring is closely related to the problem of deciding whether a graph embeds in the hypercube and the hypercube is one of the most popular architectures used for parallel computations [12].

Our work

In the first chapter we study theoretical properties of the parity vertex chromatic number. It is easy to see that it is monotone under the subgraph relation. We show that it is not monotone under the minor relation. Cheilaris et al. [6] use the parity vertex chromatic number as a tool to prove lower bounds on the conflict free chromatic number. They proved that for every subdivision B^* of a complete binary tree B_d (i.e. a tree obtained from the complete binary tree by replacing edges with paths) it holds that $\chi_p(B^*) \geq \sqrt{d}$. They did it by proving that every such coloured subdivision contains a certain subgraph that has to use that many colours. We generalize the class of subgraphs that every coloured subdivision must contain to obtain a sharp inequality in the bound. Next, we use this bound to obtain a new bound $\chi_p(T) > \sqrt[3]{\log n}$ where T is an arbitrary binary tree on n vertices.

In the next chapter we study the complexity of computing the parity vertex chromatic number. We start with applying the ideas of Cheilaris and Tóth [5] to

prove that the problem of checking if a given colouring is a parity vertex colouring is coNP-complete. Next, we design an exponential algorithm computing the parity vertex chromatic number that is based on generating all subsets of vertices that lie on a subpath. Applying this algorithm we verified that $\chi_p(Q_5) > 12$ where Q_5 is the hypercube of dimension 5. Then we describe an algorithm checking if a tree has a parity vertex colouring using k colours that is exponential only in k and thus runs in a polynomial time whenever the parameter k is bounded.

In the next chapter we formalize this in terms of parametrized complexity. This theory was established by Downey and Fellows [9]. For more recent results, we refer to [8]. It turns out that many problems that are polynomially solvable on trees are efficiently solvable on a wider class of graphs, namely graphs of bounded treewidth. We state the precise definition later, for now it suffices to say that the treewidth is a parameter of a graph that express how "close" the graph is to a tree. The most interesting theoretical result for graphs of bounded treewidth is Courcelle's theorem [7, 8] stating that every graph property definable in a certain logic can be decided in linear time on graphs of bounded treewidth. We use this theorem to prove that checking if a graph G has parity vertex colouring using k colours can be done in polynomial time whenever k and the treewidth of G are bounded. Since the algorithm guaranteed by Courcelle's theorem is not easily implementable, we consequently design our own algorithm solving the problem based on the standard dynamic programming on a tree decomposition [8].

In the last chapter we discuss the relevance of presented results to other colourings. Specifically, we consider extending our results to conflict free, unique maximum, and parity edge colourings.

Preliminaries

Throughout the paper we consider all graphs to be finite, undirected and simple. We use the standard graph notation. For example $V(G), E(G)$ for the set of vertices and edges of a graph G, respectively, and $G - v$ for a graph obtained from G by deleting vertex v and all of its incident edges. By a colouring we always mean a colouring with integers. Thus the colours are always linearly ordered. We start with defining the main objects of our interest: a parity vertex colouring, a parity path, and a parity vector.

Definition 1. *Let G be a graph, c be a colouring of vertices of G with k colours, V be a subset of vertices of G, and $\mathbb{V}$ be a family of disjoint subsets of vertices of G. A* parity vector *of V (denoted as $pv(V)$) is an element of $\{0,1\}^k$ where the i-th coordinate equals the parity of the number of vertices in V coloured by the i-th colour of c. A* parity vector *of $\mathbb{V}$ (denoted as $pv(\mathbb{V})$) is an element of $\{0,1\}^k$ defined as $pv(\bigcup \mathbb{V})$.*

For a single vertex the parity vector is defined analogously. In addition to standard algebra on parity vectors, we define an operation that adds one parity vector to the set of parity vectors and an operation that combines two sets of parity vectors together.

Definition 2. *Let k be a positive integer, S be a set of parity vectors of dimension k and v be a parity vector of dimension k. We define a commutative operation $S \oplus v$ as follows.*

$$S \oplus v = \bigcup_{s \in S} (s + v)$$

where $+$ is the standard plus operation adding two elements of $\{0,1\}^k$.

Additionally, let P be a set of parity vectors of dimension k. We define a commutative operation $S \oplus P$ as follows.

$$S \oplus P = \bigcup_{s \in S} (s \oplus P)$$

Definition 3. *Let G be a graph, c be a colouring of $V(G)$, and P be a non-empty path in G. The path P is called a* parity path *if $pv(P)$ is the zero parity vector.*

Note that every parity path has odd length (even number of vertices).

Definition 4. *Let G be a graph. A* parity vertex colouring *of G is a colouring of vertices of G such that there exists no parity path in G. The* parity vertex chromatic number *of G (denoted by $\chi_p(G)$) is the minimal number of colours in a parity vertex colouring of G.*

For brevity we sometimes talk about parity vertex colourings just as colourings or as proper colourings when it is clear from the context what type of colouring we mean.

We use the standard notation P_n for the path with n vertices (of length $n-1$), K_n for the complete graph on n vertices, and Q_n for the hypercube of dimension n. We recall definitions of some other types of graphs we use in the thesis.

Definition 5. *A* complete binary tree *is a rooted binary tree in which all interior nodes have two children and all leaves are on the same layer. A complete binary tree with i layers is denoted by B_i.*

Definition 6. *A* binomial tree *of order n (denoted by Bi_n) is a rooted tree defined recursively.*

1. *$Bi_0 = K_1$ with the only vertex as its root.*
2. *For $n > 0$ the binomial tree Bi_n is obtained by joining roots of two disjoint copies of Bi_{n-1} by an edge and then taking the root of the first copy to be the root of Bi_n.*

We use a slightly uncommon definition of separators.

Definition 7. *Let G be a graph, A, B be subsets of vertices of G, and S be a subset of $A \cup B$. We say that S* separates *A and B if the sets $A \setminus S$ and $B \setminus S$ are empty or they are in distinct components of $G - S$.*

Additionally, if $A \cup B = G$, we say that (A, B) is a separation of G with the separator S.

Furthermore, we use log to denote the logarithm of base 2.

1. Bounds on the parity vertex chromatic number

1.1 Properties of the parity chromatic number of paths, cycles, and trees

In this section we start with introducing the parity vector argument, which we use throughout the paper. To illustrate the argument, we first reprove the exact value of the parity chromatic number of paths. It was already proven by Cheilaris et al. [6] and Borowiecki et al. [3].

Lemma 1 ([3, 6])**.** *For every $n \geq 1$, $\chi_p(P_n) = \lfloor \log n \rfloor + 1$.*

Proof. The upper bound is easily seen by induction. We colour the middle vertex (or one of the middle vertices) by a unique colour, and we colour by induction the two remaining paths, which have length at most $\frac{1}{2}n$. This requires at most $\lfloor \log n \rfloor + 1$ colours.

Now we prove the lower bound. Consider any proper parity colouring of P_n with k colours. Let $S_1, \ldots, S_n$ be the subpaths of P_n starting in the first endvertex and having length $1, 2, \ldots, n$, respectively. Suppose that two of them, say S_i, S_j for some $i \geq j$, have the same parity vector, so $pv(S_i) = pv(S_j)$. It follows that the parity vector of the subpath $P_i \setminus P_j$ is the zero vector. Thus there exists a parity path, a contradiction. Therefore the parity vector of every S_i is unique.

There are only 2^k different parity vectors of length k, and no path S_i can have the zero parity vector. Hence $n \leq 2^k - 1$. For positive integers n, k this is equivalent to $\lfloor \log n \rfloor + 1 \leq k$. □

We use a similar argument to find the parity vertex chromatic number of cycles.

Lemma 2. *For every $n \geq 3$, $\chi_p(C_n) = \lceil \log n \rceil + 1$.*

Proof. To see the upper bound, we colour $n - 1$ consecutive vertices on a cycle optimally as on a path, and we colour the remaining vertex with a new colour. This colouring uses exactly $\lfloor \log (n-1) \rfloor + 1 + 1$ colours. By integrality this is equal to $\lceil \log n \rceil + 1$.

To prove the lower bound, consider any proper parity colouring of C_n using k colours. Denote the vertices of the cycle in order of traversal by $v_1, \ldots, v_n$. For every $i \in [n]$, denote the subpath on the first i vertices by S_i. Next, for every $i \in \{1, \ldots, n-1\}$, denote by S_{i+n}, the subpath on vertices $v_{n-i+1}, \ldots, v_n$. Observe that all subpaths S_i are different and that the symmetric difference of any two of them (i.e. we consider only vertices contained in exactly one of them) is also a subpath of the cycle. Thus, by the same argument as in the previous proof, parity vectors of all $2n-1$ paths S_i must be different and non-zero. Hence $2n - 1 \leq 2^k - 1$. Therefore $k \geq \lceil \log n \rceil + 1$. □

The upper bound for paths can also be easily extended to trees.

Lemma 3. *For every $n \geq 1$ and every tree T_n on n vertices, $\chi_p(T_n) \leq \lfloor \log n \rfloor + 1$.*

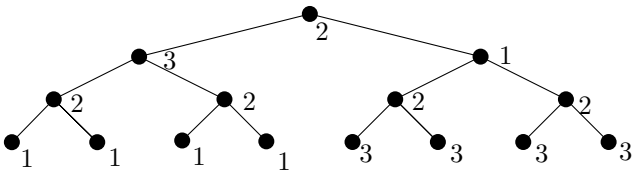

Figure 1.1: A parity vertex colouring of B_4 with three colours.

Proof. We show that a colouring with that many colours exists. We define the colouring inductively. If n is equal to 1, then we colour T_n by one colour. For other n we find a vertex whose deletion will split the graph in parts of size at most $\lfloor \frac{n}{2} \rfloor$. It is well known that such vertex exists. Then we colour these parts by induction, so with at most $\lfloor \log (\lfloor \frac{n}{2} \rfloor) \rfloor + 1$ colours (each with the same set of colours). Since $\lfloor \log (\lfloor \frac{n}{2} \rfloor) \rfloor + 1 = \lfloor \log n \rfloor$, we can use the one remaining colour to colour the splitting vertex. Every path connecting two different parts of the tree, or the splitting vertex, is crossing the splitting vertex. Thus it is not a parity path. Every other paths is entirely in one part of the graph. Thus, by induction, it is not a parity path. □

The parity chromatic number is monotone with respect to subgraphs because the set of all paths in a subgraph is a subset of the set of all paths in the original graph. It is only logical to ask whether the parity chromatic number is also monotone with respect to minors. We give a negative answer to this question by providing a counterexample.

We first determine the parity chromatic number of certain trees that we will need later in the proof. Recall that B_n denote the complete binary tree with n layers.

Lemma 4. $\chi_p(B_4) = 3$.

Proof. The parity chromatic number of B_4 is at least 3 because B_4 contains P_7 as a subgraph. And since there exists a parity vertex colouring with 3 colours, as you can see on Figure 1.1, we have $\chi_p(B_4) = 3$. □

We denote the graph consisting of two copies of B_3 connected by their roots as $T_{3,3}$. We refer to these two rooted subtrees as the first and the second main subtree.

Lemma 5. $\chi_p(T_{3,3}) = 4$

Proof. First of all, there exists a proper colouring with 4 colours, as you can see on Figure 1.2. And since the tree $T_{3,3}$ contains P_6 as a subgraph, it is sufficient to prove that there is no proper parity colouring with 3 colours.

Suppose, for the contradiction, that a proper parity colouring with 3 colours exists. First assume that in both main subtrees there exists a leaf that has different colour than the root of the respective subtree. In both subtrees the vertex between the respective root and the leaf has a different colour than both of them. Thus in both subtrees there exists a path starting in the root and using every colour once. But this contradicts our assumption because the roots are connected. Thus there exists a parity path.

Now assume that one of the main subtrees, without a loss of generality the first one, has the root and all its leaves coloured with one colour. The only possibility to properly complete the parity colouring of this subtree is to colour the two remaining vertices with one of the remaining colours; each with a different one. No matter what colour the root of the second subtree has, there always exists a parity path starting in this root and ending in the first subtree. Therefore our assumption was false and there is no proper colouring using 3 colours. □

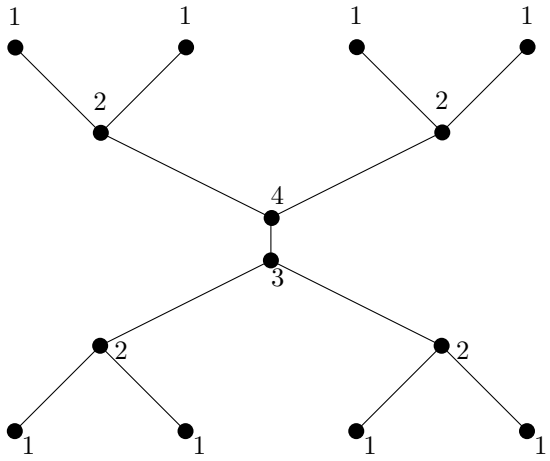

Figure 1.2: A parity vertex colouring of $T_{3,3}$ with four colours.

Theorem 6. *The parity vertex chromatic number is not monotone with respect to minors.*

Proof. Observe that $T_{3,3}$ is a minor of B_4 (it suffices to contract one of the edges incident to the root). The rest immediately follows from Lemmas 4 and 5. □

1.2 Lower bound on the parity chromatic number of subdivisions of complete binary trees

In this subsection we first present a result of Cheilaris et al. [6] about the parity chromatic number of subdivisions of complete binary trees. In the next section we suggest a way to improve it (we actually prove a slightly stronger version of the theorem).

Definition 8. *A graph H is a subdivision of a graph G if H is obtained from G by replacing some edges with paths. We call the original vertices of H the branched vertices.*

Definition 9. *Let T and T' be rooted trees. We say that T' is a compatible subtree of T if T' is a subtree of T and the root of T' is the closest vertex of T' to the root of T.*

We sometimes use the expression a *compatible subgraph* instead of a compatible subtree when we want to emphasize that it is a subgraph (but it will always be a subgraph of some tree).

Theorem 7 ([6]). *For every $n \geq 1$ and every subdivision B^* of B_n, it holds that $\chi_p(B^*) \geq \sqrt{n}$.*

We first prove that a colouring of B^* that has all branched vertices monochromatic cannot use only few colours. Then we show that every colouring of B^* either uses too many colours or contains a large subdivision of B_k as a compatible subgraph with branched vertices coloured monochromatically. This will yield the desired bound.

Lemma 8. *For $n \geq 1$ and for every subdivision B^* of B_n that has all branched vertices coloured with one colour, $\chi_p(B^*) \geq n$.*

Proof. Let Φ be a proper parity colouring of B^* with k colours such that all branched vertices are coloured with one colour. Let M be the set of all subpaths of B^* starting in the root and ending in the branched vertices. There are exactly $2^n - 1$ such subpaths.

Suppose that two distinct paths C_1, C_2 from M have the same parity vector. Let v be the lowest common vertex of C_1, C_2. It exists because they have at least one vertex, the root, in common. The symmetric difference of C_1 and C_2 together with the vertex v composes a subpath in B^*. Denote this subpath by C. Thus $pv(C) = pv(C_1) + pv(C_2) + pv(v) = pv(v)$. Moreover, C is of length at least 1 (it has at least 2 vertices) and its endvertices as well as the vertex v are branched vertices, so they have the same colour. Thus the path obtained from C by deleting one of its endvertices is a parity path, a contradiction. Therefore all of the paths in M have different parity vectors.

Since there are only $2^k - 1$ different non-zero parity vectors, it follows that $2^k - 1 \geq 2^n - 1$. Therefore $k \geq n$. □

Lemma 9 ([6]). *Let $k, n \geq 1$ and B^* be the subdivision of B_n properly coloured with colours $1, \ldots, k$. There exists a vector $(a_1, \ldots, a_k)$ such that $\sum_{i=1}^{k} a_i \geq n$ and for every i, $1 \leq i \leq k$, there exists a compatible subgraph G_i of B^* such that G_i is a subdivision of B_{a_i} and all branched vertices of G_i are coloured with colour i.*

Proof. We proceed by induction on n. For $n = 1$ the tree consists of only one vertex. Let p denote its colour. We set $a_i := 1$, $G_i := B^*$ if $i = p$, and we set $a_i := 0$, $G_i := (\emptyset, \emptyset)$ otherwise. Obviously $\sum_{i=1}^{k} a_i \geq 1$ and every G_i satisfies the required properties.

For $n > 1$ let v be the root of B^* and p its colour. Both of the subtrees of v contain a subdivision of B_{n-1} as a compatible subgraph. Call these subdivisions X, Y, respectively. From the induction hypothesis we get a vector $(x_1, \ldots, x_k)$ with corresponding compatible subgraphs $X_1, \ldots X_k$ of X such that $\sum_{i=1}^{k} x_i \geq n-1$, every graph X_i is a subdivision of B_{x_i}, and all branched vertices of every X_i are coloured with i. Similarly, we get $(y_1, \ldots, y_k)$ and $Y_1, \ldots, Y_k$ for Y. Consider two possibilities:

1. For every i it holds that $x_i = y_i$. In this case we set $a_i := x_i + 1$ for $i = p$ and $a_i := x_i$ otherwise. Clearly, $\sum_{i=1}^{k} a_i \geq n$. It remains to show that there exist corresponding subgraphs G_i.

   If $i \neq p$, we set $G_i := X_i$. By induction hypothesis this is a compatible subgraph of X, thus it is also a compatible subgraph of B^*. The other conditions for G_i follows immediately by induction.

Finally, we set G_p to be the following subtree of B^*. If $x_p = 0$, then let G_p have just one vertex v (recall that v is the root of B^*), and it clearly satisfies all conditions. Otherwise we set v to be the root of G_p. And we connect X_p and Y_p by their roots to the vertex v by a path (in other words we set G_p to be the minimal subtree of B^* containing X_p, Y_p and v). In this way we obtain a compatible subtree of B^* that is also a subdivision of B_{a_p} and has all its branched vertices coloured by p (branched vertices of X_p, Y_p are coloured by p by induction hypothesis and the vertex v has also the colour p).

2. There exists j such that $x_j \neq y_j$. We can assume that $x_j > y_j$. We set $a_i := x_i$, $G_i := X_i$ if $i = j$, and we set $a_i := y_i$, $G_i := Y_i$ otherwise. By induction hypothesis every G_i is a subdivision of B_{a_i} and has its branched vertices coloured by colour i. It is also a compatible subgraph of X or Y, thus it is also a compatible subgraph of B^*. Furthermore, by induction hypothesis $\sum_{i=1}^{k} y_i \geq n - 1$. Hence

$$\sum_{i=1}^{k} a_i = x_j + \sum_{\substack{i=1 \\ i \neq j}}^{k} y_i \geq 1 + \sum_{i=1}^{k} y_i \geq n\,,$$

and all conditions are satisfied.

□

Lemma 10. *Let $k, n \geq 1$ and B^* be the subdivision of B_n. Then every proper colouring of B^* with k colours contains a subdivison of the complete binary tree with $\lceil \frac{n}{k} \rceil$ layers as a compatible subgraph and with all branched vertices coloured with one colour.*

Proof. By Lemma 9 there exists a vector $(a_1, \ldots, a_k)$ such that $\sum_{i=1}^{k} a_i \geq n$ and for every i from $\{1, \ldots, k\}$ there exists a compatible subgraph G_i of B^* such that G_i is a subdivision of B_{a_i} and all branched vertices of G_i are coloured by one colour. By the pigeonhole principle there exists j such that $a_j \geq \lceil \frac{n}{k} \rceil$. Thus G_j is the desired subgraph. □

We are finally ready to prove Theorem 7.

Proof of Theorem 7. Consider a proper parity vertex colouring of B^* with k colours. By Lemma 10 either $k \geq \sqrt{n}$, and we are done, or B^* contains a subdivision of the complete binary tree with at least $\lceil \frac{n}{\sqrt{n}} \rceil$ layers (so at least $\sqrt{n}$ layers) with all branched vertices coloured by one colour as a compatible subgraph. Hence by Lemma 8 we obtain $k \geq \sqrt{n}$ anyway. □

1.3 New sharp bound on the parity chromatic number of subdivisions of complete binary trees

Motivated by Lemma 8, we present a more general class of partially coloured graphs that has a nice lower bound on its parity chromatic number. We use this result to prove that Theorem 7 holds with a sharp inequality.

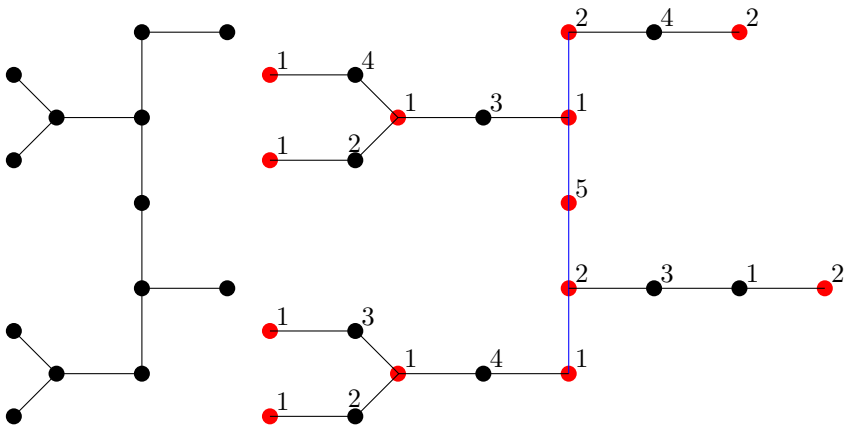

Figure 1.3: An example of a graph from $\mathbb{F}$ (on the left) and a safflower obtained from it (on the right). Edges of the stem of the safflower are blue. Main vertices are red.

Definition 10. *Let $\mathbb{F}$ be the family of all rooted trees consisting of a central branch (a path from the root to some leaf) and at most one complete binary tree attached to each vertex of this branch.*

Let G be a properly coloured subdivision of some graph G' from $\mathbb{F}$. We say that G is a safflower *if G has all its branched vertices of every attached subdivision coloured with the same colour as the vertex of the central branch the subdivision is attached to. Furthermore, we use the following notation.*

1. *A* stem *of G is the path of G that is the subdivision of the central branch of G'.*
2. Main vertices *of G are the vertices of the stem together with all original vertices of G'*
3. $\mathrm{Num}(G)$ *is the number of main vertices in G.*

Example 1.

(a) The tree on Figure 1.3 (on the right) is a safflower.

(b) Every rooted complete binary tree can be viewed as one central branch (we can choose any branch of the tree) with attached complete binary trees to it. Therefore every properly coloured subdivision F of a binary tree such that F has all branched vertices (the original vertices) monochromatic is by definition a safflower. Furthermore, as stated previously, we can choose any branch of the complete binary tree to be a central branch. Thus the stem of the safflower F can be any branch of F.

Note that main vertices of such safflower F are exactly all branched vertices of the subdivision together with all vertices of the stem. Therefore, if we choose the longest branch of F to be the stem then $num(F) = 2^n - 1 - n + k$ where n is the number of layers of the original complete binary tree and k is the length of the longest branch of F.

(c) If we are given a safflower, then if we prolong the stem in the endvertices by attaching a path of new main vertices, we obtain another safflower (if we label the new vertices so that the colouring remains proper). This is indeed a safflower because we can prolong the central branch of the graph the safflower is obtained from by attaching the same path.

Lemma 11. *Every safflower F requires at least* $\log(Num(F)+1)$ *colours.*

Proof. Let M be the set of subpaths of F from the root to some main vertex of F. Clearly $|M| = Num(F)$. If $Num(F) = 1$, the lemma clearly holds. Thus assume $|M| \geq 2$.

Suppose that two distinct paths C_1, C_2 from M have the same parity vector. Let v be the lowest common vertex of C_1, C_2. Then the symetric difference of C_1 and C_2 together with the vertex v compose a subpath (denote it by C) in F. Thus $pv(C) = pv(C_1) + pv(C_2) + pv(v) = pv(v)$. Moreover, one endvertex u of C is either the vertex v or it is a branched vertex of a subdivision connected to v, so it has the same colour as v. Therefore the path obtained from C by deleting u is a parity path (note that it is not empty). Since the colouring of F is proper, every path in M has a distinct parity vector.

Furthermore, none of the paths in M can have the zero parity vector, hence the number of used colours is at least $\log_2(num(F)+1)$. □

Lemma 11 can be used to improve the lower bound on the parity chromatic number of certain trees. The general idea is the same as in Theorem 7. Every colouring of a graph either uses a large number of colours or it contains a large safflower. Thus it also uses many colours. We give an example how to obtain a slightly stronger version of Theorem 7.

Theorem 12. *For every $n \geq 2$ and every subdivision B^* of B_n, $\chi_p(B^*) > \sqrt{n}$.*

Proof. Consider any proper colouring of B^*. Suppose, for a contradiction, that it uses at most $\sqrt{n}$ colours. By the same argument as in Theorem 7, there exists a subdivision of the complete binary tree with at least $\sqrt{n}$ layers and all branched vertices coloured monochromatically as a compatible subgraph of B^*. By Example 1(b), it is also a safflower. By Example 1(c) we can prolong its stem to stretch from the root of B^* to some leaf of B^*. This safflower has at least n levels, hence by Example 1(b) it has at least $2^{\sqrt{n}} - 1 - \sqrt{n} + n$ main vertices. Therefore by Lemma 11 the safflower uses at least $\lceil log(2^{\sqrt{n}} - 1 - \sqrt{n} + n + 1) \rceil$ colours. For $n \geq 2$ this is clearly greater than $\sqrt{n}$, hence we have obtained a contradiction. Thus B^* uses at least $\sqrt{n} + 1$ colours. □

1.4 Lower bound on the parity chromatic number of binary trees

In this section we use the lower bound on the parity chromatic number of subdivisions of the complete binary trees to prove a lower bound on the parity chromatic number of a general binary tree.

Theorem 13. *For every binary tree B on n vertices, $\chi_p(B) > \sqrt[3]{\log n}$.*

We show that every binary tree either contains a long path or it contains a subdivision of a large binary tree. For this purpose, we estimate the maximum number of vertices a binary tree can have when it has a bounded number of layers and does not contain certain subdivisions.

Definition 11. *For integers $l, d \geq 0$ let $A(l, d)$ be the maximal number n such that there exists a binary tree on n vertices with at most l layers and not containing a subdivision of B_{d+1} as a compatible subgraph.*

Claim 14. *For every $l \geq 0$ it holds that $A(l, 0) = 0$.*

Proof. The tree B_1 is a single vertex, so any tree not containing B_1 as a compatible subgraph must be empty. □

Claim 15. *For every $d \geq l \geq 0$ it holds that $A(l, d) = 2^l - 1$.*

Proof. A binary tree with at most l layers can have at most $2^l - 1$ vertices. The complete binary tree with l layers has this number of vertices and does not contain a subdivision of B_{d+1} for $d \geq l$. □

Lemma 16. *For every $l > d > 0$ it holds that $A(l, d) = A(l-1, d-1) + A(l-1, d) + 1$.*

Proof. Let B be a binary tree with at most l layers that does not contain a subdivision of B_{d+1} as a compatible subgraph. Let r be its root. Both child subtrees of r, let us denote them by T_1, T_2, have at most $l - 1$ layers. They cannot both contain a subdivision of B_d as a compatible subgraph. Otherwise it would be possible to connect their roots through r to compose a subdivision of B_{d+1} as a compatible subgraph of B. Thus, without a loss of generality, T_1 and T_2 does not contain a subdivision of B_{d+1} and B_d, respectively, as a compatible subgraph. Hence $|V(T_1)| \leq A(l-1, d)$ and $|V(T_2)| \leq A(l-1, d-1)$. Therefore $A(l, d) \leq A(l-1, d-1) + A(l-1, d) + 1$.

On the other hand let, T_1 and T_2 be trees with respectively $A(l-1, d)$ and $A(l-1, d-1)$ vertices, maximally $l - 1$ layers and not containing respectively B_{d+1} and B_d as compatible subgraphs. When we connect their roots by a new vertex, we obtain a new tree with at most l layers that does not contain B_{d+1} as a compatible subgraph. Hence $A(l, d) \geq A(l-1, d-1) + A(l-1, d) + 1$. □

Lemma 17. *For every integers l and d such that $l > d \geq 0$ it holds that*

$$A(l, d) = \sum_{i=0}^{d} \left[\left(2^i - 1\right) \cdot \binom{l-i-1}{l-d-1} \right] + \binom{l}{d} - 1. \tag{1.1}$$

Proof. We prove it by induction on both l and d. First assume that $d = 0$. In this case we need to prove that $A(l, 0) = \binom{l}{0} - 1$. This is true by Claim 14.

Now assume that $l = d+1$. We need to prove that $A(l, l-1) = \sum_{i=0}^{l-1} (2^i - 1) + l - 1$. We use induction on l to do it. For $l = 2$ we need to prove that $A(2, 1) = 2$. It holds because by Lemma 16 we get $A(2, 1) = A(1, 1) + A(1, 0)$. And that is equal to 2 by Claim 15 and 14. For $l > 2$ we use the Lemma 16, the induction

hypothesis, and Claim 15 to obtain

$$\begin{aligned} A(l,l-1) &= A(l-1,l-1) + A(l-1,l-2) + 1 \\ &= 2^{l-1} - 1 + \sum_{i=0}^{l-2} \left(2^i - 1\right) + (l-1) - 1 + 1 \\ &= \sum_{i=0}^{l-1} \left(2^i - 1\right) + l - 1. \end{aligned}$$

Finally, assume that $l - 1 > d > 0$. By the induction hypothesis, $A(l-1,d)$ and $A(l-1,d-1)$ satisfy (1.1). Therefore by the Lemma 16 we see

$$\begin{aligned} A(l,d) &= A(l-1,d) + A(l-1,d-1) + 1 \\ &= \sum_{i=0}^{d} \left[\left(2^i - 1\right) \cdot \binom{l-i-2}{l-d-2}\right] + \binom{l-1}{d} - 1 \\ &\quad + \sum_{i=0}^{d-1} \left[\left(2^i - 1\right) \cdot \binom{l-i-2}{l-d-1}\right] + \binom{l-1}{d-1} - 1 + 1 \\ &= \sum_{i=0}^{d-1} \left[\left(2^i - 1\right) \cdot \left(\binom{l-i-2}{l-d-1} + \binom{l-i-2}{l-d-2}\right)\right] \\ &\quad + \left(2^d - 1\right) \cdot 1 + \binom{l-1}{d} + \binom{l-1}{d-1} - 1 \\ &= \sum_{i=0}^{d-1} \left[\left(2^i - 1\right) \cdot \binom{l-i-1}{l-d-1}\right] + \left(2^d - 1\right) + \binom{l}{d} - 1 \\ &= \sum_{i=0}^{d} \left[\left(2^i - 1\right) \cdot \binom{l-i-1}{l-d-1}\right] + \binom{l}{d} - 1. \end{aligned}$$

□

The formula provided by Lemma 17 does not have a simple form. Instead, we provide a simple upper bound.

Lemma 18. *For $l \geq d \geq 0$ it holds that $A(l,d) \leq l^d$.*

Proof. In case $l = d$ it easily follows from Claim 15.

Now assume that $l > d$. We apply Lemma 17. In case $d = 0$ we see that $A(l,d) = 0 < l^0$. In case $d = 1$ we get $A(l,d) = l$. In case $d = 2$ we get

$$\begin{aligned} A(l,d) &= \sum_{i=0}^{2} \left[\left(2^i - 1\right) \cdot \binom{l-i-1}{l-2-1}\right] + \binom{l}{2} - 1 \\ &= \left(2^2 - 1\right) + \binom{l-2}{l-3} + \binom{l}{2} - 1 \\ &= (l-2) + \binom{l}{2} + 2 = l + \frac{l^2 - l}{2} < l^2 \end{aligned}$$

where the last inequality holds because $l > d = 2$. In case $d = 3$ we get

$$\begin{aligned} A(l,d) &= \sum_{i=0}^{3} \left[\left(2^i - 1\right) \cdot \binom{l-i-1}{l-3-1} \right] + \binom{l}{3} - 1 \\ &= 7 + 3 \cdot \binom{l-3}{l-4} + \binom{l-2}{l-4} + \binom{l}{3} - 1 \\ &= 3l - 9 + \binom{l-2}{2} + \binom{l}{3} + 6 \\ &\leq \frac{l^3}{3} + \frac{l^3}{2} + \frac{l^3}{6} - 3 < l^3 \end{aligned}$$

where we used $l > d = 3$. In case $d > 3$ we get

$$\begin{aligned} A(l,d) &= \sum_{i=0}^{d} \left[\left(2^i - 1\right) \cdot \binom{l-i-1}{l-d-1} \right] + \binom{l}{d} - 1 \\ &\leq \left(2^d - 1\right) \cdot \sum_{i=0}^{d} \binom{l-i-1}{l-d-1} + \binom{l}{d} - 1 \\ &= \left(2^d - 1\right) \cdot \binom{l}{l-d} + \binom{l}{d} - 1 \\ &= \left(2^d\right) \cdot \binom{l}{d} - 1 \\ &< l^d. \end{aligned}$$

In the last step we used inequality $\left(2^d\right) \cdot \binom{l}{d} \leq l^d$, which clearly holds for $l > d \geq 4$. In the second step we used the following well known Hockey-stick identity.

Proposition 1 (Hockey-stick identity)**.** *For every $n \geq r > 0$ it holds that*

$$\sum_{i=r}^{n} \binom{i}{r} = \binom{n+1}{r+1}.$$

□

Now we are finally ready to prove Theorem 13.

Proof of Theorem 13. Let $n > 2$ and B be an arbitrary binary tree on n vertices and let $b = \chi_p(B)$. Lemma 1 implies that B does not contain a path on 2^b vertices as a subgraph. Thus B has at most 2^b layers. Theorem 12 implies that B does not contain a subdivision of B_{b^2} as a subgraph, thus particularly not as a compatible subgraph. Hence $n \leq A(2^b, b^2 - 1)$. By Lemma 18 we see that $n \leq \left(2^b\right)^{b^2-1} < 2^{b^3}$. Therefore $b > \sqrt[3]{\log n}$. □

1.5 Parity vertex colouring of Q_5

Borowiecki et al. [3] formulated the following conjecture.

Conjecture 1. *It holds that $\chi_p(Q_d) = F_{d+2}$ where F_i is the i-th Fibonacci number.*

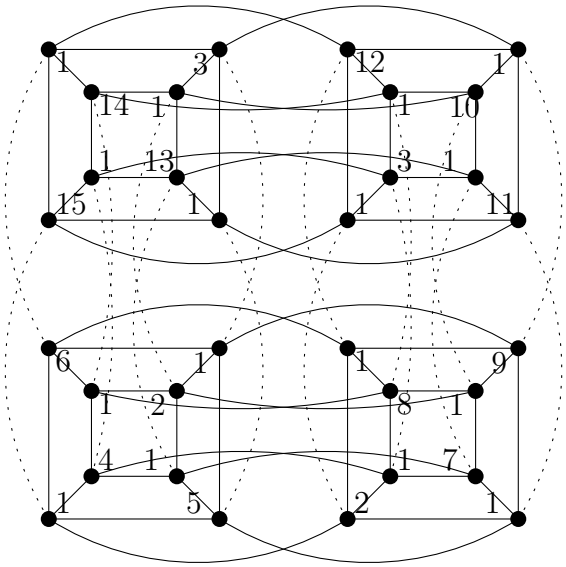

Figure 1.4: A parity vertex colouring of Q_5 with 15 colours.

They provided a proof that it holds for $d < 5$. We tried to prove it for Q_5, but in the end we were not even able to find a colouring with 13 colours. We also believe that it does not exist and that we need at least 15 colours to colour Q_5. We also implemented the algorithm described in Section 2.2 and tried to find a colouring with 13 colours, or prove there is none. Unfortunately, we only manage to show that there is no colouring using 12 colours. For 13 colours the running time would be too long. We at least give the colouring using 15 colours.

Lemma 19. $\chi_p(Q_5) \leq 15$.

Proof. The colouring of Q_5 using 15 colours is given on Figure 1.4. It is not easily seen that it is a proper colouring. Thus we provide a short proof. There are 16 vertices of colour 1, two vertices of colour 2, two vertices of colour 3, and one vertex of each colour between 4 and 15. Recall that every parity path has odd length. Clearly, no two vertices with the same colour are adjacent. Thus there is no parity path of length 1. Additionally, on every path of odd length there are exactly half of the vertices coloured by 1. Thus every path of length 5 or at least 9 contains either colour 1 odd number of times or it contains one of the unique colours (the colours between 4 and 15), and so it is not a parity path. It remains to check the paths on 4 and 8 vertices. A parity path on 4 vertices has to contain either the pair of vertices coloured by 2 or the pair coloured by 3. But the vertices in these pairs are at distance 4 from each other, so it is not possible. A parity path on 8 vertices has to contain both pairs of vertices coloured by 2 and by 3. But the vertices coloured by 2 or 3 in the left part of the hypercube are at distance 4 from both of the vertices coloured by 2 or 3 on the right side. Thus the smallest length of a path connecting all of the vertices coloured by 2 and 3 is 8. Therefore there is no parity path. □

2. Complexity of computing the parity vertex chromatic number

In this chapter we present a straightforward method for computing the parity chromatic number of general graphs. We also design a more efficient method to compute the parity chromatic number of trees.

But at first we show that computing the parity vertex chromatic number of graphs is hard. In particular, we show that even verifying that a colouring of a graph is proper is coNP-complete.

2.1 coNP - completeness

Theorem 20. *Given a graph and its colouring, it is coNP-complete to decide whether the parity vertex colouring is proper.*

The following proof is a slight variation of a proof from Cheilaris and Tóth [5], where the authors proved the same statement about a different colouring (conflict free colouring, i.e. a colouring is proper if on every path exists a unique colour). For the specially coloured graphs used in the proof, these colouring are identical.

Proof. We need to prove that the problem is coNP-hard and that it lies in coNP. In other words we have to prove that this problem is at least as hard as any problem in coNP, and that given an appropriate certificate we can verify in polynomial time that an instance of this problem is not properly coloured. We start with the first part.

We show that the complement of the Hamiltonian path problem can be reduced to our problem. That is, given a graph G we construct in polynomial time a graph G^* with colouring Col of its vertices such that G has no Hamiltonian path if and only if Col is proper.

Let $\{v_1, \ldots, v_n\}$ be the vertex set of G. We define G^* to consist of two isomorphic copies G' and G'' of G with vertex sets $\{v'_1, \ldots, v'_n\}$ and $\{v''_1, \ldots, v''_n\}$, respectively. Additionally, G^* contains for every pair of vertices v'_i, v''_i a path $P_i = (v'_i, v_{i,1}, \ldots, v_{i,i-1}, v_{i,i+1}, \ldots, v_{i,n}, v''_i)$ where $v_{i,1}, \ldots, v_{i,i-1}, v_{i,i+1}, \ldots, v_{i,n}$ are new vertices. We now define the colouring Col. For every $i \in [n]$ we set $Col(v'_i) = Col(v''_i) = i$ and for every $n \geq i > j \geq 1$ we set $Col(v_{i,j}) = Col(v_{j,i}) = (i-1)\cdot n+j$. Observe that every colour is used exactly twice and that inner vertices of every P_i are coloured with distinct colours. Furthermore, every two distinct paths P_i, P_j use the same colour on exactly one pair of inner vertices, namely $v_{i,j}$ and $v_{j,i}$.

Let G contain some Hamiltonian path, say $F = (v_1, v_2, \ldots, v_n)$. It follows that G^* also contains a Hamiltonian path obtained from F by replacing every vertex v_i by the path P_i or its reverse in a way that the consecutive paths can be connected by an edge of G' or G'' (so the end of the first path and the start of the second path lies in the same copy of G). Since every colour in G^* is used exactly twice, it follows that this path is a parity path and so the colouring Col is not proper.

On the other hand, let Col not be a proper parity colouring and let F be a parity path of G^*. We show that G^* and consequently G contains a Hamiltonian

path. Since F is a parity path, every colour is used even number of times on F. Since every colour is used exactly twice in G^*, it follows that every colour is used twice or zero times on F. Recall that inner vertices of every path P_i have different colours. Thus F must contain some vertex of G' or G'', say v'_i. Vertex v''_i is the only other vertex using the same colour as v'_i. Hence F contains both v'_i and v''_i. Therefore it must contain an entire P_j for some j (subgraphs G', G'' are connected only by these paths). Since exactly one colour of every other P_l is used also on P_j, it follows that F contains vertices from all P_l. Therefore F also contains all vertices of both G', G''.

Assume, for now, that F is not Hamiltonian. Observe that if it does not contain all vertices of some P_i, then one of its end vertices must be on that path. Thus F contains all paths P_i, except for at most two exceptions P_k, P_l. Suppose that it does not contain two different vertices of P_k, say $v_{k,i}$, $v_{k,j}$. At least one of indices i, j is different from l, say i. Hence F contains the entire P_i and consequently also $v_{i,k}$. Therefore it must contain also $v_{k,i}$, a contradiction. We can use the same argumentation for P_l, and so it follows that F does not contain at most one vertex of P_k and at most one vertex of P_l. It easily follows that we can extend F by these vertices and obtain a Hamiltonian path.

So there always exists a Hamiltonian path in G^*, say the path F. Observe that in every Hamiltonian path of G^* with end vertices in $G' \cup G''$, paths P_i are part of the Hamiltonian path. So in this case we would obtain a Hamiltonian path in the graph G by contracting paths P_i. We now show that we can modify F to find such Hamiltonian path in G^*. If the end vertices of F are adjacent we obtained a Hamiltonian cycle. Thus we can split it in between some pair of vertices v'_i, v'_j and consequently find a Hamiltonian path in G. Otherwise, let x, y be end vertices of F. The vertex x must be adjacent to some vertex of $G' \cup G''$, say v'_i (otherwise one of its neighbours would not be on the Hamiltonian path). So the Hamiltonian path looks like this $(x, \ldots, v'_j, v'_i, v'_k, \ldots y)$. We can reconnect it to obtain a Hamiltonian path $(v'_j, \ldots, x, v'_i, v'_k, \ldots y)$. In a similar way we can reconnect y and its neighbour to obtain a Hamiltonian path with end vertices in $G' \cup G''$, and consequently a Hamiltonian path in G.

It remains to prove that the problem is in coNP. If the given certificate is a parity path, then we can easily verify in linear time that this path really contains every colour even number of times.

□

2.2 Brute-force approach

We design an algorithm `FindColouring(`G`,`k`)` that for a given graph G and an integer k decides whether there exists a proper parity colouring with k colours. In addition, if a proper colouring exists, the algorithm returns one. We can always assume $k < |V(G)|$, otherwise we could colour every vertex uniquely. We have to tackle two problems.

First, the colouring must be proper. I.e. every path must use some colour odd number of times. We could go through all possible paths, but some paths can have the same set of vertices, and so the condition for them is the same. Thus, we will instead construct a family of all subsets of vertices of G such that for

each subset in this family there exists a subpath in G with exactly the same set of vertices.

Our second problem is the following. If the algorithm outputs NO, we have to be sure that every possible colouring of G is not proper. For this purpose, we define some ordering of vertices and then we recursively try all possible colourings. In particular, in i-th recursion we try all colours of i-th vertex and call the algorithm recursively on remaining vertices. To make it more effective, we observe that even partial colourings of G must satisfy some conditions in order to be admissible for extending into proper parity colourings. So, during the process of colouring we discard bad partial colourings. For that purpose we use the family of subsets of vertices discussed above.

Now we proceed to describing the algorithm in detail and more formally.

Definition 12. *Let G be a graph. A subset S of its vertices is called* path-induced *if there exists a subpath P of G such that $V(P) = S$.*

First, we design an algorithm `Walkable(`G`)` finding all path-induced subsets of a graph G.

We actually first find a set of all pairs (E, S) such that there exists a subpath P in G such that $V(P) = S$ and the set E is exactly the set of endpoints of P. We call these pairs *path pairs*. Its easy to see that for each such path pair, the set S is path-induced. On the other hand, for every path-induced set S there exists a set E such that (E, S) is a path pair.

We proceed inductively on the size of S. In the i-th step we compute a set M_i of all path pairs (E, S) such that $|S| = i$. In the beginning we set M_1 to be the set of all $(\{v\}, \{v\})$ for every vertex v.

To construct the set M_{i+1} we go through all sets (E, S) from M_i. For every v from E and every neigbour u of v that is not in S, we construct a pair (E', S') where $S' := S \cup \{u\}$ and $E' := E \cup \{u\}$ if $|E| = 1$, and $E' := E \setminus \{v\} \cup \{u\}$ otherwise. We add this pair to M_{i+1} unless it is already there.

After computing all M_i's, the set of all path pairs of G is $\bigcup_i M_i$, as we will prove in the following lemma. Let us denote this union by M. Then it is clear that the family of all walkable subsets of $V(G)$ is the set of all S such that there exists a set E with $(E, S) \in M$. So to compute all path-induced subsets of $V(G)$, it suffices to go through all pairs (E, S) from M and accumulate all distinct S.

We will prove that the set M computed by the algorithm is really the set of all path pairs in G. That proves the correctness of the algorithm.

Lemma 21. *The algorithm `Walkable(`G`)` computes the set M of all path pairs of G, and consequently return the set of all path-induced subsets.*

Proof. Suppose that M (the set of path pairs computed by the algorithm) is not the set of all path pairs in G. Recall that we compute M as a union of all M_i where M_i should be a set of all path pairs (E, S) of G such that $|S| = i$. Let M_k be the first such set computed wrongly. From the algorithm it is clear that M_k contains only path pairs belonging to M_k. Thus, there must be some pair missing. Let (E, S) be a path pair missing in M_k. Let $S = \{c_1, \ldots, c_k\}$. And let $C = (c_1, \ldots, c_k)$ be one of the corresponding subpaths of G. If $k = 1$ then $C = (c_1)$ and so $(E, S) = (\{c_1\}, \{c_1\})$, but this pair is contained in M_1, a contradiction. Therefore $k \geq 2$. Since M_k is the first set

computed wrongly and $C' := (c_1, \ldots, c_{k-1})$ is also a subpath of G, it follows that $(E', S') := (\{c_1, c_{k-1}\}, \{c_1, \ldots, c_{k-1}\})$ is in M, specifically it is in M_{k-1}. Since $\{c_{k-1}, c_k\}$ is an edge of G and c_k is not in S', the algorithm added pair (E, S) to M_k during processing the pair (E', S'). That contradicts our assumption and thus the set M is indeed the set of all path pairs of G. Since every path-induced subset is a part of some path pair and every path pair contains a path-induced subset, it follows that the accumulated set returned by the algorithm is indeed the set of all path-induced subsets of G. □

We give a pseudocode of this algorithm. We now analyse the time complexity of the algorithm.

```
1  Algorithm Walkable(G)
       Input  : A graph G
       Output: A family of all path-induced subsets of vertices of G
2      M_1 ← ∅
3      foreach v ∈ V(G) do
4          M_1 ← M_1 ∪ ({v}, {v})
5      for i = 2 to n do
6          M_i ← ∅
7          foreach (E, S) ∈ M_{i−1} do
8              foreach v ∈ E do
9                  foreach u ∈ N(v) do
10                     if u ∉ S then
11                         S' ← S ∪ {u}
12                         if i=1 then
13                             E' ← E ∪ {u}
14                         else
15                             E' ← E \ {v} ∪ {u}
16                         M_i ← M_i ∪ {(E', S')}
17     M ← ⋃_i M_i
18     Res ← ∅
19     foreach (E, S) ∈ M do
20         Res ← Res ∪ {S}
21     return Res
```

Algorithm 1: An algorithm for computing all path-induced subsets of a graph G.

Theorem 22. *The algorithm* `Walkable(G)` *runs in time* $n^{O(1)} \cdot 2^n$ *where* n *is the number of vertices of the input graph* G*.*

Proof. There are at most $n^2 \cdot 2^n$ distinct path pairs in total. The size of each path pair is in $O(n)$. The algorithm gradually compute sets M_i of path pairs. To compute M_i it goes through M_{i-1} and for every path pair there compute in polynomial time some path pairs from M_i. The only operation for each path pair possibly running in a time not polynomial with n is checking if some M_i contains some element, but using space $n^2 \cdot 2^n$, or hashing, it can be done in constant time. Thus the algorithm runs in $n^{O(1)} \cdot 2^n$ time. □

We proceed to design the algorithm `FindColouring(`G`,`k`)`. We already have an algorithm generating all path-induced subsets. Thus in order to check if $\chi_p(G) \leq k$, we could easily try all possible colourings and check if all path-induced subsets are properly coloured. We present a heuristics improving this approach.

Firstly we choose some ordering σ of vertices in which we colour them. Later we discuss some good choices of the ordering.

Next, we arrange walkable subsets to groups labelled by vertices. A group with label v (denote it by $M[v]$) will contain all walkable subsets S of vertices such that $v \in S$ and all vertices of S are before v in the ordering σ.

Finally we colour vertices of G in the order of σ, trying all possible colours for every vertex. When we colour the vertex v, we check whether there exists a path-induced subset in $M[v]$ that is a parity subset. If it is so, the colouring is bad and so we proceed directly to the next colour of v, or alternatively to the previous vertex, if we have already ran out of colours for v. If it is not, it means there is no parity subset amongst subsets from $M[v]$. So we can proceed to the next vertex. When we colour the last vertex v and check that there is no parity subset in $M[v]$, we know that there is no parity subset in $M[u]$ for any vertex u. Thus we obtain a proper colouring.

We can also improve the algorithm a bit when we realize that we can permute colours. So when we use colours $\{1, \ldots, k\}$, we can assume that the colour i is the i-th used colour in the ordering σ. In particular, first vertex will always have the colour 1. The asymptotic complexity remains the same.

We give a simple pseudocode describing this colouring process.

Theorem 23. *The running time of the algorithm* `FindColouring(`G`,`k`)` *is* $n^k \cdot n^{O(1)} \cdot 2^n$ *in the worst case where* n *is the number of vertices of the input graph* G.

Proof. The algorithm constructs walkable subsets in time $n^{O(1)} \cdot 2^n$. To rearrange them into sets M according to latest vertex in σ, it suffices to go through them and always find the latest vertex. We can do it in $n^{O(1)} \cdot 2^n$ time. Then the algorithm colours vertices of G. At worst the colour of the last vertex always spoils the colouring. So, for every possible colouring the algortihm checks parity vectors of almost all path-induced subsets. There are n^k colouring. Checking if a parity vector of a walkable subset uses some colour odd number of times can be easily implemented in linear time with the size of the subset. So the algorithm needs at worst $n^k \cdot n^{O(1)} \cdot 2^n$ time. □

So far we have not defined how the ordering σ should look like. And we do not have any clear answer for that. Intuitively, we want to find out that a colouring is bad as soon as possible, thus a good heuristic seems to be to maximize the number of walkable subsets among the first k vertices in the ordering for small k. For certain classes of graphs, good ordering can speed up the algorithm significantly.

We have implemented this algorithm with several hypercube specific improvements, trying to disprove Conjecture 1 by showing that $\chi_p(Q_5) > 13$. But it was feasible only for at most 12 colours. For 13 colours the running time was to high.

```
1  Algorithm FindColouring(G,k)
       Input  : A graph G, an integer k
       Output: A proper parity vertex colouring of G, or False if no such
               colouring exists
2      W ← Walkable(G)
3      σ ← a permutation of V(G)
4      M ← Sets of W sorted out according to their latest vertex in the
        ordering σ
5      Col ← an empty colouring
6      if FindColouringsubproc(1,0) then
7          return Col
8      else
9          return False
10 FindColouringsubproc(v,maxc)
       Input  : an index v (to ordering σ) of the vertex to be processed, an
               integer maxc of maximal used colour so far
       Output: If proper colouring exists it outputs True and it sets the
               ordering to global variable Col, otherwise it outputs False.
       /* If we have already checked all vertices, the colouring is proper.
          */
11     if v = |V(G)| + 1 then
12         return True
13     Actvertex ← σ[v]
14     for i = 1 to maxc + 1 do
15         if  i=k+1 then                    // cannot use more than k colours
16             break
17         Col[Actvertex] = i ;              // set a colour of the v-th vertex
           /* Check the colouring of all subsets, where Actvertex is the
              last vertex.  We omit the code of Isparitysubset(X) function,
              because it has a straiforward implementation.  It just goes
              through vertices of X and checks that every colour is used
              even number of times                                          */
18         Badcolouring ← False
19         foreach X ∈ M[Actvertex] do
20             if Isparitysubset(X) then
21                 Badcolouring ← True
22         if Badcolouring then
23             continue
           /* Proceed to the next vertex                                    */
24         if FindColouringsubproc(v+1,max(maxc,i)) then
25             return True
26     return False
```

Algorithm 2: An algorithm for deciding whether a graph G has a parity vertex colouring using k colours.

2.3 Algorithms for trees

In this section we design a completely different algorithm for computing the parity chromatic number of trees. Instead of constructing path-induced subsets we use dynamic programming on the tree structure. Before we describe the final algorithm, we first solve an easier problem of deciding whether a given colouring is proper.

2.3.1 Verifying the correctness of a colouring

For a given tree T, an integer k, and a colouring $c : V(G) \to \{1 \dots k\}$ we describe an algorithm checking whether the colouring c is proper.

First we root T in some vertex r. Note that we use notation T_v for the subtree of T rooted in v and containing all descendants of v. We have to check whether there exists a parity path. We proceed by dynamic programming on the tree structure starting from leaves. For every node v we compute the set $M(v)$ of all parity vectors of paths in T_v starting in v. Moreover, in every node v we check whether there is a parity subpath in T_v going through v. In this way we check all subpaths of T during the algorithm. Therefore after running the algorithm we know if the colouring is proper. We now describe these procedures for leaves and inner vertices separately.

1. Let v be a leaf of T. We set $M(v)$ to contain only the parity vector of the colour $c(v)$ because there is only one path in T_v that starts in v: the path consisting of only the vertex v. This path is the only path in T_v, and it is not a parity path.

2. Let v be an inner vertex with sons $u_1, \dots, u_m$. Clearly, every path in T_v starting in v has either length 1 or continues to one of the sons of v. On the hand, for every son u_j of v, a path in T_{u_j} starting in u_j can be extended by v to a path in T_v starting in v. Thus,

$$M(v) = pv(v) \cup \bigcup_{i=1}^{m} \left(M(u_i) \oplus pv(v) \right).$$

   Recall that $M(u_i) \oplus pv(v)$ means the set obtained from $M(u_i)$ by adding the parity vector $pv(v)$ to each of its elements.

   Every path in T_v going through v can be split into three parts; to the vertex v and to two paths P_1, P_2 contained in respectively T_{u_i}, T_{u_j} for $i < j$ and starting in u_i, u_j, respectively. These paths can also be empty. On the other hand, every such parts compose a path in T_v going through v. Therefore we check whether some distinct $M(u_i), M(u_j)$ contain two parity vectors differing only in the colour $c(v)$, or if one of them contains the parity vector of the colour $c(v)$. If so, then there is a parity path. Otherwise there is no parity path in T_v going though v.

With a straightforward implementation of this algorithm we get the following theorem.

Theorem 24. *For a given tree T on n vertices, an integer k, and a colouring $c : V(G) \to \{1 \dots k\}$, there is an algorithm deciding whether the colouring c is a proper parity colouring in time $n^{O(1)} \cdot 2^{O(k)}$.*

Proof. We prove that the algorithm we discussed above has this time complexity. In every node we remember a set of parity vectors of dimension k, thus at most 2^k values. The set operations describe above can easily be implemented in time polynomial with the size of the sets and the size of the tree. Thus it runs in time $n^{O(1)} \cdot 2^{O(k)}$. □

Let us note that with clever implementation of this algorithm we can obtain a running time $O(n \cdot 2^k)$. Furthermore, by Lemma 3, for every tree on n vertices there exists a proper parity colouring with at most $\log n + 1$ colours. Observe that for such k this algorithm runs in polynomial time with respect to the size of the tree. Thus with clever implementation we would obtain a quadratic algorithm.

2.3.2 Computing the parity chromatic number of trees

For a given tree T with n nodes and an integer k we give an algorithm deciding whether there exists a proper parity colouring of T using colours $\{1, \dots, k\}$. By using the above algorithm, we could easily try all possible colourings and check if they are proper. But it would be inefficient. Instead, we use a similar approach as in the previous algorithm; i.e. applying dynamic programming. We start with simplifying the notation.

Definition 13. *For every tree T with a root v and for every colouring c of vertices of T, we call the set of all parity vectors of all paths starting in v in the tree T coloured by c a* parity path set of T corresponding to the colouring c.

Firstly, we root T in an arbitrary vertex r. We proceed inductively on the tree structure. For every node v we compute the family $F(v)$ of all distinct parity path sets corresponding to some proper colouring of T_v. Therefore, $F(r)$ is non-empty if and only if there exists a proper parity colouring of T. Observe that one colouring of T_v unambiguously determines one parity path set of T_v, but one parity path set of T_v can correspond to more colourings of T_v. We show how to compute $F(v)$ for every node.

Let v be a leaf of T. The subtree T_v consists of only one vertex v. We can colour the vertex v with k colours. Thus there are only k different colourings of T_v. It is clear that they are all proper. Therefore $F(v) = \bigcup_{i \in [k]} \{\{e_i\}\}$ where e_i is the parity vector corresponding to the colour i.

Let v be an inner vertex of T and $u_1, \dots, u_m$ be its sons. For each son u_i, we have already computed the family $F(u_i)$ of all parity path sets corresponding to some proper colouring of T_{u_i}. Observe that the subtrees T_{u_i} are disjoint, and so their colourings are also disjoint. Thus every set of representatives from families $F(u_i)$ together with the colour of v unambiguously determines one parity path set of T_v (every corresponding colouring can be obtained by combining colourings corresponding to the representatives of each $F(u_i)$ and the colour of v). Therefore for every colour of v, we could go through all possible sets of representatives of families $F(u_i)$ and compute the corresponding parity path set of T_v. We could check if it corresponds to a proper colouring, and possibly add it to $F(v)$. But

every vertex can have many sons. Thus it would be inefficient. Instead, we build the family $F(v)$ step by step, always taking into account more and more sons of v. More precisely, for every colour c we inductively compute families $F_1(v,c),\ldots,F_m(v,c)$ where $F_i(v,c)$ is the family of all distinct parity path sets of the rooted subtree $T_v^i := T_v - \bigcup\{V(T_{u_{i+1}}),\ldots,V(T_{u_m})\}$ corresponding to some proper colourings of this subtree that use the colour c on the vertex v. By definition, $F(v) = \bigcup_{c\in[k]} F_m(v,c)$.

We first show that,

$$F(v,c)_1 = \bigcup_{\substack{S\in F(u_1)\\ pv(v)\notin S}} \left[\{pv(v)\} \cup (S \oplus pv(v))\right].$$

The subtree T_v^1 consists of the root v with only one son u_1 (and the whole T_{u_1}). For every parity path set of T_v^1 corresponding to some proper parity colouring, there exists a parity path set of T_{u_1} corresponding to the same proper colouring (but without the vertex v). Therefore every parity path set from $F_1(v,c)$ is an extension of some parity path set from $F(u_1)$ (the extension depends solely on the colour c of v). On the other hand, given a parity path set S from $F(u_1)$ together with the colour of v, we can easily compute the parity path set P of T_v^1 corresponding to some colouring, simply by adding the parity vector of the colour c to every parity vector of S as well as to the zero parity vector (to add a vector of the path consisting of only v). The corresponding colouring is proper on T_{u_1} and uses the colour c on v, thus it is proper on T_v^1 if and only if paths starting in v are not parity paths. Hence the colouring is proper if and only if P does not contain the zero parity vector or equivalently if S does not contain $pv(v)$. Therefore the formula is correct.

Now, we show how to compute $F_i(v,c)$ for every colour c and an integer $i \geq 2$, under the assumption that we have already computed $F_{i-1}(v,c)$. In particular, we show that

$$F(v,c)_i = \bigcup_{\substack{S_2\in F(u_i)\\ S_1\in F_{i-1}(v,c)\\ S_1\cap S_2=\emptyset}} \left[(S_2 \oplus pv(v)) \cup S_1\right].$$

Observe that T_v^{i-1} and T_{u_i} are disjoint and together compose T_v^i. Recall that parity path sets of T_v^i contain parity vectors of paths of this tree starting in v. These paths are either entirely in T_v^{i-1} or, when we delete their first vertex, they are entirely in T_{u_i} and start in u_i. Therefore every parity path set of T_v^i can be computed as a union of some parity path set of T_v^{i-1} and the parity path set obtained from some parity path set of T_{u_i} by adding the parity vector of the colour c to all of its vectors. On the other hand, each such union yields parity path set of T_v^i. Therefore it remains to decide what pairs of parity path sets of T_v^{i-1} and T_{u_i} can be combined together so that the ensuing parity path set corresponds to some proper colouring.

Let S_1 be a parity path set of T_v^{i-1} corresponding to some proper colouring of T_v^{i-1}, and S_2 be a parity path set of T_{u_i} corresponding to some proper colouring of T_{u_i}. And let S be the parity path set of T_v^i composed from these two, so corresponding to the combined colouring. Since the partial colourings on T_v^{i-1} and T_{u_i} are proper, there are no parity paths in these subgraphs. So only the

paths that contain the edge (v, u_i) can be parity paths. But these paths can be split in an edge (v, u_i) into two paths. One in T_v^{i-1} starting in v and one in T_{u_i} starting in u_i. And conversely, every such pair of paths compose a path in T_v^i going through (v, u_i). These pairs of paths are exactly the paths having their respective parity vectors in S_1 and S_2 respectively. Thus there exists a parity path in S if and only if there exists some parity vector contained in both S_1 and S_2. Therefore the formula is correct.

Theorem 25. *For a given tree T on n vertices and an integer k there is an algorithm deciding whether there exists proper parity colouring of T using k colours in time $n \cdot 2^{O(2^k)}$.*

Proof. We prove that the above algorithm, which solves this problem, has the desired time complexity. In every node v we remember the family $F(v)$ of elements from the power set of parity vectors of dimension k. There are at most 2^{2^k} distinct elements. In order to construct the set $F(v)$ for an inner node v with sons $u_1, \ldots, u_m$, we first need to construct the sets $F_i(v, c)$. The complexity of operations with these sets depends on the representation of these sets, for our purposes it suffices that all of them run in a polynomial time with the size of these sets. Similarly it is easy to see that the formulas given above for computing these sets can be implemented in polynomial time with the size of these sets.

We can charge the son u_i for constructing sets $F_i(v, c)$ for every c. In this way we charge every son for constructing k sets. Since every vertex is a son of at most one parent, we construct $k \cdot n$ such sets in total. Therefore the total time complexity of the algorithm is in $n \cdot 2^{O(2^k)}$. □

3. Algorithms for graphs with bounded treewidth

In the previous chapter we have designed several algorithms, we estimated their runtime, and we claimed that they are in some sense good. We now formalize this in terms of the parametrized complexity. We use results and definitions of Cygan et al. [8] and Courcelle and Engelfriet [7].

The main result of this chapter is that the problem of computing the parity chromatic number of graphs is fixed parameter tractable with respect to the treewidth of the graph together with the number of colours. We give two proofs. The first by using Courcelle's theorem, the second by designing a fixed-parameter algorithm solving the problem.

3.1 Introduction to parametrized complexity

Let us first give a name to the main problem we are dealing with.

Definition 14. *We call the problem of deciding whether the graph G has a parity vertex colouring with k colours* ODDCOLOURING(G,k).

In the problem *ODDCOLOURING(G,k)* the input is not only the graph but also a parameter, the integer k. We give a rigorous definition what a parametrized problem is.

Definition 15. *A* parametrized problem *is a language $L \subseteq \Sigma^* \times \mathbb{N}^d$ where Σ is a fixed, finite alphabet and d is a positive integer. For an instance $(x, k) \in \Sigma^* \times \mathbb{N}^d$, elements of k are called parameters.*

For example ODDCOLOURING(G,k) is a parametrized problem. The graph G can be easily encoded in some alphabet Σ and k is the only parameter.

What is important is that the complexity of algorithms solving these problems is sometimes not dependent only on the size of the input, but also on the parameters. For example, for a bounded size of parameters these algorithms can run in polynomial time. To capture this property we use the following terminology.

Definition 16. *A parametrized problem $L \subseteq \Sigma^* \times \mathbb{N}^d$ is called* fixed-parameter tractable *(FPT) if there exists an algorithm A (called a fixed-parameter algorithm), a computable function*[1] *$f : \mathbb{N}^d \to \mathbb{N}$, and a constant c such that given $(x, k) \in \Sigma^* \times \mathbb{N}^d$ the algorithm A correctly decides whether $(x, k) \in L$ in time bounded by $f(k) \cdot |(x, k)|^c$. The complexity class containing all fixed-parameter tractable problems is called FPT.*

For example, by Theorem 25, ODDCOLOURING(G,k) is solvable in time $n \cdot 2^{O(2^k)}$ for trees. Thus the problem ODDCOLOURING(G,k) for trees is FPT with respect to k. Therefore if k is constant, the problem is solvable in polynomial time.

[1] Every function we are using is computable. Informally, computable functions are functions that can be calculated by Turing machines.

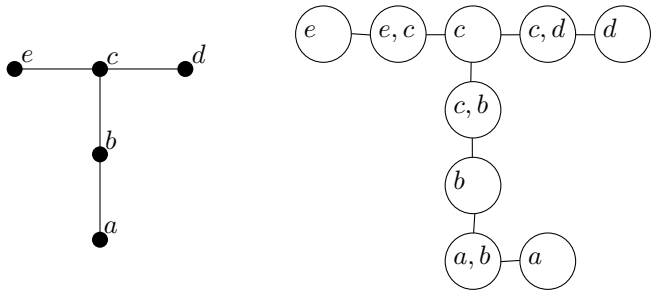

Figure 3.1: A tree with its tree decomposition.

We now proceed to defining a treewidth, the parameter of our main interest. For that we first need to define tree decompositions of graphs.

Definition 17. *A* tree decomposition *of a graph G is a pair $\Gamma = (T, \{X_t\}_{t \in V(T)})$ where T is a tree whose every vertex t has assigned a so called bag X_t such that $X_t \subseteq V(G)$ and the following conditions hold:*

1. *Every vertex of G is in at least one bag. In other words $\bigcup_{t \in V(T)} X_t = V_G$.*
2. *For every edge $\{u, v\}$ of G, there exists a bag X_t, $t \in V(T)$, containing both u and v.*
3. *For every vertex v of G, the set of nodes $\{t; v \in X_t\}$ induces a subtree in T.*

To avoid ambiguity we refer to the vertices of the graph G as vertices and to the vertices of the tree T of a decomposition as nodes.

Definition 18. *The* width *of a tree decomposition $\Gamma = (T, \{X_t\}_{t \in V(T)})$ is defined as $\max_{t \in V(T)} |X_t| - 1$.*

Definition 19. *The* treewidth *of a graph G, denoted by $tw(G)$, is the smallest possible width of a tree decomposition of G.*

For example, the treewidth of a tree (with at least one edge) is 1 because there exists a tree decomposition with every bag of size at most two. For every vertex we create a node with a bag containing just this vertex and for every edge we create a node with a bag containing the two vertices of this edge. Then, we connect all pairs of nodes corresponding to an incident pair, a vertex and an edge, to obtain a tree. See Figure 3.1 for an example of this construction.

As another example the treewidth of a clique with n vertices is $n-1$. Clearly, a node with a bag containing all vertices is a correct tree decomposition. It can be proven that every tree decomposition of a clique must contain a node with a bag containing all vertices. For the proof and other examples we refer to Cygan et al. [8].

Working directly with a general tree decomposition is usually not convenient. We instead work with the following special tree decomposition.

Definition 20. *A* nice tree decomposition *of a graph G is a tree decomposition $\Gamma = (T, \{X_t\}_{t \in V(T)})$ where the tree T is binary and rooted, and for every leaf and root t, the bag X_t is empty. Furthermore, every non-leaf node t must be one of the following types:*

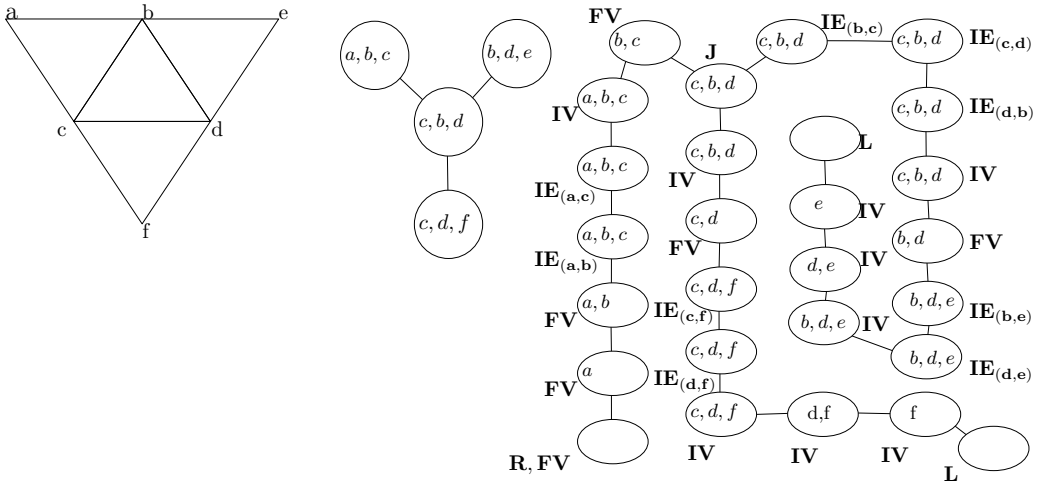

Figure 3.2: A graph with its tree decomposition and its nice tree decomposition (the type of node is denoted by **L** for leaves, **R** for root, **IE** for introduced edge nodes, **FV** for forget vertex nodes, **IV** for introduce vertex nodes, and **J** for join nodes).

1 ***Forget node***: *t has exactly one son t', and there exists $v \in X_{t'}$ such that $X_t = X_{t'} \setminus \{v\}$. We say that v is* forgotten *in t.*

2 ***Introduce vertex node***: *t has exactly one son t', and there exists $v \in V(G)$ such that $X_t = X_{t'} \cup \{v\}$ and $v \notin X_{t'}$. We say that v is* introduced *in t.*

3 ***Join node***: *t has exactly two sons t_1, t_2, and $X_t = X_{t_1} = X_{t_2}$.*

4 ***Introduce edge node***: *t is labelled by an edge $\{u, v\}$, it has exactly one son t', $X_t = X_{t'}$, and $u, v \in X_t$.*

Additionally, for every edge $e = \{u, v\}$, there is exactly one introduce edge node t labelled with the edge e (we say that the edge e is introduced *in t)*

For an example of a graph with its tree decomposition and its nice tree decomposition see Figure 3.2.

It is important that we do not loose generality by using this special version of the decomposition because it can be constructed in polynomial time from a general decomposition, according to the following lemma.

Lemma 26 ([8])**.** *If a graph G admits a tree decomposition of width at most k, then it also admits a nice tree decomposition of width at most k. Moreover, given a tree decomposition $\Gamma = (T, \{X_t\}_{t \in V(T)})$ of G of width at most k, one can in time $O(k^2 \cdot \max(|V(T)|, |V(G)|))$ compute a nice tree decomposition of G of width at most k that has at most $O(k \cdot |V(G)|)$ nodes.*

For every node t of a nice tree decomposition let V_t, E_t be the sets of vertices and edges, respectively, introduced in the subtree T_t (subtree of T rooted in t and containing all descendants of t). And let G_t be the graph (V_t, E_t).

The most important property of a tree decomposition of a graph G is that for every pair of adjacent nodes, the intersection of their bags of vertices is a separator in the graph G. More formally, the following lemma holds.

Lemma 27 ([8]). *Let $(T, \{X_t\}_{t \in V(T)})$ be a tree decomposition of a graph G and let (a, b) be an edge of T. The forest $T - (a, b)$ obtained from T by deleting the edge (a, b) consists of two connected components T_a (containing a) and T_b (containing b). Let $A = \bigcup_{t \in V(T_a)} X_t$ and $B = \bigcup_{t \in V(T_a)} X_t$. Then (A, B) is a separation of G with a separator $X_a \cap X_b$.*

We will usually use it in the following form.

Corollary 28. *Let $(T, \{X_t\}_{t \in V(T)})$ be a tree decomposition of a graph G and let s be a node of T, and let the forest $T - s$ consist of connected components $T_1, \ldots, T_m$. Let $A_i = X_s \cup \bigcup_{t \in V(T_i)} X_t$ for all $i \in [m]$. Then for each distinct $i, j \in [m]$ the vertices of A_i and A_j are separated by X_s in G (and also in all subgraphs of G containing them).*

Proof. Let T_i and T_j be two distinct components of $T - s$. Let a, b be the nodes of T_i and T_j, respectively, adjacent to s. Let $A'_j = \bigcup_{t \in V(T_j)} X_t$. Thus $A_j = X_s \cup A'_j$. By Lemma 27, the vertices of A_i and A'_j are separated by $X_s \cap X_b$ in G. Thus they are separated also by X_s. And since $A_j \setminus A'_j \subseteq X_s$, it follows that A_i and A_j are also separated by X_s in G (and thus also in all subgraphs of G containing them), which concludes the proof. □

We are interested in graphs with bounded treewidth. We need the structure of a tree decomposition. Therefore it is essential that we can construct this decomposition in time "good enough". Luckily, according to the following theorem, this is possible.

Theorem 29 ([1]). *There exists an algorithm that, given an n-vertex graph G and an integer k, runs in time $k^{O(k^3)} \cdot n$ and either constructs a tree decomposition of G of width at most k, or concludes that $tw(G) > k$.*

We now state the main theorem of this chapter. We prove it twice. For the first time in section 3.2 by Courcelle's Theorem (without explicitly describing the FPT algorithm). For the second time in section 3.3 by designing our own FPT algorithm.

Theorem 30. *The problem ODDCOLOURING(G,k) is fixed-parameter tractable with respect to k together with the treewidth of G.*

In particular, for graphs with bounded treewidth this problem is FPT only with respect to the number of colours.

3.2 Courcelle's theorem

In this section we prove Theorem 30 by using Courcelle's Theorem. Before we start, we need to describe a special logic used by the theorem and represent a graph using a logical structure. We assume that the reader has an elementary knowledge of logic.

For every graph G let $\lfloor G \rfloor$ be the structure $\langle V(G) \cup E(G), inc_G \rangle$ where the domain contains both vertices $V(G)$ and edges $E(G)$ of the graph and inc_G is the binary incidence relation, $inc_G \subseteq E \times V$, saying which vertices belongs to which edges. Recall that we are dealing with simple undirected graphs. Thus the structure $\lfloor G \rfloor$ completely defines the graph G.

Properties of a graph can be expressed by sentences (formulas without free variables) of relevant logical languages. A property expressed by a sentence φ holds for a graph G if $\lfloor G \rfloor \models \varphi$ ($\lfloor G \rfloor$ is a model of φ).

For example,

$$\lfloor G \rfloor \models \forall_{x \in V}(\exists_{e \in E}(inc(e, x)))$$

if and only if the degree of every vertex in G is at least one.

A *monadic second order logic* (MSO_2[2]) of graphs is a second order logic that uses only variables representing vertices, edges, set of edges, and set of vertices. It contains the predicate inc for testing edge-vertex incidence, the predicate $=$ for equality testing, and the predicate $\in$ for membership testing. Additionally, it contains constants V and E interpreted as the set of all vertices and edges, respectively. Interpretation of a formula for a given graph G is then naturally defined by its logical structure $\lfloor G \rfloor$.

To distinguish between different types of variables, we use lower-case letters for vertices and edges and upper-case letters for sets.

It is easy to see that the previous formula for testing if a graph has no isolated vertices is a formula of MSO_2. For brevity, we additionally use predicates $adj(u, v)$ for vertex-vertex adjacency testing and $A \subseteq B$ for subset testing (for both vertex and edge subsets). They can be easily rewritten as $u \neq v \wedge \exists_{e \in E}(inc(e, u) \wedge inc(e, v))$ and as $\forall_{x \in V}(x \in A \implies x \in B)$, respectively.

We give one more example to familiarize ourselves with this logic.

$$Hamil = \exists_{C \subseteq E}[Conn(C) \wedge \forall_{x \in V}\ Deg2(x, C)]$$

where the formulas $Conn(C)$ and $Deg2(x, C)$ are written as follows:

$$\begin{aligned} Conn(C) = \forall_{A \subseteq V}[(\exists_{u \in V}\ u \in A \wedge \exists_{v \in V}\ v \notin A) \implies \\ \implies (\exists_{e \in C}\ \exists_{u,v \in V}(inc(e, u) \wedge inc(e, v) \wedge u \in A \wedge v \notin A))] \\ Deg2(x, C) = \exists_{e_1, e_2 \in C}\ [inc(e_1, x) \wedge inc(e_2, x) \wedge e_1 \neq e_2 \wedge \\ \forall_{e \in C}((e \neq e_1 \wedge e \neq e_2) \implies \neg inc(e, x))]. \end{aligned}$$

Rewritten in words, the edge set C connects all vertices if for all non-empty proper subsets A of V, there exists an edge in C connecting a vertex from A with a vertex not from A.

The second formula can be read as follows. The vertex x has degree 2 in the edge set C if exactly two distinct edges from C are incident to x.

Therefore $\lfloor G \rfloor$ models $Hamil$ if and only if G has a Hamiltonian cycle because a Hamiltonian cycle consists of exactly a set of edges connecting all vertices so

[2]Monadic because it uses only variables for vertices, edges and their sets. Subscripted by 2 because it allows edge variables and edge set variables.

that all vertices are incident to exactly two edges from this set.

In MSO_2 it is not possible to describe the cardinality of sets. When we deal with ODDCOLOURING(G, k) we most likely have to describe the parity of some sets. Therefore we use an extension of MSO_2 called the *counting monadic second order logic* (CMSO_2). This logic uses one additional predicate, $Card_p(X)$, checking if the cardinality of X is a multiple of p. In particular, for $p = 2$ we get the predicate $Even(X)$ checking if the cardinality of X is even.

We are ready to state the Courcelle's theorem, in a variant for the logic CMSO_2.

Theorem 31 (Courcelle's theorem [7])**.** *Let φ be a $CMSO_2$ sentence and G be a graph given with its tree decomposition. There exists an algorithm deciding whether $\lfloor G \rfloor \models \varphi$ in time $f(|\varphi|, t) \cdot n$ where t is the treewidth of G and n is the size of G and f is some computable function.*

In order to prove Theorem 30, we construct a sentence $ParityColourable_k$ for a given k in CMSO_2 deciding if a graph has a parity colouring using k colours, and then apply Theorem 31.

$$\begin{aligned} ParityColorable_k = \exists_{X_1,X_2,\ldots,X_k \subseteq V} [&Partition(X_1, \ldots, X_k) \wedge \\ &\forall_{Y \subseteq V} Path(Y) \implies (Oddtimes(X_1, Y) \vee \\ &Oddtimes(X_2, Y) \vee \cdots \vee Oddtimes(X_k, Y))] \end{aligned}$$

where $Path$, $Partition$ and $Oddtimes$ are auxiliary subformulas given below. Rewritten in words, there exists a proper colouring with k colours if and only if we can partition vertices into k sets according to their colours and for every path in G at least one of the partition sets has odd number of vertices in common with that path. The auxiliary subformulas are defined bellow.

$$\begin{aligned} Partition(X_1, \ldots, X_k) = \forall_{x \in V} [&(v \in X_1 \vee \cdots \vee v \in X_k) \wedge \\ &(v \notin X_1 \vee v \notin X_2) \wedge \cdots \wedge (v \notin X_1 \vee v \notin X_k) \wedge \\ &(v \notin X_2 \vee v \notin X_3) \wedge \cdots \wedge (v \notin X_2 \vee v \notin X_k) \wedge \\ &\ldots \\ &(v \notin X_{k-1} \vee v \notin X_k)] \end{aligned}$$

$$Path(X) = \exists_{Y \subseteq E}\ \exists_{x_1,x_2 \in X}[Conn(X,Y) \wedge Deg1(x_1,Y) \wedge$$
$$Deg1(x_2,Y) \wedge \forall_{x \in X}((x \neq x_1 \wedge x \neq x_2) \implies Deg2(x,Y))]$$

$$Conn(X,Y) = \forall_{A \subseteq X}[(\exists_{u \in X}\ u \in A \wedge \exists_{v \in X}\ v \notin A) \implies$$
$$\implies (\exists_{e \in Y}\ \exists_{u,v \in X}(inc(e,u) \wedge inc(e,v) \wedge u \in A \wedge v \notin A))]$$

$$Deg1(x,Y) = \exists_{e_1 \in Y}\ [inc(e_1,x) \wedge \forall_{e \in Y}(e \neq e_1 \implies \neg inc(e,x))]$$

$$Deg2(x,Y) = \exists_{e_1,e_2 \in Y}\ [inc(e_1,x) \wedge inc(e_2,x) \wedge e_1 \neq e_2 \wedge$$
$$\forall_{e \in Y}((e \neq e_1 \wedge e \neq e_2) \implies \neg inc(e,x))]$$

$$Oddtimes(X,Y) = \exists_{A \subseteq X}[\neg Even(A) \wedge$$
$$\forall_{x \in X}\ ((x \in A \implies x \in Y) \wedge (x \notin A \implies x \notin Y))]$$

The formula $Partition(X_1, \ldots, X_k)$ expresses that the variables $X_1, \ldots, X_k$ form a partition of V. It does so by ensuring that every vertex is in some partition set, and no vertex is in two partition sets.

The formula $Path(X)$ expresses that vertices of X form a path of at least 2 vertices. In particular, there must exist a subset Y of edges such that one or two vertices in X are vertices of exactly one edge from Y. The rest of the vertices must be vertices of exactly two edges from Y. Moreover, to really ensure that it is a path, X and Y must compose a connected graph. These properties are expresses by subformulas $Conn(X,Y), Deg1(x,Y), Deg2(x,Y)$ that are all similar to the ones we used in the example describing the Hamiltonian cycle.

The formula $Oddtimes(X,Y)$ checks if the intersection of X and Y has an odd size. It does so by using the predicate $Even$.

Lemma 32. *The size of the formula $ParityColourable_k$ is in $O(k^2)$.*

Proof. Clearly the subformula $Partition$ is the largest one and it uses $O(k^2)$ symbols. □

Proof of Theorem 30. Let G be a graph and k a positive integer. By Theorem 29 we can find a tree decomposition of G in time $tw(G)^{O(tw(G)^3)} \cdot n$ where n is the size of the graph G. Thus we can apply Courcelle's Theorem 31 on this graph, its decomposition, and the formula $ParityColourable_k$. Therefore there exists an algorithm solving ODDCOLOURING(G,k) in time $g(tw(G), |ParityColourable_k|) \cdot n + tw(G)^{O(tw(G)^3)} \cdot n$ for some computable function g. By Lemma 32 we can estimate the size of the formula, and so the problem is solvable in time $g(tw(G), d \cdot k^2 + c) \cdot n + tw(G)^{O(tw(G)^3)} \cdot n$ for some constants c, d. That can be bounded by some other computable function h such that

$$g(tw(G), d \cdot k^2 + c) \cdot n + tw(G)^{O(tw(G)^3)} \cdot n < h(tw(G), k) \cdot n.$$

This concludes the proof. □

Even though we have a proof of Theorem 30, we still do not have the algorithm solving ODDCOLOURING(G,k). We know it exists, but to obtain it, we would have to look deep inside the proof of Courcelle's theorem. The running time of such algorithm depends on the inner structure of the formula $ParityColourable_k$ and thus it is likely to be very high. We rather present our own algorithm solving the problem.

3.3 FPT algorithm computing the parity chromatic number

In this section we prove Theorem 30 by explicitly describing an algorithm solving problem ODDCOLOURING(G,k). This algorithm is an extension of the algorithm used in Section 2.3. Similarly to our previous approach, we first solve a different problem: deciding whether a given colouring is proper.

3.3.1 Verifying the correctness of a colouring using tree decompositions

Theorem 33. *Given a graph G with its nice tree decomposition Γ of width tw and a colouring $c\colon V(G) \to \{1,\dots,k\}$, there exists an algorithm that in time $tw^{O(tw)} \cdot 2^{O(k)} \cdot n$ decides whether the colouring c is a proper parity vertex colouring where n is the maximum from the size of G and the number of nodes in the tree of Γ.*

We explicitly design the algorithm to prove the theorem. Let G be the input graph given with its nice tree decomposition $\Gamma = (T, \{X_t\}_{t \in V(T)})$ and with a colouring c using k colours. We proceed by dynamic programming on the nice tree decomposition, from leaves to the root. We have to check that there is no parity path. But instead of looking at every path separately, we exploit the structure of the tree decomposition. When we were dealing with trees, we used the property that every node of a tree is a separator, so every path connecting two parts separated by the node had to go through that node. Thus for our inductive approach we could forget already processed parts of the tree and only remember the parity vectors of paths ending in the current separator node and contained in already processed parts.

Using a nice tree decomposition, we similarly know by Corollary 28 that for every node t, the bag X_t forms a separator that separates the graph G_t from the rest of G. Thus we can also forget already processed parts of the graph and only remember relevant information in the nodes. When we consider how the intersection of a path with G_t can look like, we see that it is some set of pair-wise disjoint paths with almost all endvertices in the separator X_t (except, possibly, the endvertices of the original path). Therefore, for every possible intersection of a set of pair-wise disjoint paths (paths in G_t and having non-empty intersection with X_t) with the separator X_t, we will remember all possible parity vectors of sets of paths in G_t that have this intersection with the separator. In this way, when we look only on the unprocessed part and on the separator we can determine parity vectors of all paths in G (except the ones entirely in the forgotten part.

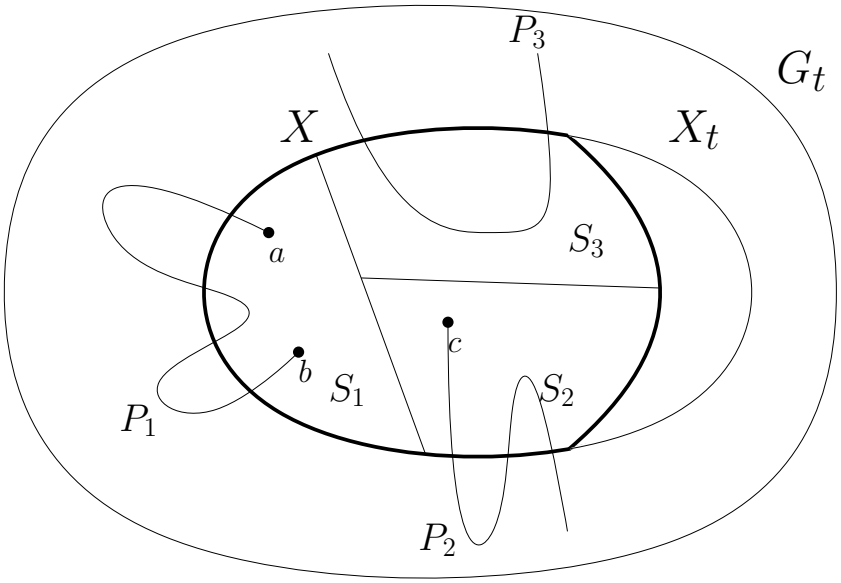

Figure 3.3: A path-partition $\mathbb{P} = \{(\{a, b\}, S_1), (\{c, 0\}, S_2), (\{0, 0\}, S_3)\}$ of (t, X) admitted by a set of paths $\mathbb{F} = \{P_1, P_2, P_3\}$.

We will remember these paths separately). For graphs with bounded treewidth, the size of bags (thus the separators), and consequently the number of possible intersections is constant.

Let us first examine how does the intersection of a set of pair-wise disjoint paths with a bag of a node of T can look like. Let $\mathbb{F} = (P_1, \ldots, P_m)$ be a set of pair-wise disjoint paths in G_t. And let S_i be the intersection of P_i with X_t (see Figure 3.3 for an example). We see that the sets S_i form a partition of some subset of X_t. Additionally, for every path we need to know if and where it ends in X_t. This lead us to the following definitions.

We start with a pair that will represent the intersection of a single path with X_t.

Definition 21. *Let t be a node of T. We call the pair $(\{a, b\}, S)$ a* path-wise pair *(of t) if $S \subseteq X_t$ and $\{a, b\} \subseteq S \cup \{0\}$ where 0 is a special, previously unused, symbol.*

The symbol 0 will represent vertices that are not in X_t.

Remark. A pair $(\{a\}, S)$, for some $S \subseteq X_t$ and $a \in S \cup \{0\}$ is a path-wise pair of t. For these types of pairs having just one element in the first set, we use the notation $(\{a, a\}, S)$ because then all path-wise pairs have the same syntax.

Now we need to define how does the pair represent an intersection of some path with X_t.

Definition 22. *Let t be a node of T and C be a path in G_t such that the intersection S of C with X_t is non-empty. And let $\{a', b'\}$ be the endvertices of C and a be defined as a' if $a' \in X_t$ and as 0 otherwise. And let b be either b' or 0 in a similar way. Then we say that the path C* admits *a path-wise pair $(\{a, b\}, S)$ in the context of t.*

Observe every path C in G_t that has an non-empty intersection with X_t, defines exactly one path-wise pair $(\{a, b\}, S)$ of t admitted by this path. For example on Figure 3.3 paths P_1, P_2, P_3 admit path-wise pairs $(\{a, b\}, S_1)$, $(\{c, 0\}, S_2)$, $(\{0, 0\}, S_3)$, respectively.

As a next step we define a structure that will represent an intersection of a set of paths with X_t.

Definition 23. *Let t be a node of T, X be a non-empty subset of X_t, l be a positive integer, and $\mathbb{P}$ be a set $\{(\{a_i, b_i\}, S_i)\}_{i\in[l]}$ of path-wise pairs of t such that $(S_1, \ldots, S_l)$ form a partition of X (i.e. $\bigcup_{i\in[l]} S_i = X$, $\forall_{i,j\in[l];i\neq j}\ S_i \cap S_j = \emptyset$, and $\forall_{i\in[l]} S_i \neq \emptyset$). Then we call the set $\mathbb{P}$ to be a* path-partition *of (t, X).*

For an example of a path-partition see Figure 3.3.

Remark. Observe that the set X is defined by its partition, so having X as a prerequisite for defining path-partition is redundant, we can just say that $\{S_1, \ldots, S_l\}$ forms a partition of some non-empty subset X of X_t. Even though it is usually more convenient to have the subset of X_t named, we sometimes use an expression *a path-partition of t* instead of (t, X).

It remains to say how exactly a path-partition represents an intersection of a set of pair-wise disjoint paths with X_t.

Definition 24. *Let t be a node of T, X be a non-empty subset of X_t, $\mathbb{P}$ be a path-partition of (t, X), and $\mathbb{F}$ be a non-empty set of pair-wise disjoint paths in G_t. Then we say that $\mathbb{F}$* admits *triple $(t, X, \mathbb{P})$ if $|\mathbb{F}| = |\mathbb{P}|$ and there exists a perfect matching between elements of $\mathbb{P}$ and $\mathbb{F}$ such that every path-wise pair $(\{a, b\}, S) \in \mathbb{P}$ is admitted by the matched path C from $\mathbb{F}$.*

For an example of a path-partition admitted by a set of paths see Figure 3.3. Observe that every t together with every non-empty set $\mathbb{F}$ of pairwise disjoint paths in G_t such that every path from $\mathbb{F}$ has a non-empty intersection with X_t defines exactly one triple $(t, X, \mathbb{P})$ admitted by $\mathbb{F}$ where $X = \bigcup \mathbb{F} \cap X_t$. Thus we can call $\mathbb{P}$ *an intersection of $\mathbb{F}$ with X_t.*

As hinted before, for every possible intersection of sets of pair-wise disjoint paths in G_t with X_t (such that these paths have non-empty intersections with X_t), we want to remember parity vectors of all paths having this intersection with X_t. This motivates the following definition.

Definition 25. *For every node t, every non-empty subset X of X_t, and every path-partition $\mathbb{P}$ of (t, X), we denote by $M(t, X, \mathbb{P})$ the set of parity vectors such that every vector $v \in \{0, 1\}^k$ is in $M(t, X, \mathbb{P})$ if and only if there exists at least one set of paths $\mathbb{F}$ admitting $(t, X, \mathbb{P})$ such that $pv(\bigcup F) = v$. In other words, $M(t, X, \mathbb{P})$ is a union of parity vectors of all sets of paths in G_t admitting the triple $(t, X, \mathbb{P})$.*

Remark. As we stated earlier, in the triple $(t, X, \mathbb{P})$ the set X is defined by $\mathbb{P}$. Thus, we sometimes use $(t, \mathbb{P})$ instead. And similarly we use $M(t, \mathbb{P})$ instead of $M(t, X, \mathbb{P})$.

Our main interest is in parity vectors of paths. Thus we will remember all parity vectors of paths in the already processed part of G in the set N defined as follows.

Definition 26. *For every node t we denote $N(t)$ the set of parity vectors o of dimension k such that there exits a path C in $G_t - X_t$ satisfying $pv(C) = o$.*

Before explaining how to compute these sets we show how they solve our problem.

Lemma 34. *Let r be the root of T. Then there exists a path C in G satisfying $pv(C) = l$ if and only if $l \in N(r)$.*

Proof. From the definition it follows that $l \in N(r)$ if and only if there exists a path C in $G_r - X_r$ such that $pv(C) = l$. Since X_r is empty (by the definition of the nice tree decomposition) and $G_r = G$, it follows that there is no restriction imposed on this path. Thus it concludes the proof. □

Corollary 35. *Let r be the root of T. Then the colouring of G is proper parity colouring if and only if the zero parity vector of dimension k is in $N(r)$.*

Therefore it is sufficient to compute the set $N(r)$ for the root r of our nice tree decompostion. For that purpose we must compute all of the sets $N(t)$, $M(t, X, \mathbb{P})$.

Now we explain how to compute these sets. We proceed inductively on the decomposition. Thus, when we are computing the set $M(t, X, \mathbb{P})$ or $N(t)$, we already know all sets $M(t', X', \mathbb{P}')$ and $N(t)$ for every child t' of t. We show how to compute this set for every type of node in the nice tree decomposition separately.

1. **Forget node.** Suppose that t is a forget node with one son t' such that $X_t = X_{t'} \setminus \{v\}$ and $v \in X_{t'}$. We need to find the set $N(t)$ and also sets $M(t, X, \mathbb{P})$ for all non-empty $X \subseteq X_t$ and all path-partitions $\mathbb{P}$ of (t, X). We start with $N(t)$. We prove that

$$N(t) = N(t') \cup \bigcup_{H \in \{\{0,0\},\{0,v\},\{v,v\}\}} M(t', \{v\}, \{(H, \{v\})\})$$

   We denote the right side of this equality by R.

   Let $o \in N(t)$. From the definition of $N(t)$ it follows that there exists a path F in $G_t - X_t$ satisfying $pv(F) = o$. It either contains v or not. In the former case $F \cap X_{t'} = v$. Thus $\{F\}$ admits $(t', \{v\}, \{(H, \{v\})\})$ where H is either $\{0, 0\}$, $\{0, v\}$, or $\{v, v\}$ (it depends on the endpoints of F; $H = \{0, 0\}$ if both of the endvertices of F are different from v, $H = \{v, 0\}$ if one of them is v and $H = \{v, v\}$ if $F = (v)$). Therefore $o \in M(t', \{v\}, \{(H, \{v\})\})$. In the latter case $F \in G_{t'} - X_{t'}$ and so $o \in N(t')$. Thus in both cases $o \in R$.

   On the other hand, let $o \in R$. Either

$$o \in \bigcup_{H \in \{\{0,0\},\{0,v\},\{v,v\}\}} M(t', \{v\}, \{(H, \{v\})\})$$

   or $o \in N(t')$. In the former case from the definition of M it follows that there exists a set of paths $\mathbb{F}$ admitting $\{(H, \{v\})\}$ such that $pv(\mathbb{F}) = o$. Thus $\mathbb{F}$ consists of only one path C such that $C \cap X_{t'} = \{v\}$. In the latter case $o \in N(t')$, so there exists a path C satisfying $C \cap X_{t'} = \emptyset$ and

$pv(C) = o$. In both cases $C \cap X_t = \emptyset$ and $pv(C) = o$. Therefore $o \in N(t)$.

It remains to find the set $M(t, X, \mathbb{P})$. Denote by M_1 the sets of parity vectors of all sets $\mathbb{F}$ of paths admitting $(t, X, \mathbb{P})$ such the set F covers v. And denote by M_2 the sets $\mathbb{F}$ of parity vectors of all sets of paths admitting $(t, X, \mathbb{P})$ such the set $\mathbb{F}$ does not cover v. It follows that $M(t, X, \mathbb{P}) = M_1 \cup M_2$, so it is sufficient to find the sets M_1 and M_2.

Recall that $v \in X_{t'}$, but $v \notin X_t$ and consequently $v \notin X$. It implies that every set of paths admitting $(t', X, \mathbb{P})$ does not cover v and it admits $(t, X, \mathbb{P})$. And conversely, every set of paths admitting $(t, X, \mathbb{P})$ and not covering v admits $(t', X, \mathbb{P})$. Thus

$$M_2 = M(t', X, \mathbb{P}).$$

Now let us proceed to M_1. For every $P = (\{a, b\}, S) \in \mathbb{P}$ (note that always $S \neq \emptyset$ and $v \notin S$) define $P' := (\{a, b\}, S \cup \{v\})$ a path-wise pair of t'. Moreover, if $a = 0$ define $P'' := (\{v, b\}, S \cup \{v\})$ a path-wise pair of t'. And if $b = 0$ define $P'' := (\{a, v\}, S \cup \{v\})$ (if $a = b = 0$ these pairs are the same). We will show that

$$M_1 = \bigcup_{\mathbb{P}'} M(t', X \cup \{v\}, \mathbb{P}') \tag{3.1}$$

where the union is taken over all possible $\mathbb{P}'$ obtained from $\mathbb{P}$ by replacing one $P \in \mathbb{P}'$ with one of P' or P'' (if P'' exists). The union includes both of these possibilities. Denote the right side of Equation (3.1) by R.

Let $o \in M_1$. Then there exists a set of paths $\mathbb{F}$ with parity vector o such that $\mathbb{F}$ admits $(t, X, \mathbb{P})$, and $v \in \bigcup \mathbb{F}$. Thus, there is a path $F \in \mathbb{F}$ satisfying $v \in F$ and admitting some $P = (\{a, b\}, S) \in \mathbb{P}$ (in the context of t). This path admits (in the context of t') either P' if F does not end in v, or P'' otherwise. Since every other path from $\mathbb{F}$ has the same intersection with X_t as with $X_{t'}$, it follows that $\mathbb{F}$ admits $(t', X \cup \{v\}, \mathbb{P}')$, where $\mathbb{P}'$ is obtained from $\mathbb{P}$ by replacing P with P' or P''. Therefore $o \in R$.

On the other hand, let $o \in R$. Then there exists a set of paths $\mathbb{F}$ with parity vector o such that $\mathbb{F}$ admits $(t', X \cup \{v\}, \mathbb{P}')$ where $\mathbb{P}'$ is obtained from $\mathbb{P}$ by replacing one $P \in \mathbb{P}$ with either P' or P''. Thus there exists a path $F \in \mathbb{F}$ such that $v \in F$, and F admits P' or P'' (The one that is actually in $\mathbb{P}'$. It also decides whether F ends in v or not). In both cases F admits also P (in context to t). Hence $\mathbb{F}$ admits $(t, X, \mathbb{P})$, therefore $o \in M_1$.

2. **Introduce vertex node.** Suppose that t is an introduce vertex node with one son t' such that $X_t = X_{t'} \cup \{v\}$ and $v \notin X_{t'}$. Recall that every edge needs to be introduced to be in G_t, but it can not happen before introducing incident vertices. Thus vertex v is isolated in G_t. We need to find the sets $N(t)$ and $M(t, X, \mathbb{P})$ (for all X and $\mathbb{P}$). First observe that $N(t) = N(t')$ because the graph $G_t - X_t$ is the same as $G_{t'} - X_{t'}$, and so a path in one of these graphs is also a path in the second one.

Now we prove that

$$M(t,X,\mathbb{P}) = \begin{cases} M(t',X,\mathbb{P}) & \text{if } v \notin X, \\ pv(v) & \text{if } X = \{v\} \text{ and } \\ & \quad \mathbb{P} = \{(\{v,v\},\{v\})\}, \\ M(t',X\setminus\{v\},\mathbb{P}') \oplus pv(v) & \text{if } v \in X,\ |X| \geq 2, \text{ and } \\ & \quad (\{v,v\},\{v\}) \in \mathbb{P}, \\ \emptyset & \text{otherwise.} \end{cases}$$

where $\mathbb{P}' = \mathbb{P} \setminus \{(\{v,v\},\{v\})\}$.

First assume that $v \notin X$. Then every set of paths admitting $(t,X,\mathbb{P})$ does not use v, thus this set lies also in $G_{t'}$, therefore it admits also $(t',X,\mathbb{P})$. Since the converse trivially holds as well, it follows that $M(t,X,\mathbb{P}) = M(t',X,\mathbb{P})$. From now on assume that $v \in X$ and let $P := (\{v,v\},\{v\})$.

Assume that $P \notin \mathbb{P}$, and suppose that there exists a set of paths $\mathbb{F}$ admiting $(t,X,\mathbb{P})$. The path-partition $\mathbb{P}$ contains some path-wise pair of t, say $A = (\{a,b\},S)$ such that $v \in S$. This pair must be admitted by some path F from $\mathbb{F}$. But that is not possible because F can consist of only the vertex v (v is isolated), and so F can admit only the pair P, which is not in $\mathbb{P}$. It follows that there is no set of paths admitting $M(t,X,\mathbb{P})$. Therefore $M(t,X,\mathbb{P}) = \emptyset$.

If $X = \{v\}$ and $\mathbb{P} = \{(\{v,v\},\{v\})\}$. Then $(t,X,\mathbb{P}) = pv(v)$ because $\mathbb{P}$ can be admitted only by $\{(v)\}$ (v is isolated).

Now assume that $P \in \mathbb{P}$ and that $|X| \geq 2$. Let $o \in M(t,X,\mathbb{P})$. Then there exists a set of paths $\mathbb{F}$ with a parity vector o, covering vertex v, and admitting $(t,X,\mathbb{P})$. The only possible path admitting $(\{v,v\},\{v\})$ is the path consisting of only vertex v because v is isolated. Thus, there exists a path $F \in \mathbb{F}$ such that $F = (v)$. Let $\mathbb{F}' := \mathbb{F} \setminus \{F\}$ (observe that $\mathbb{F}'$ is not empty). Clearly $\mathbb{F}'$ admits $(t, X\setminus\{v\}, \mathbb{P}\setminus\{P\})$. Since $X_{t'} = X_t \setminus \{v\}$ and $v \notin \bigcup \mathbb{F}'$, it follows that $\mathbb{F}'$ admits also $(t', X\setminus\{v\}, \mathbb{P}\setminus\{P\})$. Therefore $o + pv(v) \in M(t', X\setminus\{v\}, \mathbb{P}\setminus\{P\})$.

On the other hand, let $o \in M(t', X\setminus\{v\}, \mathbb{P}\setminus\{P\})$. Then there exists a set of paths $\mathbb{F}$ (because $|X\setminus\{v\}| \geq 1$) with the parity vector o and admitting $(t', X\setminus\{v\}, \mathbb{P}\setminus\{P\})$. Note that $\mathbb{F}$ does not cover v, so it holds that $\mathbb{F}\cup\{(v)\}$ admits $(t,X,\mathbb{P})$. Thus $o + pv(v) \in M(t,X,\mathbb{P})$.

3. **Join node.** Suppose that t is a join node with two son t_1, t_2. It implies that $X_t = X_{t_1} = X_{t_2}$. Note that by Lemma 28 it follows that V_{t_1} and V_{t_2} are separated by X_t. We need to find the sets $M(t,X,\mathbb{P})$ and $N(t)$.

   The set $N(t)$ contains parity vectors of all paths in G_t that have an empty intersection with X_t. Since vertices in V_{t_1} and V_{t_2} can be connected only through X_t, it follows that these paths can only be entirely in G_{t_1} or in G_{t_2}. Therefore

$$N(t) = N(t_1) \cup N(t_2).$$

Observe that every edge of G_t is either in G_{t_1} or G_{t_2} but not in both because every edge is introduced only once. Thus every path in G_t can be

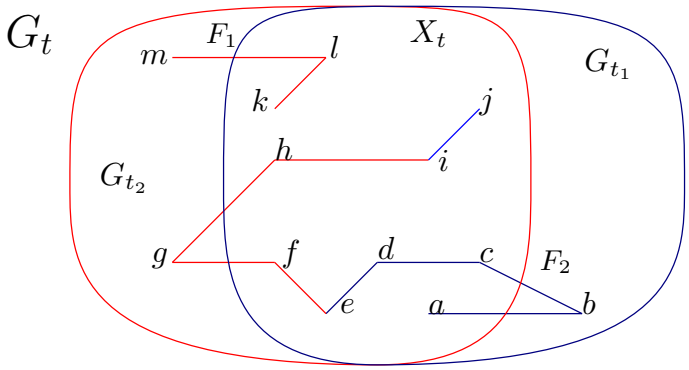

Figure 3.4: An example of a set of pair-wise disjoint paths in G_t where t is a join node. Blue edges belong to G_{t_1} and red ones belong to G_{t_2}. The set of paths $\{F_1, F_2\}$ splits into sets of paths $\{(a,b,c,d,e),(i,j)\}$ and $\{F_1,(e,f,g,h,i)\}$. The admitted path-partitions $\{(\{a,e\},\{a,c,d,e\}),(\{i,j\},\{i,j\})\}$ and $\{(\{0,k\},\{k,l\}),(\{e,i\},\{e,f,h,i\})\}$ alternately compose $\{(\{0,k\},\{k,l\}),(\{a,j\},\{a,c,d,e,f,h,i,j\})\}$ that is admitted by $\{F_1,F_2\}$.

split to two sets of pair-wise disjoint paths, one containing edges from G_{t_1} and one from G_{t_2}. This leads to the following definition.

Definition 27. *Let $\mathbb{F}$ be a set of pair-wise disjoint paths in G_t. And let $\mathbb{F}_1, \mathbb{F}_2$ be sets of pair-wise disjoint paths obtained from $\mathbb{F}$ by splitting (in vertices) every path of length at least 1 in $\mathbb{F}$ to maximal paths (of length at least 1) contained entirely in G_{t_1}, or G_{t_2} such that $\mathbb{F}_1$ contains the resulting paths in G_{t_1}, and $\mathbb{F}_2$ contain the resulting paths in G_{t_2}, and the original paths of length 0 are also each in one of these sets. We say that $\mathbb{F}$* splits *to $\mathbb{F}_1$ and $\mathbb{F}_2$. Alternatively, we say that $\mathbb{F}_1$ and $\mathbb{F}_2$ compose $\mathbb{F}$.*

For an example how a set of paths in G_t splits into two sets of paths see Figure 3.4.

If we take the split paths in the order of the original paths, then the paths from G_{t_1} and G_{t_2} will alternate (see Figure 3.4). We have to be able to describe all of this by our path-partition terminology. Thus we need some more definitions.

Definition 28. *Let $A := (\{a'_1, a'_l\}, \{a_1, \ldots, a_l\})$ be a path-wise pair of t such that a'_1 is either a_1 or 0 and a'_l is either a_l or 0. We say that A is* splittable into path-wise pairs $(A_1, A_2, \ldots, A_p)$, *if $A_1 = A$ in case $p = 1$ and in case $p > 1$ if there exist integers $b_1, b_2, \ldots, b_{p-1}$ such that $1 \leq b_1 < b_2 < \cdots < b_{p-2} < b_{p-1} \leq l$, $a_{b_1} \neq a'_1$, $a_{b_{p-1}} \neq a'_l$, and*

$$\begin{aligned}
A_1 &= (\{a'_1, a_{b_1}\}, \{a_1, \ldots, a_{b_1}\}), \\
A_2 &= (\{a_{b_1}, a_{b_2}\}, \{a_{b_1}, \ldots, a_{b_2}\}), \\
&\vdots \\
A_{p-1} &= \left(\{a_{b_{p-2}}, a_{b_{p-1}}\}, \{a_{b_{p-2}}, \ldots, a_{b_{p-1}}\}\right), \\
A_p &= \left(\{a_{b_{p-1}}, a'_l\}, \{a_{b_{p-1}}, \ldots, a_l\}\right).
\end{aligned}$$

We say that $(A_1, A_2, \ldots, A_p)$ *compose* A.

Observe that if $(A_1, A_2, \ldots, A_p)$ compose A and $p > 1$, then each A_i can be admitted only by paths of length least 1.

Example 2. A pair-wise pair $A = (\{a, b\}, \{a, b, c, d, e, f, g\})$ is splittable into pairs $\Big((\{a, d\}, \{a, e, g, d\}), (\{d, f\}, \{d, c, f\}), (\{f, b\}, \{f, b\})\Big)$.

Additionally, we need the paths to be composed alternately. Thus we need to define alternate composition also for path-partitions. Recall that path-partitions are special types of sets of path-wise pairs.

Definition 29. *We say that path-partitions (sets of path-wise pairs)* $\mathbb{A}$ *and* $\mathbb{B}$ alternately compose *a path-wise pair* C *if* $\mathbb{A} \cup \mathbb{B}$ *can be ordered so that it composes* C *and the pairs from* $\mathbb{A}$ *and* $\mathbb{B}$ *in the ordering of this composition alternate*[3].

We say that path-partitions $\mathbb{A}$ *and* $\mathbb{B}$ alternately compose *a path-partition* $\{C_1, \ldots, C_m\}$ *if we can partition* $\mathbb{A}, \mathbb{B}$ *into* $(\mathbb{A}_1, \ldots, \mathbb{A}_m)$ *and* $(\mathbb{B}_1, \ldots, \mathbb{B}_m)$ *so that*

(a) $\bigcup_i \mathbb{A}_i = \mathbb{A}$ *and* $\bigcup_i \mathbb{B}_i = \mathbb{B}$,

(b) for all distinct i, j *in* $[m]$: $\mathbb{A}_i \cap \mathbb{A}_j = \emptyset$ *and* $\mathbb{B}_i \cap \mathbb{B}_j = \emptyset$,

(c) for all i *in* $[m]$: $A_i \neq \emptyset$, *or* $B_i \neq \emptyset$,

(d) for all i *in* $[m]$: $\mathbb{A}_i$ *and* $\mathbb{B}_i$ *alternately compose* C_i.

Example 3. The sets $\{(\{a, d\}, \{a, e, g, d\}), (\{f, b\}, \{f, b\})\}$ and $\{(\{d, f\}, \{d, c, f\})\}$ alternately compose $(\{a, b\}, \{a, b, c, d, e, f, g\})$.

For another example see Figure 3.4. We now show how does the splittability of a set of paths in G_t to sets of paths in G_{t_1} and G_{t_2} corresponds to the splittability of an admitted path-partition.

Lemma 36. *Let* $\mathbb{F}$ *be a set of paths in* G_t *such that* $\mathbb{F}$ *splits to* $\mathbb{F}_1, \mathbb{F}_2$. *And let* $\mathbb{P}$ *be the intersection of* $\mathbb{F}$ *with* X_t, $\mathbb{P}_1$ *be the intersection of* $\mathbb{F}_1$ *with* X_t, *and* $\mathbb{P}_2$ *be the intersection of* $\mathbb{F}_2$ *with* X_t. *And let* $(t, X, \mathbb{P})$, $(t_1, X_1, \mathbb{P}_1)$, *and* $(t_2, X_2, \mathbb{P}_2)$ *be the triples admitted by* $\mathbb{F}$, $\mathbb{F}_1$, *and* $\mathbb{F}_2$, *respectively. Then* $\mathbb{P}_1$ *and* $\mathbb{P}_2$ *alternately compose* $\mathbb{P}$.

Proof. It is sufficient to prove it just for $\mathbb{F}$ containing only one path. If $\mathbb{F}$ contained more paths, we could split $\mathbb{F}_1$ and $\mathbb{F}_2$ to parts according to the paths of $\mathbb{F}$ they belonged to and prove it separately. Thus let $\mathbb{F} = \{F\}$. Additionally, assume that F splits into the same amount of paths in G_{t_1} and G_{t_2}. Due to the maximality of split paths, two paths from the same $\mathbb{F}_i$ can not be next to each other (as subpaths of F). Thus the number of paths in $\mathbb{F}_1$ and $\mathbb{F}_2$ can differ at most by one, so other cases are analogous. Thus $\mathbb{F}_1$ and $\mathbb{F}_2$ have the following form: $\{F_1^1, \ldots, F_1^a\} = \mathbb{F}_1$ and $\{F_2^1, \ldots, F_2^a\} = \mathbb{F}_2$ such that

$$F = F_1^1 \circ F_2^1 \circ \cdots \circ F_1^a \circ F_2^a \tag{3.2}$$

where the $\circ$ denotes concatenation by vertex (i.e. the endvertex of one path is joined with the startvertex of the consecutive path).

[3]Specifically, $\{C\}$ and $\{\}$ alternately compose C.

The set of paths $\mathbb{F}$ admits triple $(t, F \cap X_t, \{(\{x, y\}, F \cap X_t)\})$ (where x corresponds to the startvertex of F and y corresponds to the the endvertex of F), $\mathbb{F}_1$ admits triple $(t_1, X_1, \mathbb{P}_1)$ (defined by the intersection of $\mathbb{F}_1$ with X_t), and $\mathbb{F}_2$ admits triple $(t_2, X_2, \mathbb{P}_2)$ (defined by the intersection of $\mathbb{F}_2$ with X_t).

Let $s(F')$, $e(F')$ denote the startvertex and the endvertex (relative to the path F), respectively, of every subpath F' of F. Then F_1^1 admits the path-wise pair $(\{x, e(F_1^1)\}, F_1^1 \cap X_t)$, F_2^a admits the pair $(\{y, s(F_2^a)\}, F_2^a \cap X_t)$, and all other F_i^j's admit the pairs $(\{s(F_i^j), e(F_i^j)\}, F_i^j \cap X_t)$. In particular, note that the vertices where the paths are joined together (thus all start and end vertices of all F_i^j, except the two corresponding to the start and the end of the original path) lie in X_t. Thus, we can order all path-wise pairs according to (3.2) and it can be easily seen that $\mathbb{P}_1 \cup \mathbb{P}_2$ alternately compose the pair $(\{x, y\}, F \cap X_t)$. □

Lemma 37. *Let $\mathbb{P}_1$ and $\mathbb{P}_2$ be path-partitions of $(t_1, X_1), (t_2, X_2)$, respectively, such that they alternately compose some path-partition $\mathbb{P}$ of (t, X). Let $\mathbb{F}_1, \mathbb{F}_2$ be sets of paths admitting $(t_1, X_1, \mathbb{P}_1)$ and $(t_2, X_2, \mathbb{P}_2)$, respectively. Then there exists a set of paths $\mathbb{F}$ such that $\mathbb{F}_1$ and $\mathbb{F}_2$ compose $\mathbb{F}$ and $\mathbb{F}$ admits $(t, \mathbb{P})$.*

Proof. Note that $\mathbb{F}_1$ lies entirely in G_{t_1} and $\mathbb{F}_2$ lies entirely in G_{t_2}. If some pairs from $\mathbb{P}_1$ and $\mathbb{P}_2$ alternaly compose some other pair P, then we can connect the corresponding admitting paths in the endvertices through which the corresponding pairs are connected, and obtain a simple path admitting P. Therefore, since $\mathbb{P}_1$ and $\mathbb{P}_2$ alternately compose $\mathbb{P}$, we can in similar fashion compose the paths from $\mathbb{F}_1$ and $\mathbb{F}_2$ to obtain a set of paths $\mathbb{F}$ admitting $(t, \mathbb{P})$. □

The last piece we need to know is how the parity vector of a set of paths in G_t and the parity vectors of its split (two set of paths in G_{t_1} and in G_{t_2}) are related.

Lemma 38. *Let $\mathbb{F}_1$, $\mathbb{F}_2$ be sets of paths in G_{t_1} and G_{t_2}, respectively, such that they compose some set of paths $\mathbb{F}$ in G_t. Let $(t_1, X_1, \mathbb{P}_1)$ be admitted by $\mathbb{F}_1$ and $(t_2, X_2, \mathbb{P}_2)$ be admitted by $\mathbb{F}_2$ (as we know these triples are defined uniquely). Then $pv(\mathbb{F}) = pv(\mathbb{F}_1) + pv(\mathbb{F}_2) + pv(X_2 \cap X_1)$.*

Proof. Since G_{t_1} and G_{t_2} are separated by X_t, it follows that $\bigcup \mathbb{F}_1 \cap \bigcup \mathbb{F}_2 \subseteq X_t$. For X_1 and X_2 it holds that $X_1 = \bigcup \mathbb{F}_1 \cap X_t$, and $X_2 = \bigcup \mathbb{F}_2 \cap X_t$. Thus $\bigcup \mathbb{F}_1 \cap \bigcup \mathbb{F}_2 = X_1 \cap X_2$. Obviously it holds that $\bigcup \mathbb{F}_1 \cup \bigcup \mathbb{F}_2 = \bigcup \mathbb{F}$. Moreover, the paths in $\mathbb{F}_1$, in $\mathbb{F}_2$, as well as in $\mathbb{F}$ are disjoint, thus in $pv(\mathbb{F}_1) + pv(\mathbb{F}_2) + pv(X_2 \cap X_1)$ we count the colour of every vertex covered by $\mathbb{F}$ odd number of times and the proof is concluded. □

We prove that we can compute $M(t, X, \mathbb{P})$ in the following way.

$$M(t, X, \mathbb{P}) = \bigcup_{X_1, \mathbb{P}_1, X_2, \mathbb{P}_2} (M(t_1, X_1, \mathbb{P}_1) \oplus (M(t_2, X_2, \mathbb{P}_2) \oplus pv(X_2 \cap X_1))) \tag{3.3}$$

where the union is taken over all non-empty X_1, X_2 and over all path-partitions $\mathbb{P}_1, \mathbb{P}_2$ of (t_1, X_1), (t_2, X_2), respectively such that $\mathbb{P}_1$ and $\mathbb{P}_2$ alternately compose $\mathbb{P}$ (thus $X_1 \cup X_2 = X$).

Denote L the left side of Equality (3.3), and R the union on the right side. Let $v \in L$. There exists a set of paths $\mathbb{F}$ admitting $(t, X, \mathbb{P})$ such that $pv(\mathbb{F}) = v$. We split $\mathbb{F}$ into $\mathbb{F}_1$ and $\mathbb{F}_2$ ($\mathbb{F}_1$, $\mathbb{F}_2$ lie in G_{t_1}, G_{t_2}, respectively). Let $(t_1, X_1, \mathbb{P}_1)$ and $(t_2, X_2, \mathbb{P}_2)$ be the triples admitted by $\mathbb{F}_1$ and $\mathbb{F}_2$, respectively. By Lemma 36, $\mathbb{P}_1$ and $\mathbb{P}_2$ alternately compose $\mathbb{P}$. Moreover, $pv(\mathbb{F}) = pv(\mathbb{F}_1) + pv(\mathbb{F}_2) + pv(X_2 \cap X_1)$ according to Lemma 38. It follows that $v \in R$.

On the other hand take v from R. Then there exist path sets $\mathbb{F}_1, \mathbb{F}_2$ admitting $(t_1, X_1, \mathbb{P}_1)$ and $(t_2, X_2, \mathbb{P}_2)$, respectively. Since $\mathbb{P}_1$ and $\mathbb{P}_2$ alternately compose $\mathbb{P}$, Lemma 37 implies that there exists a set of paths $\mathbb{F}$ admitting $\mathbb{P}$ such that $\mathbb{F}_1$ and $\mathbb{F}_2$ compose $\mathbb{F}$. Moreover, $pv(\mathbb{F}) = pv(\mathbb{F}_1) + pv(\mathbb{F}_2) + pv(X_2 \cap X_1)$ according to Lemma 38. Thus $v \in L$. And the case of a join node is concluded.

4. **Leaf node.** Suppose that t is a leaf node. Recall that for a leaf t it holds that $X_t = \emptyset$. Since the only subset of an empty set is an empty set, there is no path-partition of t and consequently no set $M(t, M, \mathbb{P})$ to be computed. Therefore we only need to find $N(t)$. It holds that $N(t) = \emptyset$ because there is no path in the graph G_t.

5. **Introduce edge node.** Suppose that t is an introduce edge node introducing an edge $e = \{v, u\}$ and let t' be the only son of t. It means that $X_t = X_{t'}$, $E_t = E_{t'} \cup \{e\}$, and $e \notin E_{t'}$. We need to find the sets $N(t)$ and $M(t, X, \mathbb{P})$. It is easy to see that $N(t) = N(t')$ because the graphs $G_t - X_t$ and $G_{t'} - X_{t'}$ are the same.

   It remains to find the value of $M(t, X, \mathbb{P})$. Let M_2 be the set of parity vectors of sets of paths admitting $(t, X, \mathbb{P})$ such that the path sets covers the edge e. And let M_1 be the set of parity vectors of sets of paths admitting $(t, X, \mathbb{P})$ such that the path sets do not cover the edge e. Clearly, $M(t, X, \mathbb{P}) = M_1 \cup M_2$.

   Recall that $X_t = X_{t'}$ and that $G_t - e = G_{t'}$, so every set of paths admits $(t, X, \mathbb{P})$ and avoids e if and only if it admits $(t', X, \mathbb{P})$. Therefore

$$M_1 = M(t', X, \mathbb{P}).$$

   In the next part we will be splitting one path from a set of paths in the edge $\{v, u\}$ (because $\{v, u\}$ is in G_t, but not in G'_t). Since path-wise pairs represent paths, we need to be able to split path-wise pairs.

   Let $A = (\{a, b\}, S)$ be a path-wise pair of t such that $u, v \in S$. We say that A is *splittable in the edge* $\{v, u\}$ into path-wise pairs $(\{a, c_1\}, S_1)$, $(\{b, c_2\}, S_2)$ if the following conditions are satisfied:

   (a) $c_1 \in S_1$, and $c_2 \in S_2$,

   (b) $\{c_1, c_2\} = \{u, v\}$,

   (c) $S_1 \cup S_2 = S$,

   (d) $S_1 \cap S_2 = \emptyset$,

   (e) $a \in S_1 \cup \{0\}$, and $b \in S_2 \cup \{0\}$.

Now we show that

$$M_2 = \begin{cases} \bigcup_{A_1,A_2} M(t, X, \mathbb{P}'), & \text{if } \exists A = (\{a, b\}, S) \in \mathbb{P} : u, v \in S \\ \emptyset, & \text{otherwise} \end{cases}$$

where $\mathbb{P}' = \mathbb{P} \cup \{A_1, A_2\} \setminus \{A\}$ and the big union is taken over all path-wise pairs A_1, A_2 such that A is splittable into A_1, A_2 in the edge $\{u, v\}$.

First, assume that u, v are not in the same set S of any pair $(\{a, b\}, S)$ from $\mathbb{P}$. Then no set of paths covering the edge e can admit $M(t, X, \mathbb{P})$.

Now assume that there exists $P \in \mathbb{P}$ such that $P = (\{a, b\}, S)$ and $u, v \in S$.

Let $o \in M_2$. There exists $\mathbb{F}$ such that $pv(\mathbb{F}) = o$, $\mathbb{F}$ admits $(t, X, \mathbb{P})$, and e is covered by $\mathbb{F}$. Thus there exists a path C from $\mathbb{F}$ admitting P. By assumption, e is used in $\mathbb{F}$, hence e must be a part of C. Therefore we can split C in e into two paths C_1, C_2 such that they admit pairs $P_1 = (\{a, x\}, C_1 \cap S)$, $P_2 = (\{y, b\}, C_2 \cap S)$ respectively, where x is the vertex from u, v that is in C_1, and y is the other one (the one that is in C_2). Note that it does not depend whether we mean the context of t or t' since $X_t = X_{t'}$ and C_1, C_2 are in both G_t and $G_{t'}$. Observe that P is splittable in the edge $\{v, u\}$ into pairs P_1, P_2. Therefore $\mathbb{F} \setminus \{C\} \cup \{C_1, C_2\}$ admits the triple $(t', X, \mathbb{P} \setminus \{P\} \cup \{P_1, P_2\})$. Since this new set of path has the same parity vector, it follows that $o \in M(t', X, \mathbb{P} \setminus \{P\} \cup \{P_1, P_2\})$.

On the other hand, let $o \in (t', X, \mathbb{P}')$ where $\mathbb{P}' = \mathbb{P} \setminus \{P\} \cup \{P_1, P_2\}$ such that P is splittable in the edge $\{v, u\}$ into pairs P_1, P_2. Then there exists a set of paths $\mathbb{F}$ admitting the triple $(t', X, \mathbb{P}')$ such that $pv(\mathbb{F}) = o$. Since e is not in $G_{t'}$, it can not be a part of $\mathbb{F}$. Let C_2, C_1 be the paths from $\mathbb{F}$ admitting P_1, P_2, respectively. We can connect them by an edge e and obtain a path C that admits the pair P in the context of t. Let $\mathbb{F}_1 := \mathbb{F} \setminus \{C_1, C_2\} \cup \{C\}$. It follows that $pv(\mathbb{F}_1) = pv(\mathbb{F})$ and $\mathbb{F}_1$ admits the triple $(t, X, \mathbb{P})$. Therefore $o \in M_2$.

Proof of Theorem 33. We are given a graph G with its nice tree decomposition Γ and a colouring c of $V(G)$ and we want to check if there exists a parity path in G. Let tw be the width of Γ and n be the maximum from the size of G and the number of nodes in the tree of Γ. First, for every node t of a nice tree decomposition Γ, all non-empty $X \subseteq X_t$, and all path-partitions $\mathbb{P}$ of (t, X), we compute the set $N(t)$ and the sets $M(t, X, \mathbb{P})$ by the dynamic programming explained above. By Corollary 35, there exists a parity path in G if and only if $z \in N(r)$ where z is the zero vector of dimension k and r is the root of Γ.

Let us briefly examine the running time of this algorithm. Every bag X_t has cardinality at most $tw+1$. Thus there are at most $(tw+1)^{tw+1}$ partitions of subsets of X_t for each node. Therefore there are at most $(tw+1)^{tw+1} \cdot (tw+2)^2$ path-partitions of t (the pair representing endvertices has at most $(tw+2)^2$ distinct values). Consequently there are at most that many sets M to be computed for every node. Each set M, N contains at most 2^k parity vectors. Additionally, the formulas to compute every set (M or N) for each node t requires considering at most all the pairs of sets M, N for some other nodes. And so the formulas are straightforwardly implementable in time polynomial with the number of path-partitions and the size of required sets M, N. Thus the running time is in $tw^{O(tw)} \cdot 2^{O(k)}$ for every node. Therefore it is $tw^{O(tw)} \cdot 2^{O(k)} \cdot n$ in total. □

Let us give a name `verifycolouring` to this algorithm because we will refer to it in the next section.

3.3.2 Computing the parity chromatic number using tree decompositions

Now we describe FPT algorithm for the problem ODDCOLOURING(G,k), which proves Theorem 30. Let us first assume we have a graph G given with its nice tree decomposition $\Gamma = (T, \{X_i\}_{i \in V(T)})$. We want to determine whether G has a parity vertex colouring with k colours. The assumption that we are given the graph with its nice tree decomposition is not a problem because, as we already know from Theorem 29 and Lemma 26, it can be constructed by a fixed parameter algorithm.

We could try all possible colourings of G and verify them, one by one, using the algorithm `verifycolouring`. But the running time would be exponential with the size of G. We rather use the same idea we used for the FPT algorithm finding the parity chromatic number of a tree. That is, we take the algorithm `verifycolouring` and proceed similarly on the decomposition. When we had just one fixed colouring, the "state" of each vertex was fixed. Now we will remember all possible "states" for each node. By "state" we mean the evaluation of all sets M, N together with colours of X_t (later we show that we do not need the colours of X_t). For computing possible "states" for each vertex we will take all possible "states" of its children and compute "states" of the new vertex by the algorithm `verifycolouring`. There are at most k^{tw+1} colourings of each X_t and at most $(tw+1)^{tw+1} \cdot (tw+2)^2$ sets M. And each set M, N can have at most 2^{2^k} distinct values. Thus for each node the number of distinct "states" is at most $k^{tw+1} \cdot (2^{2^k})^{1+(tw+1)^{tw+1} \cdot (tw+2)^2}$, which for a bounded tw and k is bounded. Let us now describe this algorithm formally.

We will use the same notation of sets $M(t), N(t)$ for each node t as in the previous section. But we do not have a fixed colouring, and so for a colouring c of G we denote the sets M, N corresponding to c by M_c, N_c. We also use the shorter notation of path-partitions and sets M: a path-partition of t (instead of a path-partition of (t, X)), and $M(t, \mathbb{P})$ instead of $M(t, X, \mathbb{P})$.

We need to be able to capture all possible "states" of each node without depending on a colouring. Thus we will gradually define an evaluation of path-partitions of t (corresponding to sets M), then an evaluation of all path-partitions of t together, and finally an evaluation of processed paths of t (corresponding to sets $N(t)$). For each of these we also specify how it corresponds to colourings.

We start with an evaluation of a single path-partition.

Definition 30. *Let t be a node of T and $\mathbb{P}$ be a path-partition of t. We call the pair $((t, \mathbb{P}), Y)$ where Y is a non-empty set of parity vectors of dimension k an* evaluated path-partition of $\mathbb{P}$. *We say that the evaluated path-partition is evaluated by Y.*

Definition 31. *Let t be a node of T, $\mathbb{P}$ be a path-partition of t, $((t, \mathbb{P}), Y)$ be an evaluated path-partition $\mathbb{P}_{ev}$, and c be a colouring of G. We say that the colouring c* admits *$\mathbb{P}_{ev}$ if $Y = M_c(t, \mathbb{P})$.*

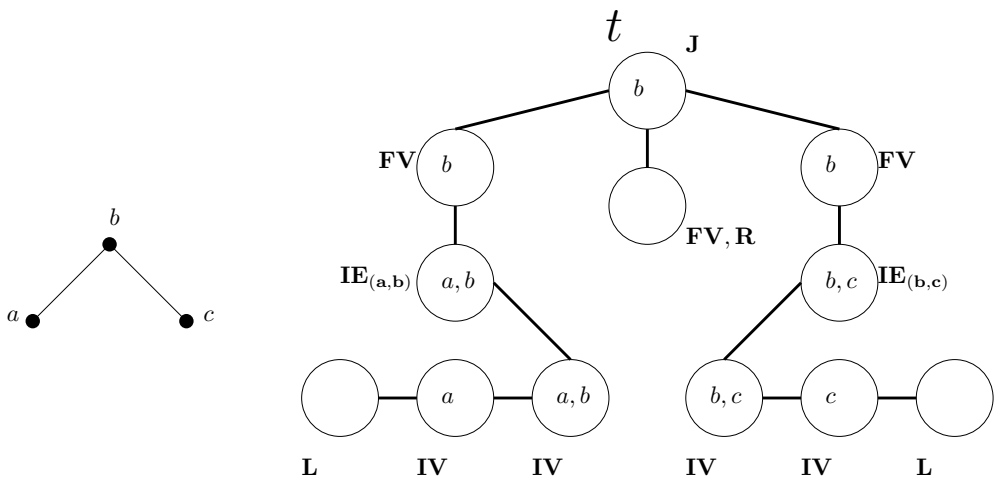

Figure 3.5: An example of a graph G with its nice tree decomposition with a join node t.

Observe that given a colouring c of G, a node t, and a path-partition $\mathbb{P}$ of t, there exists exactly one set Y of parity vectors so that c admits an evaluated path-partition of $\mathbb{P}$ evaluated by Y. Specifically, $Y = M_c(t, \mathbb{P})$.

Example 4. Consider a graph G with its nice tree decomposition as on Figure 3.5. Let t be the join node of the decomposition as on the Figure. Assume $k = 2$. Let *col* be a colouring of G given as $col(a) = 1$, $col(b) = 1$, $col(c) = 2$ and $\mathbb{P} = \{(\{0, b\}, \{b\})\}$ be a path-partition of t. Observe that the only paths admitting $\mathbb{P}$ (in context of t) are (a, b) and (b, c). Then an evaluated path-partition $\mathbb{P}_{ev} = ((t, \mathbb{P}), \{(0, 0), (1, 1)\})$ is admitted by *col* because $(0, 0)$ is the parity vector of (a, b) and $(1, 1)$ is the parity vector of (b, c). Simultaneously, $\mathbb{P}_{ev}$ is the only evaluated path-partition of $\mathbb{P}$ admitted by *col* and $M_{col}(t, \mathbb{P}) = \{(0, 0); (1, 1)\}$.

We continue with defining an evaluation of all path-partitions of one node.

Definition 32. *Let t be a node of T, P be a family of all path-partitions of t, and for every $\mathbb{P} \in P$ let $\mathbb{P}_{ev}$ be an evaluated path-partition of $\mathbb{P}$ evaluated by some set $Y_{\mathbb{P}}$ of parity vectors. We call the set $\mathbb{M}(t)$ of all such $\mathbb{P}_{ev}$ an* evaluated separator of t.

Definition 33. *Let t be a node of T, and c a be a colouring of G_t with k colours, and $\mathbb{M}(t)$ be an evaluated separator of t. We say that the colouring c* admits $\mathbb{M}(t)$ *if c admits all evaluated path-partitions from $\mathbb{M}(t)$.*

Observe that given a node t and a colouring c of G, there exists exactly one evaluated separator of t admitted by the colouring c. Specifically, it is the set of evaluated path-partitions of all path-partitions $\mathbb{P}$ of t admitted by c (as we know, for every path-partition there is exactly one such evaluated path-partition). Thus, it corresponds to the set of all $M_c(t, \mathbb{P})$ for all path-partition $\mathbb{P}$ of t. Let us denote this evaluated separator by $\mathbb{M}_c(t)$.

Example 5. Consider a graph G with its nice tree decomposition as on Figure 3.5. Assume $k = 2$. Let *col* be a colouring of G given as $col(a) = 1$, $col(b) = 1$, $col(c) = 2$. Let $\mathbb{P}_1 := \{(\{0, 0\}, \{b\})\}$, $\mathbb{P}_2 := \{(\{b, 0\}, \{b\})\}$, and $\mathbb{P}_3 := \{(\{b, b\}, \{b\})\}$. Then a set P of all path-partition of t is equal to $\{\mathbb{P}_1, \mathbb{P}_2, \mathbb{P}_3\}$. Observe that $\mathbb{P}_1$ is

admitted only by path (a, b, c), $\mathbb{P}_2$ is admitted only by paths (a, b), (b, c), and $\mathbb{P}_3$ is admitted only by path (b).

Thus an evaluated separator

$$\mathbb{M}(t) = \left\{ \Big((t, \mathbb{P}_1), \{(0,1)\}\Big), \Big((t, \mathbb{P}_2), \{(0,0),(1,1)\}\Big), \Big((t, \mathbb{P}_3), \{(1,0)\}\Big) \right\}$$

is admitted by *col*. Simultaneously, $\mathbb{M}(t)$ is the only evaluated separator of t admitted by *col* and $\mathbb{M}(t) = \mathbb{M}_{col}(t)$.

Note that an evaluated path partition of t does not have to determine any colouring of G, nor G_t. But according to the following lemma it does determine the colouring of X_t.

Lemma 39. *Let t be a node of T and $\mathbb{M}(t)$ be an evaluated separator of t such that $\mathbb{M}(t)$ is admitted by at least one colouring. Then the colour of every vertex from X_t in every colouring c of G admitting $\mathbb{M}(t)$ is uniquely determined by $\mathbb{M}(t)$. Specifically, the colour of a vertex v from X_t is the colour corresponding to the only vector in the evaluation of the evaluated path-partition of $\{(\{v, v\}, \{v\})\}$ from $\mathbb{M}(t)$.*

Proof. First of all, for every $v \in X_t$ the set $\mathbb{P} = \{(\{v, v\}, \{v\})\}$ is a path-partition of t. Let $\mathbb{P}_{ev} = ((t, \mathbb{P}), Y)$ be the only evaluated path-partition of $\mathbb{P}$ from $\mathbb{M}(t)$. And let c be a colouring admitting $\mathbb{P}_{ev}$. Then the evaluation of $\mathbb{P}_{ev}$ has to be $M_c(t, \mathbb{P})$. Thus $M_c(t, \mathbb{P}) = Y$. But $M_c(t, \mathbb{P})$ contains only the parity vector of the colour of v because the only path-wise pair $(\{v, v\}, \{v\})$ from $\mathbb{P}$ is admitted only by the path composed of only the vector v. Therefore $Y = \{pv(v)\}$, but Y is constant, and so the colour of v has to be the same in every colouring admitting $\mathbb{M}(t)$. Thus, it is exactly the colour corresponding to the only parity vector from Y. □

This lemma implies that we do not need a colouring of X_t as a part of our "state". The last part is to define an evaluation of a set $N(t)$.

Definition 34. *Let t be a node of T. We call the set $\mathbb{N}(t)$ of some parity vectors of dimension k an* evaluation of processed paths of t.

Definition 35. *Let t be a node of T, and c a be a colouring of G, and $\mathbb{N}(t)$ be an evaluation of processed paths of t. We say that the colouring c* admits $\mathbb{N}(t)$ *if $N_c(t) = \mathbb{N}(t)$.*

Example 6. Consider a graph G with its nice tree decomposition as on Figure 3.5. Assume $k = 2$. Let *col* be a colouring of G given as $col(a) = 1$, $col(b) = 1$, $col(c) = 2$. Recall that $N_{col}(t)$ contains parity vectors of all paths in $G_t - X_t$ coloured by *col*. Since the only paths in $G_t - X_t$ are (a) and (b), it follows that $N_{col}(t) = \{(1,0); (0,1)\}$. Thus the set $\mathbb{N}(t) := \{(1,0); (0,1)\}$ is admitted by *col* (and it is the only one admitted by *col*).

Now we put this together to finally define the state of each node that we use in the dynamic programming.

Definition 36. *Let t be a node of T. We denote by $\mathbb{L}(t)$ the set of all pairs $(\mathbb{M}(t), \mathbb{N}(t))$ where $\mathbb{M}(t)$ is an evaluated separator of t and $\mathbb{N}(t)$ is an evaluation of processed paths of t such that there exists a colouring of G admitting both $\mathbb{M}(t)$ and $\mathbb{N}(t)$.*

Thus one pair in $\mathbb{L}(t)$ completely defines the state of the node t in the algorithm `verifycolouring` for at least one colouring.

Example 7. When we consider the situation on Figure 3.5, the set $\mathbb{M}(t)$ from Example 5, and the set $\mathbb{N}(t)$ from Example 6, we see that the pair $(\mathbb{M}(t), \mathbb{N}(t))$ is admitted by the colouring from Example 5. Thus this pair is in $\mathbb{L}(t)$.

Before we explain how to compute these sets, observe how it solves our problem.

Lemma 40. *Let r be a root of T. Then the graph G is parity colourable with k colours if and only if $\mathbb{L}(r)$ contains some pair $(\mathbb{M}(r), \mathbb{N}(r))$ such that $\mathbb{N}(r)$ does not contain the zero parity vector.*

Proof. The set $\mathbb{L}(r)$ contains all pairs $(\mathbb{M}(r), \mathbb{N}(r))$ admitted by some colouring. Since every colouring using k colours admits some such pair, $\mathbb{L}(r)$ contains all pairs $(\mathbb{M}_c(r), \mathbb{N}_c(r))$ for all such colourings. By Lemma 34, a colouring c of G is proper if and only if $N_c(r)$ does not contain the zero parity vector. Thus there exists a parity vertex colouring if and only if $\mathbb{L}(r)$ contains some pair $(\mathbb{M}(r), \mathbb{N}(r))$ such that $\mathbb{N}(r)$ does not contain the zero parity vector. □

We now describe the dynamic programming on Γ computing the sets $\mathbb{L}(t)$ for every node t. We proceed from leaves to root and we use the procedures for computing sets M, N from the algorithm `verifycolouring`.

1. **Leaf node.** Suppose that t is a leaf node. Thus X_t is empty. Hence there is no path-partition of t, nor an evaluated path-partition of t. Therefore $\mathbb{M}(t) = \emptyset$, and every colouring admits $\mathbb{M}(t)$.

   For every colouring c of G it holds that $N_c(t) = \emptyset$. Thus $\mathbb{N}(t) = \emptyset$. Observe that every colouring admits $\mathbb{N}(t)$. Therefore $\mathbb{L}(t) = \{(\emptyset, \emptyset)\}$.

2. **Forget node.** Suppose that t is a forget node with a son t'. We will show that $\mathbb{L}(t)$ contains exactly the pairs $(\mathbb{M}(t), \mathbb{N}(t))$ such that $(\mathbb{M}(t), \mathbb{N}(t))$ are computed from some $(\mathbb{M}(t'), \mathbb{N}(t')) \in \mathbb{L}(t')$ by the procedure for forget node in the algorithm `verifycolouring` (every set M and N of t is computed from $(\mathbb{M}(t'), \mathbb{N}(t'))$).

   Let $(\mathbb{M}(t), \mathbb{N}(t)) \in \mathbb{L}(t)$. There exists a colouring c of G admitting $(\mathbb{M}(t), \mathbb{N}(t))$. Thus $(\mathbb{M}(t), \mathbb{N}(t)) = (\mathbb{M}_c(t), \mathbb{N}_c(t))$ and also $(\mathbb{M}_c(t'), \mathbb{N}_c(t')) \in \mathbb{L}(t')$. Moreover, the algorithm `verifycolouring` implies that $(\mathbb{M}_c(t), \mathbb{N}_c(t))$ can be computed from $(\mathbb{M}_c(t'), \mathbb{N}_c(t'))$ and the colouring of $X_{t'}$. But according to Lemma 39, the colouring of X_t is determined by $\mathbb{M}(t')$. Thus we can compute $(\mathbb{M}_c(t), \mathbb{N}_c(t))$ just from $(\mathbb{M}_c(t'), \mathbb{N}_c(t'))$.

   On the other hand, let $(\mathbb{M}(t'), \mathbb{N}(t')) \in \mathbb{L}(t')$. There exists a colouring c of G admitting $(\mathbb{M}(t'), \mathbb{N}(t'))$. Thus $(\mathbb{M}(t'), \mathbb{N}(t')) = (\mathbb{M}_c(t'), \mathbb{N}_c(t'))$ and also $(\mathbb{M}_c(t), \mathbb{N}_c(t)) \in \mathbb{L}(t)$. By the same argument as before we see that $(\mathbb{M}_c(t), \mathbb{N}_c(t))$ can be computed from $(\mathbb{M}_c(t'), \mathbb{N}_c(t'))$ by the algorithm `verifycolouring`.

3. **Introduce edge node.** Suppose that t is an introduced edge node with a son t'. By the same argument as for the forget node, we see that $\mathbb{L}(t)$ contains exactly the pairs $(\mathbb{M}(t), \mathbb{N}(t))$ such that $(\mathbb{M}(t), \mathbb{N}(t))$ are computed from some $(\mathbb{M}(t'), \mathbb{N}(t')) \in \mathbb{L}(t')$ by the procedures for introduced edge node in the algorithm `verifycolouring`.

4. **Introduce vertex node.** Suppose that t is an introduced vertex node with a son t' such that the vertex v is introduced in t. We can compute the set $\mathbb{L}(t)$ similarly as before. The only difference is that $(\mathbb{M}(t'), \mathbb{N}(t'))$ from $\mathbb{L}(t')$ is not sufficient to compute some $(\mathbb{M}(t), \mathbb{N}(t))$ from $\mathbb{L}(t)$. We additionally need the colour of v. Thus $\mathbb{L}(t)$ contains exactly the pairs $(\mathbb{M}(t), \mathbb{N}(t))$ such that $(\mathbb{M}(t), \mathbb{N}(t))$ are computed from some $(\mathbb{M}(t'), \mathbb{N}(t')) \in \mathbb{L}(t')$ and some colour of v by the procedure for introduced vertex node in the algorithm `verifycolouring`.

5. **Join node.** Suppose that t is a join node with sons t_1, t_2. We will show that $\mathbb{L}(t)$ contains exactly the pairs $(\mathbb{M}(t), \mathbb{N}(t))$ such that $(\mathbb{M}(t), \mathbb{N}(t))$ are computed by the procedure for a join node in the algorithm `verifycolouring` from some $(\mathbb{M}(t_1), \mathbb{N}(t_1)) \in \mathbb{L}(t_1)$ and some $(\mathbb{M}(t_2), \mathbb{N}(t_2)) \in \mathbb{L}(t_2)$ such that the corresponding colours of X_t are the same (by Lemma 39 they are uniquely determined).

   Let $(\mathbb{M}(t), \mathbb{N}(t)) \in \mathbb{L}(t)$. There exists a colouring c of G_t admitting $(\mathbb{M}(t), \mathbb{N}(t))$. Thus $(\mathbb{M}(t), \mathbb{N}(t)) = (\mathbb{M}_c(t), \mathbb{N}_c(t))$ and also $(\mathbb{M}_c(t_1), \mathbb{N}_c(t_1)) \in \mathbb{L}(t_1)$ and $(\mathbb{M}_c(t_2), \mathbb{N}_c(t_2)) \in \mathbb{L}(t_2)$. Moreover, the algorithm `verifycolouring` implies that $(\mathbb{M}_c(t), \mathbb{N}_c(t))$ can be computed from $(\mathbb{M}_c(t_1), \mathbb{N}_c(t_1))$, $(\mathbb{M}_c(t_2), \mathbb{N}_c(t_2))$ and the colouring of X_t. But according to Lemma 39 this the colouring of X_t is determined by $\mathbb{M}_c(t_1)$. Thus we can compute $(\mathbb{M}_c(t), \mathbb{N}_c(t))$ from $(\mathbb{M}_c(t_1), \mathbb{N}_c(t_1))$ and $(\mathbb{M}_c(t_2), \mathbb{N}_c(t_2))$ without knowing the actual colouring c.

   On the other hand, let $(\mathbb{M}(t_1), \mathbb{N}(t_1)) \in \mathbb{L}(t_1)$ and $(\mathbb{M}(t_2), \mathbb{N}(t_2)) \in \mathbb{L}(t_2)$ be such that the corresponding colourings of X_t are the same. There exists a colouring c_1 of G_{t_1} admitting $(\mathbb{M}(t_1), \mathbb{N}(t_1))$ and a colouring c_2 of G_{t_2} admitting $(\mathbb{M}(t_2), \mathbb{N}(t_2))$. These colourings are the same on X_t. Moreover, the colourings c_1, c_2 can be combined together (because they are same on $V(G_{t_1}) \cap V(G_{t_2}) = X_t$) to obtain a colouring c of G_t. Thus $(\mathbb{M}(t_1), \mathbb{N}(t_1)) = (\mathbb{M}_c(t_1), \mathbb{N}_c(t_1))$ and $(\mathbb{M}(t_2), \mathbb{N}(t_2)) = (\mathbb{M}_c(t_2), \mathbb{N}_c(t_2))$. Then there exists $(\mathbb{M}_c(t), \mathbb{N}_c(t)) \in \mathbb{L}(t)$ and we can compute it by the procedure for a join node in the algorithm `verifycolouring` just from $(\mathbb{M}_c(t_1), \mathbb{N}_c(t_1))$ and $(\mathbb{M}_c(t_2), \mathbb{N}_c(t_2))$ (without knowing the colouring c).

Proof of Theorem 30. We are given a graph G and we want to check if there exists a parity vertex colouring c using k colours. Let n be the size of G and tw be the treewidth of G (we do not know it yet). We can find the treewidth of G and a tree decomposition of width tw in time $tw^{O(tw)} \cdot n$ by applying Theorem 29 for integers from 1 until we find a decomposition (thus, at most until tw). Theorem 29 does not give any bound on the number of nodes in the returned decomposition, but from the running time we know it is in $tw^{O(tw)} \cdot n$. Then by Lemma 26 we can find a nice tree decomposition $\Gamma = (T, \{X_i\}_{i \in V(T)})$ of width tw having $O(tw \cdot n)$ nodes in time $O(tw^2 \cdot tw^{O(tw)} \cdot n) = tw^{O(tw)} \cdot n$. Let r be the root of T.

Now for every node t of T we compute the set $\mathbb{L}(t)$ by the dynamic programming explained above. By Lemma 40, G has a parity vertex colouring if $\mathbb{L}(r)$ contains a pair $(\mathbb{M}(r), \mathbb{N}(r))$ such that $\mathbb{N}(r)$ does not contain the zero parity vector. We can easily check that. It remains to estimate the running time of the dynamic programming.

For each node t of T, there are at most $(tw+1)^{tw+1} \cdot (tw+2)^2$ path-partitions, and each evaluation of a path-partition and of a processed path can have at most

2^{2^k} distinct values. Hence for each node the number of distinct pairs $(\mathbb{M}(t), \mathbb{N}(t))$ and thus the size of $\mathbb{L}(t)$ is at most $(2^{2^k})^{1+tw+1^{tw+1}\cdot(tw+2)^2}$. In our dynamic approach we always generate $\mathbb{L}(t)$ from $\mathbb{L}$ of the sons of t and from all the possible colourings of at most one vertex . Each node has at most 2 sons, so to compute $\mathbb{L}(t)$ we go through at most $k\cdot(2^{2^k})^{2+2\cdot(tw+1)^{tw+1}\cdot(tw+2)^2}$ combinations of evaluations of its sons and a colour of some node. Possibly (in a join node) we need to verify that the combinations satisfy some condition. It can clearly be done polynomially with the size of evaluations, thus polynomially with $(2^{2^k})^{2+2\cdot tw+1^{tw+1}\cdot(tw+2)^2}$. For each such combination we apply a part of the algorithm `verifycolouring` that runs in time $tw^{O(tw)} \cdot 2^{O(k)}$. Thus the total running time for each node is definitely in $(2^{2^k})^{tw+1^{O(tw+1)}}$. Since there are $O(tw \cdot n)$ nodes, the total running of the algorithm is in $(2^{2^k})^{tw+1^{O(tw+1)}} \cdot n$ including the construction of the decomposition Γ. Therefore the problem ODDCOLOURING(G,k) is FPT with respect to the number of colours k and the treewidth of G.

□

4. Relations to other types of colourings

In this chapter we decribe several related colourings of graphs. We present several known results about them that are interesting in the context of the parity vertex colouring and we discuss if the results in this paper hold for the other colourings as well, or alternatively, if the algorithms we designed can be modified for the other colourings.

In this section it would be ambiguous to call the parity vertex colouring just as colouring. Thus for clarity, we always specify the type of each colouring.

4.1 Vertex colourings

We start with reminding the "standard" proper vertex colouring.

Definition 37. *Let G be a graph. A* proper vertex colouring *of G is a colouring of vertices of G such that no pair of adjacent vertices have the same colour. The* chromatic number *of G (denoted by $\chi(G)$) is the minimal number of colours in a proper vertex colouring of G.*

We now present two other colourings that are usually studied together with the parity vertex colouring.

Definition 38. *Let G be a graph. A* unique maximum colouring *of G is a colouring of vertices of G such that for every path P in G the maximal colour used on P occurs exactly once on P. The* unique maximum chromatic number *of G (denoted by $\chi_{um}(G)$) is the minimal number of colours in a unique maximum colouring of G.*

This colouring is alternatively known as vertex ranking.

Definition 39. *Let G be a graph. A* conflict free colouring *of G is a colouring of vertices of G, such that for every path P in G there exists a colour used on P that occurs exactly once on P. The* conflict free chromatic number *of G (denoted by $\chi_{cf}(G)$) is the minimal number of colours in a conflict free colouring of G.*

Let us first order these colourings in a chain of generality.

Lemma 41 ([6])**.** *Let G be a graph. Then every unique maximum colouring of G is a conflict free colouring. Every conflict free colouring is a parity vertex colouring. And every parity vertex colouring is a proper vertex colouring. Thus $\chi_{um}(G) \geq \chi_{cf}(G) \geq \chi_p(G) \geq \chi(G)$.*

Proof. Every unique maximum colouring of G contains for every path P of G a colour used once on P (the maximal colour). Thus it is a conflict free colouring.

Every conflict free colouring of G contains for every path P of G a colour used once (thus odd number of times) on P. Therefore the path P is not a parity path. Hence the colouring is a parity vertex colouring.

In every parity vertex colouring of G every path of length 2 (thus every pair of adjacent vertices) has to contain two different colours. Thus the colouring is a proper vertex colouring. □

Let us first briefly discuss that all of these chromatic numbers may differ. It is known that the conflict free chromatic number can be smaller than the unique maximum (e.g. Cheilaris et al. [6] have shown that $\chi_{cf}(B_7) \leq 6$ and that $\chi_{um}(B_7) = 7$). Since every tree is bipartite, Lemma 4 gives an example of a graph with the higher parity vertex chromatic number than the chromatic number. It remains to show that the conflict free chromatic number may differ from the parity vertex chromatic number. We show that it differs for B_4. By Lemma 4, $\chi_p(B_4) = 3$. Thus, it suffices to show that $\chi_{cf}(B_4) > 3$.

Lemma 42. $\chi_{cf}(B_4) = 4$.

Proof. Since B_4 contains P_7 as a subgraph, Lemma 1 together with Lemma 41 implies that $\chi_{cf}(B_4) \geq 3$. It remains to show that there is no conflict free colouring of B_4 using 3 colours.

Suppose that such colouring exists. Without any loss of generality, let 1 be a colour of the root r of the tree B_4. Let X be one of the B_3 subtrees connected to r. Since X is a copy of B_3, it contains P_5 as a subgraph. Thus by Lemma 1 and Lemma 41, X has at least three colours. Thus in our colouring it is coloured by exactly three colours. Let B be one of the branches of X that has the colour 1 used on it. If there were only two colours used on B, the path composed of B and r would use only 2 colours. That is not possible, thus there are 3 colours used on B. Similarly there exists a branch B' in the second B_3 subtree connected to r such that there are three colours used on B'. We can connect B', r, and B to obtain a path that has each colour used at least twice on it. Thus we obtained a contradiction and there is no conflict free colouring of B_4 using 3 colours. □

Lemma 41 implies that every upper bound on the parity vertex chromatic number is an upper bound on the chromatic number, and more interestingly every lower bound on the parity vertex chromatic number is a lower bound on the conflict free and the unique maximum chromatic numbers.

Therefore we immediately obtain a version of Theorem 12 and Theorem 13.

Corollary 43. *For every $n \geq 2$ and every subdivision B^* of B_n, $\chi_{cf}(B^*) > \sqrt{n}$.*

Corollary 44. *For every $n \geq 2$ and every binary tree B on n vertices, $\chi_{cf}(B) > \sqrt[3]{\log n}$.*

Corollaries 43 and 44 hold also for the unique maximum chromatic number, but for this chromatic number better bounds exist. Specifically, Cheilaris et al. [6] showed that the unique maximum chromatic number of a complete binary tree with d layers is d.

Lemma 41 additionally implies that every upper bound on the unique maximum or the conflict free chromatic number is also an upper bound on the parity vertex chromatic number. We use this and another known results to present two bounds improving Lemma 3 for certain classes of trees.

Theorem 45 ([6])**.** *For the sequence of complete binary trees, $\{B_i\}_{i=1}^{\inf}$, the limit of $\frac{i}{\chi_{cf}(B_i)}$ is at least* $\log 3$.

From this we immediately obtain the following corollary.

Corollary 46. *For the sequence of complete binary trees,* $\{B_i\}_{i=1}^{\inf}$*, the limit of* $\frac{i}{\chi_p(B_i)}$ *is at least* $\log 3$.

In other words, $\chi_p(B_i) \leq \frac{i}{log(3)}$ as i tends to infinity.

Another interesting upper bound on the parity vertex chromatic number was proved by Gregor and Škrekovski [10]. They proved a bound for binomial trees.

Theorem 47 ([10])**.** *For every* n *the binomial tree* Bi_n *of order* n *has a parity vertex colouring with* $\lceil \frac{2 \cdot n+3}{2} \rceil$ *colours. Thus* $\chi_p(Bi_i) \leq \lceil \frac{2 \cdot n+3}{2} \rceil$.

It is worth mentioning that Theorems 45 and 47 were proved by explicitly constructing the colourings.

4.1.1 Complexity of computing the unique maximum and the conflict free chromatic numbers

The problem of computing the unique maximum chromatic number of general graphs is known to be NP-complete (see e.g. Bodlaender et al. [2]). This implies that unlike parity chromatic colouring, checking if a colouring is proper can be done in polynomial time.

Generally, the problem of computing the unique maximum chromatic number has been already intensively studied and there are several efficient algorithms solving this problem for certain classes of graphs. For example, Bodlaender et al. [2] proved that there exists a polynomial algorithm finding the conflict free chromatic number for graphs with bounded treewidth.

On the other hand, even checking if a colouring is a conflict free colouring is known to be coNP-complete ([5]) just like the same problem for the parity vertex colouring. Thus it make sense to adjust the algorithms presented in Sections 2 and 3 to work with a conflict free colouring instead of a parity vertex colouring. The algorithm `FindColouring(`G`,`k`)` presented in Section 2.2 can be adjusted very easily. The only difference is in checking if a coloured subset of path-induced vertices is coloured properly. Instead of checking if some colour is used odd number of times on a given subset of vertices, we check if some colour is used exactly once.

It takes more work to change the other algorithms. The general idea is to use different vectors for paths than the parity vectors. These vectors of paths should remember for each colour if it is not used, used once, or used at least twice on the path. Then we can add vectors of two disjoint paths together in a natural way. We need to be more carefull to add vectors of two intersecting paths, but if we know the colours of intersecting vertices, it is easy to see how to do it. Also a vector of a "wrongly" coloured path is not the zero vector, but a vector where every colour is used at least twice or not at all. Hence we can work with these vectors similarly as we worked with parity vectors. If we were more precise, we believe we could prove the following conjecture using this approach.

Conjecture 2. *The problem of checking if a graph* G *has a conflict free colouring using* k *colours is fixed-parameter tractable with respect to* k *together with the treewidth of* G.

4.2 Parity edge colouring

In this section we describe the edge variant of parity colourings and compare known results with the vertex version.

Definition 40. *Let G be a graph. A* parity edge colouring *of G is a colouring of edges of G such that there exists no path P in G with even usage of every colour on the edges of P. The* parity edge chromatic number *of G (denoted by $\chi'_p(G)$) is the minimal number of colours in a parity edge colouring of G.*

It is easy to see that a parity vertex colouring of P_n corresponds to a parity edge colouring of P_{n+1}. Thus by Lemma 1 we get the following, already known, corollary.

Corollary 48 ([4])**.** *For every $n \geq 2$, $\chi'_p(P_n) = \lfloor \log(n-1) \rfloor + 1$.*

But that is the end of similarities. Even for cycles the vertex and the edge parity chromatic numbers are different, as the Lemma 2 and the following result of Bunde et al. [4] shows.

Lemma 49 ([4])**.** *For every even n, $\chi'_p(C_n) = \lceil \log(n) \rceil$. For every odd n, $\chi'_p(C_n) = \lceil \log n \rceil + 1$.*

More interestingly, a parity edge colouring of trees has an interesting relation to subgraphs of Q_n.

Theorem 50 ([4])**.** *A tree T embeds in the k-dimensional hypercube Q_k if and only if $\chi'_p(T) \leq k$.*

This yields an interesting lower bound on the parity edge chromatic number.

Corollary 51 ([4])**.** *If G is a connected graph on n vertices, then $\chi'_p(G) \geq \lceil \log n \rceil$.*

When we compare it with Lemma 3, we get an interesting inequality between the two parity colourings of trees.

Corollary 52. *For every tree T on n vertices, $\chi_p(T) \leq \chi'_p(T) + 1$.*

Proof. By Lemma 3 we have $\chi_p(T) \leq \lfloor \log n \rfloor + 1$. And by Corollary 51 we have $\chi'_p(T) \geq \lceil \log n \rceil$. Since $\lfloor \log n \rfloor + 1 = \lceil \log(n+1) \rceil$, we see that

$$\chi_p(T) \leq \lfloor \log n \rfloor + 1 = \lceil \log(n+1) \rceil \leq \lceil \log n \rceil + 1 \leq \chi'_p(T) + 1.$$

□

Conclusion

In this thesis we continued the study of the parity vertex colouring. We have defined a new class of coloured graphs, safflowers. With safflowers we have improved the lower bound on the parity vertex chromatic number of subdivisions of complete binary trees by showing that for every subdivison B^* of B_n ($n > 1$) it holds that $\chi_p(B^*) > \sqrt{n}$. We believe that every properly coloured subdivision of a complete binary tree contains even larger safflower and consequently the lower bound can be further improved. We have also used this bound to show that for every binary tree B with n vertices ($n > 1$) it holds that $\chi_p(B) > \sqrt[3]{\log n}$.

We find it striking that we know the exact parity vertex chromatic number only for the simplest classes of graphs, like paths and cycles. Even for certain classes of trees the best upper bounds (Lemma 3, Corollary 46, and Theorem 47) are asymptotically quadratically higher than the best lower bounds (Theorem 12 and Theorem 13). Therefore it would be interesting to either find better colourings or prove that there are none.

From the computational point of view we have designed algorithms for trees and for general graphs that can be easily implemented and used for finding the parity chromatic number of small graphs. Our result that the computation of the parity vertex chromatic number is FPT with respect to the treewidth of the input graph and the actual chromatic number has probably only theoretical impact since even for small parameters the running time seems to be too high. What we consider to be an interesting open problem is whether there exist some approximation algorithms computing the parity vertex chromatic number that run in some reasonable, preferably polynomial, time.

We have discussed the possibility of applying the presented results on the other colourings in Section 4. Apart from Conjecture 2, we find interesting the question how much smaller the parity vertex chromatic number can be compared to the conflict free chromatic number of the same graph. We have shown that $\chi_p(B_4) + 1 = \chi_{cf}(B_4)$, but we do not even know if there is a graph G such that $\chi_p(G) + 2 \geq \chi_{cf}(G)$. Moreover, every presented upper bound on the parity chromatic number (Lemma 3, Corollary 46, and Theorem 47) was proved by finding a conflict free colouring. Thus, there is so far no example how a parity vertex colouring that is not a conflict free colouring should be designed.

Another interesting question is whether there is some deeper relation between the edge and the vertex parity colourings. For example, for paths the edge colouring is equivalent to the vertex colouring of the corresponding line graph. Unfortunately, this approach fails even for cycles, although the line graph is the same as the original graph, because unlike vertices we can not put all edges on a single path. Maybe there is some other transformation of graphs that maps one of these problems to the second one, or some other relation can be found. This would be interesting since the edge variant has been more studied and especially for trees there are better known results.

Bibliography

[1] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 12 1996.

[2] H. L. Bodlaender, J. S. Deogun, K. Jansen, T. Kloks, D. Kratsch, H. Müller, and Z. Tuza. Rankings of graphs. *SIAM Journal on Discrete Mathematics*, 11:168–181, 1998.

[3] P. Borowiecki, K. Budajová, S. Jendrol', and S. Krajči. Parity vertex colouring of graphs. *Discussiones Mathematicae Graph Theory*, 31:183–195, 2011.

[4] D. P. Bunde, K. Milans, D. B. West, and H. Wu. Parity and strong parity edge-coloring of graphs. *Congressus Numerantium*, 187:193–213, 2007.

[5] P. Cheilaris and G. Tóth. Graph unique-maximum and conflict-free colorings. *Journal of Discrete Algorithms*, 9:241–251, 2011.

[6] P. Cheilaris, B. Keszegh, and D. Pálvölgyi. Unique-maximum and conflict-free colouring for hypergraphs and tree graphs. *SIAM Journal on Discrete Mathematics*, 27:1775–1787, 2013.

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

[8] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, London, 2015.

[9] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness i: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

[10] P. Gregor and R. Škrekovski. Parity vertex colorings of binomial trees. *Discussiones Mathematicae Graph Theory*, 32:177–180, 2012.

[11] H.-C. Hsu and G. J. Chang. Parity and strong parity edge-colorings of graphs. *Journal of Combinatorial Optimization*, 24(4):427–436, 2012.

[12] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Array, Trees, Hypercubes.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[13] J. W. H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Analysis and Applications*, 11:134–172, 01 1990.

[14] A. Sen, H. Deng, and S. Guha. On a graph partition problem with application to vlsi layout. *Information Processing Letters*, 43(2):87–94, 1992.

List of Figures