University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Arkadiusz Socała

Lower Bounds Under Strong Complexity Assumptions

PhD dissertation

Supervisor
dr hab. Łukasz Kowalik
Institute of Informatics
University of Warsaw

May 2017

Author's declaration:
I hereby declare that this dissertation is my own work.

May 20, 2017 .....

Arkadiusz Socała

Supervisor's declaration:
The dissertation is ready to be reviewed

May 20, 2017 .....

dr hab. Łukasz Kowalik

Abstract

This work is devoted to lower bounds on running time under strong complexity assumptions. We prove that under the Exponential Time Hypothesis there is no algorithm working in time

- $2^{o(n \log n)}$ (times a polynomial in the bit size) for Channel Assignment,
- $2^{o(n\sqrt{\log n})}$ for Subgraph Isomorphism,
- $2^{o(n^{3/2})}$ for Rainbow k-Coloring (for any $k \geq 2$),
- $f(b) \cdot 2^{o(\log b) \cdot n}$ for $(a{:}b)$-coloring (for any computable $f(b)$),
- $O^*(2^{o(d \log d)})$ for Minimax Approval Voting.

This in particular exclude the existence of algorithms solving these problems in time $O^*(c^n)$ (or $O^*(c^d)$ in the case of Minimax Approval Voting) for any constant c. Moreover in the cases of Channel Assignment, $(a{:}b)$-coloring and Minimax Approval Voting our lower bounds are tight, showing that currently known algorithms are optimal (unless ETH fails).

We also give a lower bound of a different flavor. We show that improving over the best known algorithm for 4-OPT Detection (a problem related to Traveling Salesman Problem), would imply an improvement for All Pairs Shortest Paths.

Keywords: Lower Bounds, Fine Grained Complexity, Exponential Time Hypothesis.
ACM Classification: F.2.2.

Contents

Chapter 1

Introduction

We would like to be able to quickly solve combinatorial problems that appear in theory or in the real life. However for many of them we do not know efficient algorithms. Therefore the question arises: is it because it is generally impossible or maybe just we do not know appropriate tools yet? In order to answer this question we need to understand the reason of the hardness of the particular problem. One approach to investigate it is to reduce another, perhaps more familiar, problem to our target problem. Such a reduction shows that the target problem has enough expressive power to capture the hardness of the source problem. If the reduced instance is concise enough we can usually derive interesting conclusions about the dependency between the time complexities of these two problems.

This approach is commonly used for NP-hard problems. For example, one can reduce an n-variable instance of 3-CNF-SAT to an equivalent $n^{O(1)}$-vertex instance of 3-Coloring and the reduction algorithm works in polynomial time. There are two ways of looking at the consequences of this reduction. First, it means that if P $\neq$ NP then 3-Coloring has no polynomial time algorithm. However, even if we do not believe that P $\neq$ NP the reduction carries an important message: instead of working on a polynomial time algorithm for 3-Coloring one should rather focus on the structurally simpler 3-CNF-SAT.

In this thesis we investigate reductions that are more constrained than the classic polynomial reductions. For example one can observe that the classic reduction from 3-CNF-SAT to 3-Coloring transforms an input formula φ into an $O(|\varphi|)$-vertex graph, i.e., the size of the output instance is not only polynomial but even linear. This means that a $2^{o(n)}$-algorithm for 3-Coloring implies a $2^{o(|\varphi|)}$ algorithm for 3-CNF-SAT. Again we can interpret this in two ways. First, it is a hint for the researchers to study rather 3-CNF-SAT directly than try to improve the algorithm for 3-Coloring, that captures the full power of 3-CNF-SAT in more convoluted way. On the other hand, if one believes that no $2^{o(|\varphi|)}$ algorithm for 3-CNF-SAT exists it means there is no hope for $2^{o(n)}$ algorithm for 3-Coloring. This belief can be captured as a hypothesis, which strengthens P $\neq$ NP, and says that 3-CNF-SAT has no $2^{o(|\varphi|)}$-time algorithm. Actually, Impagliazzo, Paturi and Zane [76, 77] show that it is equivalent to assume a slightly weaker claim as follows.

Hypothesis 1 (Exponential Time Hypothesis (ETH) [76])**.** *There exists a constant* $c > 0$*, such that there is no deterministic algorithm solving* 3-CNF-SAT *in time* $O^*(2^{cn})$*.*[1]

By now ETH became one of standard assumptions for proving lower bounds (see [102] for a survey).

As discussed above, a linear reduction from 3-CNF-SAT to 3-Coloring shows that the known $O(1.33^n)$-time algorithm for 3-Coloring [10] is essentially optimal (up to constants). The same situation holds for many NP-complete graph problems, i.e., we know algorithms running in time $2^{O(n)}$, and this is tight under ETH. This is obviously the case for problems asking for a set of vertices, like Clique or Vertex Cover, or more generally, for problems which admit polynomially (or even subexponentially) checkable $O(n)$-bit certificates. But there are $2^{O(n)}$-time algorithms also for some problems for which such certificates are not known, including e.g., Hamiltonicity [71] and Graph Coloring [90]. However, for some problems $2^{O(n)}$ algorithms are not known, for example the Channel Assignment problem, which is a generalization of Graph Coloring, where every edge uv of the graph comes with integer weight w, which means that the colors of u and v must differ by at least w. The best known algorithm for Channel Assignment works in $2^{O(n \log n)}$ time. How can we prove it is tight?

For example this thesis contains a reduction from 3-CNF-SAT to Channel Assignment which transforms an instance of 3-CNF-SAT with n variables (and $O(n)$ clauses, which will be explained later) into an instance of Channel Assignment of size $O(n/\log n)$. This compression is an important feature of our reduction and it is different than in the case of many classic reductions for showing NP-hardness. As a corollary, a hypothetical $2^{o(n \log n)}$-time algorithm for Channel Assignment would imply a $2^{o(n)}$-time algorithm for 3-CNF-SAT. If we assume the Exponential Time Hypothesis (ETH) then we obtain that Channel Assignment cannot be solved in time $2^{o(n \log n)}$ and the currently known algorithm is essentially optimal. It is worth to note that even though our reduction works in polynomial time it could work as well in arbitrarily large time complexity that is in $2^{o(n)}$ and still composed with a $2^{o(n \log n)}$-time algorithm for Channel Assignment it would solve 3-CNF-SAT in time $2^{o(n)}$.

In this thesis we mostly focus on such instance-compressing reductions which exclude $2^{O(n)}$ algorithms. By studying these dependencies we try to discover the underlying structure of hardness among NP-complete problems. This young subfield, living on the border of algorithmics and complexity theory, is known under the name of *Fine Grained Complexity*.

Let us mention also a stronger version of ETH called Strong Exponential Time Hypothesis (SETH) [76, 77] that says that for every $\varepsilon < 1$ there exists $k > 3$ such that k-CNF-SAT cannot be solved in time $O^*(2^{\epsilon n})$. Indeed, an algorithm refuting even this stronger version would be a breakthrough. SETH can be useful for example if one cares about the multiplicative constant in the exponent, i.e., if one's goal is to

[1] The O^* notation suppresses factors polynomial in the input size.

prove a lower bound for some problem that would be of the form $\Omega^*(2^{cn})$ for some particular value of $c > 0$.

Although we will not spend much time on the polynomial-time problems in this dissertation it is worth to mention that the landscape of the fine grained complexity for the problems that are solvable in polynomial time is also very interesting. We have three main assumptions that are used for showing polynomial-time lower bounds:

- the already mentioned Strong Exponential Time Hypothesis (SETH),
- APSP assumption, that states that All Pairs Shortest Paths cannot be solved in time $O(n^{3-\delta}(\log M)^{O(1)})$ for any $\delta > 0$, and
- 3SUM assumption, that states that 3SUM cannot be solved in time $O(n^{2-\delta})$ for any $\delta > 0$.

One may be surprised by seeing SETH on this list as the statement of SETH regards exponential time complexity of k-CNF-SAT. However there is a reduction from k-CNF-SAT to Orthogonal Vectors that gives a polynomial-time lower bound for Orthogonal Vectors under the assumption of SETH [131]. In consequence, further reductions can possibly start from Orthogonal Vectors instead of starting directly from k-CNF-SAT. A famous result from 2014 by Backurs and Indyk says that Edit Distance cannot be solved in time $O(n^{2-\delta})$ for any $\delta > 0$ unless SETH fails [9]. The second assumption is connected to the class of polynomial-time solvable problems called a class of subcubic equivalence to All Pairs Shortest Paths (APSP) [132, 3]. In this class either each of the problems has a $O(n^{3-\delta}(\log M)^{O(1)})$-time algorithm for some $\delta > 0$ depending on the problem, or none of them has. A notable member of this class of problems is Negative Edge-Weighted Triangle [132] because it is very convenient starting point for the further reductions. Also the last of these three assumptions implies a number of polynomial-time lower bounds [61]. The most natural question is about possible dependencies between these three assumptions. Does one of the assumptions imply one of the others? In 2016 Carmosino, Gao, Impagliazzo, Mihajlin, Paturi and Schneider showed that under NSETH (co-nondeterministic version of SETH) such reductions from k-CNF-SAT to APSP or to 3SUM cannot exist [20]. While NSETH may or may not be true, this is certainly an obstacle showing that proving these two dependencies (if they do exist) can be hard. Another approach to connect somehow these three assumptions is to find a problem such that if *any* of the the (preferably) three assumptions is true then this intermediate problem is also hard. An example of such a problem that merges two different assumptions, namely APSP and 3SUM, is Zero Weight Triangle [120, 132] and for merging all the three assumptions we have Single Source Flow and Triangle Collection [4].

In this dissertation we present a few examples of ETH based lower bounds and also one example of a reduction from All Pairs Shortest Paths to a polynomial-time problem.

1.1 Organization of the dissertation

In Section 1.2 of the introduction we introduce the problems considered in this dissertation, discuss the related work and state our results. In Chapter 2 we present the common preliminaries and notation. Each of the chapters 3-8 corresponds to one of the considered problems.

1.2 Considered problems

1.2.1 Channel Assignment

Assume that we are given a symmetric weight function $w : V^2 \to \mathbb{N}$ (we assume that $0 \in \mathbb{N}$). The elements of V will be called vertices (as w induces a graph on the vertex set V with edges corresponding to positive values of w). We say that w is ℓ-bounded when for every $x, y \in V$ we have $w(x, y) \leq \ell$. An assignment $c : V \to \mathbb{Z}$ is called *proper* when for each pair of vertices x, y we have $|c(x) - c(y)| \geq w(x, y)$. The number $(\max_{v \in V} c(v) - \min_{v \in V} c(v) + 1)$ is called the *span* of c.

Channel Assignment
Input: $w : V^2 \to \mathbb{N}$, $s \in \mathbb{N}$
Question: Is there a proper assignment of span at most s?

In the optimization version of the problem the goal is to find an assignment of minimum span.

Note that the special case when w is 1-bounded corresponds to the classical graph coloring problem. It is therefore natural to associate the instance of the channel assignment problem with an edge-weighted graph $G = (V, E)$ where $E = \{uv : w(u, v) > 0\}$ with edge weights $w_E : E \to \mathbb{N}$ such that $w_E(xy) = w(x, y)$ for every $xy \in E$ (in what follows we abuse the notation slightly and use the same letter w for both the function defined on V^2 and E). The minimum span is called also the span of (G, w) and denoted by $\mathrm{span}(G, w)$.

It is interesting to realize the place of Channel Assignment in a hierarchy of constraint satisfaction problems. We have already seen that it is a generalization of the classical graph coloring. It is also a special case of the constraint satisfaction problem (CSP). In CSP, we are given a vertex set V, a constraint set $\mathcal{C}$ and a number of colors d. Each constraint is a set of pairs of the form (v, t) where $v \in V$ and $t \in \{1, \ldots, d\}$. An assignment $c : V \to \{1, \ldots, d\}$ is *proper* if every constraint $A \in \mathcal{C}$ is satisfied, i.e., there exists $(v, t) \in A$ such that $c(v) \neq t$. The goal is to determine whether there is a proper assignment. Note that Channel Assignment corresponds to CSP where d is equal to the maximum allowed span and every edge uv of weight $w(uv)$ in the instance of Channel Assignment corresponds to the set of all constraints of the form $\{(u, t_1), (v, t_2)\}$ where $|t_1 - t_2| < w(uv)$.

In the general (unbounded) case of Channel Assignment the best known algorithm runs in $O^*(n!)$ time, where n is the number of the vertices (see McDiarmid [114]). However, there has been some progress on the ℓ-bounded variant.

McDiarmid [114] came up with an $O^*((2\ell+1)^n)$-time algorithm which has been next improved by Král [85] to $O^*((\ell+2)^n)$, further to $O^*((\ell+1)^n)$ by Cygan and Kowalik [42] and to $O^*((2\sqrt{\ell+1})^n)$ by Kowalik and Socała [84]. These are all dynamic programming (and hence exponential space) algorithms. The last but one applies the fast zeta transform to get a minor speed-up and the last one uses the meet-in-the-middle approach. Interestingly, all these works show also algorithms which *count* all proper assignments of span at most s within the same running time (up to polynomial factors) as the decision algorithm.

Since graph coloring is solvable in time $O^*(2^n)$ [12] it is natural to ask whether Channel Assignment is solvable in time $O^*(c^n)$, for some constant c. It was an open problem (see [85, 42, 75]) to find such a $O(c^n)$-time algorithm for c independent of ℓ or prove that it does not exist under a reasonable complexity assumption. The problem mentioned above becomes even more interesting when we realize that under ETH, CSP does not have a $O^*(c^n)$-time algorithm for a constant c independent of d, as proved by Traxler [128].

Our Results. The main result contained in Chapter 3 is a proof that Channel Assignment does not admit a $O(c^n)$-time algorithm for any constant c under the ETH assumption. By applying a sequence of reductions (see Figure 1.1) starting in 3-CNF-SAT and ending in Channel Assignment we were able to solve the mentioned open problem regarding the hypothetical $O^*(c^n)$-time algorithm solving Channel Assignment for any constant c and to show the following theorem.

Theorem 1. *Unless ETH fails, there is no algorithm solving* Channel Assignment *in time* $2^{o(n \log n)} \cdot r^{O(1)}$ *where* n *is the number of the vertices and* r *is the bit size of the instance.*

Recall that the currently best known algorithm works in time $O^*(n!) = 2^{O(n \log n)}$, so our lower bound is tight.

In order to get the lower bound we show that in polynomial time one can transform an instance of 3-CNF-SAT with n variables into an instance of our target problem with $O\left(\frac{n}{\log n}\right)$ vertices. Then a $2^{o(n \log n)}$-time algorithm for our target problem would imply a $2^{o(n)}$-time algorithm for 3-CNF-SAT which contradicts the ETH. However such reductions which compress the size of the instance from $O(n)$ to e.g. $O\left(\frac{n}{\log n}\right)$ are very rare (for more examples of sublinear reductions see: [47], [101], [103]). As shown in Figure 1.1 we do this for the problem Equal Weight Matchings defined as follows:

Equal Weight Matchings
Input: Two complete weighted bipartite graphs $G_1 = (V_1 \cup W_1, E, w_1)$ and $G_2 = (V_2 \cup W_2, E, w_2)$ such that $|V_1| = |W_1|$ and $|V_2| = |W_2|$. The weight functions w_1, w_2 have nonnegative integer values.
Question: Are there two perfect matchings M_1 in G_1 and M_2 in G_2 such that $w_1(M_1) = w_2(M_2)$?

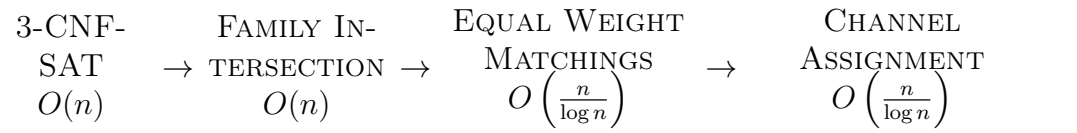

Figure 1.1: The sequence of the used reductions and the size of the instance. The compression follows between Family Intersection and Equal Weight Matchings. While the definition of Family Intersection is rather technical the Equal Weight Matchings problem is quite natural and it can be used as a generic problem without $2^{o(n \log n)}$-time algorithm.

Theorem 2. *Unless ETH fails, there is no algorithm solving* Equal Weight Matchings *in time* $2^{o(n \log n)} \cdot r^{O(1)}$ *where* n *is the total number of vertices, and* r *is the bit size of the input.*

Note that in order to show that a new problem P does not admit a $2^{o(n \log n)}$-time algorithm it suffices to give a reduction that preserves the size of the instance (up to the big-O notation) from Equal Weight Matchings to P. We showed such a reduction for Channel Assignment and we hope that the same thing can also be done for other problems.

1.2.2 Subgraph Isomorphism

Perhaps the most basic relation between graphs is that of being a subgraph. We say that G is a subgraph of H if one can remove some edges and vertices of H, so that what remains is isomorphic to G. Formally, the question of one graph being a subgraph of another is the base of the Subgraph Isomorphism problem.

Subgraph Isomorphism
Input: Undirected graphs G, H.
Question: Is G a subgraph of H, i.e., does there exist an injective function $g : V(G) \to V(H)$, such that for each edge $uv \in E(G)$ we have $g(u)g(v) \in E(H)$?

Subgraph Isomorphism is an important and very general question, having the form of a pattern matching – we will call G the *pattern graph* and H the *host graph*. Observe that several flagship graph problems can be viewed as instances of Subgraph Isomorphism:

- Hamiltonian: is C_n (a cycle with n vertices) a subgraph of G?
- Clique: is K_k a subgraph of G?
- 3-Coloring: is G a subgraph of $K_{n,n,n}$, a complete tripartite graph with n vertices in each of its three independent sets?
- VertexCover: is G a subgraph of H, H being a full join between a clique of size k and an independent set of size $n - k$?

One can continue showing the richness of Subgraph Isomorphism by simple linear reductions from Bandwidth, Set Packing and several other problems.

All of the mentioned problems are NP-complete, and the best known algorithms for all the listed special cases work in exponential time. In fact, all those problems are well-studied from the exact exponential algorithms perspective [10, 11, 12, 14, 46], where the goal is to obtain an algorithm of running time $O(c^n)$ for smallest possible value of c. Furthermore, the Subgraph Isomorphism problem was very extensively studied from the viewpoint of fixed parameter tractability[2], see [112] for a discussion of 19 different parametrizations. All the mentioned special cases of Subgraph Isomorphism admit $O(c^n)$ time algorithms, by using either branching, inclusion-exclusion principle or dynamic programming. On the other hand, a simple exhaustive search for the Subgraph Isomorphism problem – numerating all possible mappings from the pattern graph to the host graph – runs in $2^{O(n \log n)}$ time, where n is the total number of vertices of the host graph and pattern graph.

Therefore, a natural question is whether Subgraph Isomorphism admits an $O(c^n)$ time algorithm. This was repeatedly posed as an open problem [1, 5, 55, 56, 75]. In particular, Fomin and Kratsch in their monograph [59] put the existence of $O(c^n)$ time algorithm for Subgraph Isomorphism among the few questions in the open problems section.

Our results and techniques The first result of Chapter 4 is a self-contained reduction which transforms a 3-CNF-SAT formula into a subexponential number of sublinear instances of the Subgraph Isomorphism problem. It results in the following theorem.

Theorem 3. *There is no algorithm which solves* Subgraph Isomorphism *in time* $2^{o\left(n\sqrt{\log n}\right)}$*, unless the Exponential Time Hypothesis fails.*

Graph Homomorphism has a similar definition to Subgraph Isomorphism, except that the mapping is not constrained to be injective (i.e., in a homomorphism many vertices of the pattern graph may be mapped to the same vertex of the host graph). One could think that Graph Homomorphism is a harder problem than Subgraph Isomorphism. For example Amini, Fomin and Saurabh [5] and Curticapean, Dell and Marx [37] have shown that counting subgraphs can be reduced to counting homomorphisms.

Our second result is a simple reduction, which given an instance of Graph Homomorphism produces a single exponential number of instances of Subgraph Isomorphism. Even though from the perspective of polynomial time algorithms such a reduction gives no implication in terms of which problem is harder, in our setting it is enough to obtain a lower bound for Subgraph Isomorphism. The proof of the following theorem is based on a simple scheme of guessing preimage sizes, an idea used also in the proof of Theorem 3.

[2]For more on FPT algorithms see the textbook of Cygan et al. [41].

Theorem 4. *Given an instance (G, H) of* Graph Homomorphism *one can in $O(2^n n^{O(1)})$ time create 2^n instances of* Subgraph Isomorphism *with n vertices, where $n = |V(G)| + |V(H)|$, such that (G, H) is a yes-instance iff at least one of the created instances of* Subgraph Isomorphism *is a yes-instance.*

The theorem above becomes particularly interesting in connection with the following theorem of Fomin, Golovnev, Kulikov and Mihajlin [57] (published finally in the merged article of Cygan, Fomin, Golovnev, Kulikov, Mihajlin, Pachocki and Socała [38, 40]).

Theorem 5 ([57, 38])**.** *Let G be an n-vertex graph and H be an $h := h(n)$-vertex graph. Unless ETH fails, for any constant $D \geq 1$ there exists a constant $c = c(D) > 0$ such that for any function $3 \leq h(n) \leq n^D$, there is no $O\left(h^{cn}\right)$ time algorithm deciding whether there is a homomorphism from G to H.*

Theorem 4 combined with Theorem 5 implies a tight lower bound for Subgraph Isomorphism.

Theorem 6. *There is no algorithm which solves* Subgraph Isomorphism *in time $2^{o(n \log n)}$, unless the Exponential Time Hypothesis fails.*

1.2.3 Rainbow k-Coloring

Consider an undirected graph $G = (V, E)$ and an arbitrary function $c : E \to \{1, ..., k\}$ called coloring. A path with all edges of different colors is called a *rainbow path*. We say that c is a *rainbow coloring* if every pair of vertices is connected by a rainbow path. A minimum such k, called the *rainbow connection number* can be viewed as yet another measure of graph connectivity. The concept of rainbow coloring was introduced by Chartrand, Johns, McKeon, and Zhang [28] in 2008, while also featured in an earlier book of Chartrand and Zhang [29]. Chakraborty, Fischer, Matsliah, and Yuster [22] describe an interesting application of rainbow coloring in telecommunications. The problem is intensively studied from the combinatorial perspective, with over 100 papers published by now (see the survey of Li, Shi, and Sun [95] for an overview). In this dissertation we focus on the computational complexity of the following decision problem.

Rainbow k-Coloring
Input: $G = (V, E)$, k
Question: Does G have the rainbow connection number at most k?

It was conjectured by Caro, Lev, Roditty, Tuza, and Yuster [21] that the Rainbow k-Coloring problem is NP-complete for $k = 2$. This conjecture was confirmed by Chakraborty *et al.* [22]. Ananth, Nasre, and Sarpatwar [6] noticed that the proof of Chakraborty *et al.* in fact proves NP-completeness for every even $k > 1$, and complemented this by showing NP-completeness of the odd cases as well. An alternative hardness proof for every $k > 1$ was provided by Le and Tuza [91]. For complexity results on restricted graph classes, see e.g., [25, 26, 27, 51].

Unfortunately it seems that the best known worst-case running time bound for Rainbow k-Coloring is $k^m 2^k n^{O(1)}$, where m is the number of edges, which is obtained by checking each of the k^m colorings by a simple $2^k n^{O(1)}$-time dynamic programming algorithm [129]. Even in the simplest variant of just two colors, i.e., $k = 2$, this algorithm takes $2^{O(n^2)}$ time if the input graph is dense. Note that this algorithm is much slower than algorithms for many NP-complete problems such as Hamiltonicity, Subgraph Isomorphism or Channel Assignment.

Main Result. Our main result concerning rainbow coloring is the following theorem.

Theorem 7. *For any $k \geq 2$,* Rainbow k-Coloring *can be solved neither in $2^{o(n^{3/2})}$ nor $2^{o(m/\log m)}$ time where n and m are the number of vertices and edges respectively, unless ETH fails.*

Hence, this is an NP-complete graph problem which does not admit a $2^{o(n^c)}$-time algorithm (under reasonable complexity assumptions), for a constant $c > 1$. Such lower bounds are fairly rare in the literature.

In Chapter 5 we also study a natural generalized problem, called Subset Rainbow k-Coloring, introduced by Chakraborty *et al.* [22] as a natural intermediate step in reductions from 3-CNF-SAT to Rainbow k-Coloring. In Subset Rainbow k-Coloring, we are given a connected graph G, and a set of pairs of vertices $S \subseteq \binom{V(G)}{2}$. Elements of S are called *requests*. For a given coloring of $E(G)$ we say that a request $\{u, v\}$ is *satisfied* if u and v are connected by a rainbow path. The goal in Subset Rainbow k-Coloring is to determine whether there is a k-coloring of $E(G)$ such that every pair in S is satisfied. Theorem 7 implies that Subset Rainbow k-Coloring admits no algorithm running in time $2^{o(n^{3/2})}$, under ETH. We show also two more lower bounds, as follows.

Theorem 8. *For any $k \geq 2$,* Subset Rainbow k-Coloring *can be solved neither in time $2^{o(n^{3/2})}$, nor in time $2^{o(m)}$, nor in time $2^{o(s)}$ where n is the number of vertices, m is the number of edges, and s is the number of requests, unless ETH fails.*

An interesting feature here is that for $k = 2$ the $2^{o(m)}$ and $2^{o(s)}$ bounds are *tight* up to a polynomial factor (a $2^m n^{O(1)}$ algorithm is immediate, and a $2^{|S|} n^{O(1)}$-time algorithm is discussed in the paper of Kowalik, Lauri and Socała [83]). Moreover for every integer $k \geq 3$, Subset Rainbow k-Coloring parametrized by $|S|$ is fixed parameter tractable and it has an algorithm running in time $|S|^{O(|S|)} n^{O(1)}$ [83]. This shows that our lower bound in terms of s for Subset Rainbow k-Coloring is nearly tight.

1.2.4 Multicoloring and low degree monomial testing

The complexity of determining the chromatic number of a graph is undoubtedly among the most intensively studied computational problems. Countless variants and generalizations of graph colorings have been introduced and investigated. Here, we focus on *multicolorings*, also known as $(a{:}b)$-colorings. In this setting, we are given a graph G, a palette of a colors, and a number $b \leq a$. An *$(a{:}b)$-coloring* of G is

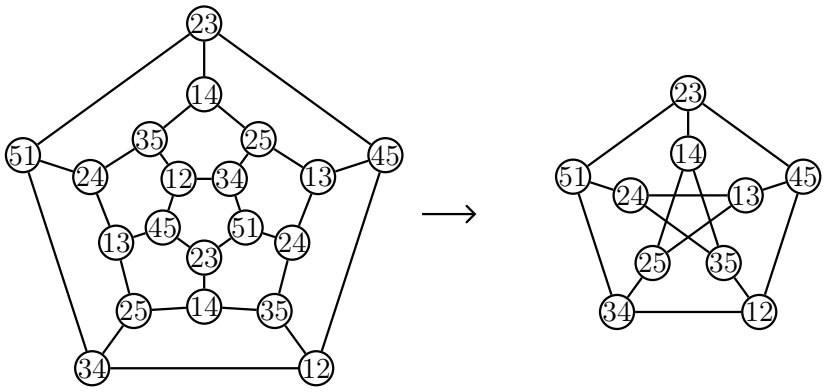

Figure 1.2: A (5:2)-coloring of the dodecahedron (left) which can be seen as a homomorphism to $KG_{5,2}$ (the Petersen graph, right). The homomorphism is given by identifying the pairs of opposite vertices in the corresponding regular solid.

any assignment of b distinct colors to each vertex so that adjacent vertices receive disjoint subsets of colors. The $(a{:}b)$-COLORING problem asks whether G admits an $(a{:}b)$-coloring. For $b = 1$ we obtain the classic graph coloring problem. The smallest a for which an $(a{:}b)$-coloring exists is called the *b-fold chromatic number*, denoted by $\chi_b(G)$.

The motivation behind $(a{:}b)$-colorings can be perhaps best explained by showing the connection with the *fractional chromatic number*. For a graph G, it is denoted as $\chi_f(G)$ and defined as the optimum value of the following natural linear programming relaxation of the problem of computing the chromatic number of G, expressed as finding a cover of the vertex set using the minimum possible number of independent sets:

$$\text{minimize} \sum_{I \in I(G)} x_I$$

$$\forall_{v \in V(G)} \sum_{v \in I \in I(G)} x_I \geq 1$$

$$\forall_{I \in I(G)}\ x_I \geq 0.$$

It can be easily seen that by relaxing the standard coloring problem by allowing b times more colors while requiring that every vertex receives b colors and adjacent vertices receive disjoint subsets, with increasing b we approximate $\chi_f(G)$ better and better. Consequently, $\lim_{b\to\infty} \chi_b(G)/b = \chi_f(G)$.

Another connection concerns *Kneser graphs*. Recall that for positive integers a, b with $b < a/2$, the Kneser graph $KG_{a,b}$ has all b-element subsets of $\{1, 2, \ldots, a\}$ as vertices, and two subsets are considered adjacent if and only if they are disjoint. For instance, $KG_{5,2}$ is the well-known Petersen graph (see Fig. 1.2, right). Thus, $(a{:}b)$-coloring of a graph G can be interpreted as a homomorphism from G to the Kneser graph $KG_{a,b}$ (see Fig. 1.2). Kneser graphs are well studied in the context of

colorings, mostly due to the celebrated result of Lovász [105], who determined their chromatic number, initiating the field of topological combinatorics.

Multicolorings and $(a{:}b)$-colorings have been studied both from combinatorial [33, 54, 98] and algorithmic [30, 69, 70, 81, 87, 109, 113, 126] points of view. The main real-life motivation comes from the problem of assigning frequencies to nodes in a cellular network so that adjacent nodes receive disjoint sets of frequencies on which they can operate. This makes (near-)planar and distributed settings particularly interesting for practical applications. We refer to the survey of Halldórsson and Kortsarz [68] for a broader discussion.

In Chapter 6 we study the computational complexity of the following decision problem.

$(a{:}b)$-COLORING
Input: $G = (V, E)$, $a, b \in \mathbb{N}$
Question: Is G $(a{:}b)$-colorable?

Since the problem is already NP-hard for $a = 3$ and $b = 1$, we do not expect it to be solvable in polynomial time, and hence we look for an efficient exponential-time algorithm. A straightforward dynamic programming approach yields an algorithm with running time $O^*(2^n \cdot (b+1)^n)$ as follows. For each function $\eta\colon V(G) \to \{0, 1, \ldots, b\}$ and each $k = 0, 1, \ldots, a$, we create one boolean entry $D[\eta, k]$ denoting whether one can choose k independent sets in G so that every vertex $v \in V(G)$ is covered exactly $\eta(v)$ times. Then value $D[\eta, k]$ can be computed as a disjunction of values $D[\eta', k-1]$ over η' obtained from η by subtracting 1 on vertices from some independent set in G.

This simple algorithm can be improved by finding an appropriate algebraic formula for the number of $(a{:}b)$-colorings of the graph and using the inclusion-exclusion principle to compute it quickly, similarly as in the case of standard colorings [12]. Such an algebraic formula was given by Nederlof [118, Theorem 3.5] in the context of a more general Multi Set Cover problem. Nederlof also observed that in the case of $(a{:}b)$-COLORING, a simple application of the inclusion-exclusion principle to compute the formula yields an $O^*((b+1)^n)$-time exponential-space algorithm. Hua et al. [74] noted that the formulation of Nederlof [118] for Multi Set Cover can be also used to obtain a polynomial-space algorithm for this problem. By taking all maximal independent sets to be the family in the Multi Set Cover problem, and applying the classic Moon-Moser upper bound on their number [117], we obtain an algorithm for $(a{:}b)$-COLORING that runs in time $O^*(3^{n/3} \cdot (b+1)^n)$ and uses polynomial space. Note that by plugging $b = 1$ to the results above, we obtain algorithms for the standard coloring problem using $O^*(2^n)$ time and exponential space, or using $O^*(2.8845^n)$ time and polynomial space, which almost matches the fastest known procedures [12].

The complexity of $(a{:}b)$-COLORING becomes particularly interesting in the context of the Graph Homomorphism problem. By the celebrated result of Hell and Nešetřil [72] the problem is in P if H is bipartite and NP-complete otherwise. For quite a while it was open whether there is an algorithm for Graph Homomorphism running in time $2^{O(n+h)}$. As we mentioned in Theorem 5, an algorithm with run-

ning time $2^{o(n \log h)}$ contradicts the Exponential Time Hypothesis. However, Graph Homomorphism is a very general problem, hence researchers try to uncover a more fine-grained picture and identify families of graphs $\mathcal{H}$ such that the problem can be solved more efficiently whenever $H \in \mathcal{H}$. For example, Fomin, Heggernes and Kratsch [58] showed that when H is of treewidth at most t, then Graph Homomorphism can be solved in time $O^*((t+3)^n)$. It was later extended to graphs of cliquewidth bounded by t, with $O^*((2t+1)^{\max\{n,h\}})$ time bound by Wahlström [130]. On the other hand, H needs not be sparse to admit efficient homomorphism testing: the family of cliques admits the $O^*(2^n)$ running time as shown by Björklund et al. [12]. As noted above, this generalizes to Kneser graphs $KG_{a,b}$, by the $O^*((b+1)^n)$-time algorithm of Nederlof. In this context, the natural question is whether the appearance of b in the base of the exponent is necessary, or whether there is an algorithm running in time $O^*(c^n)$ for some universal constant c independent of b.

Our contribution. We show that the algorithms for $(a{:}b)$-coloring mentioned above are essentially optimal under the Exponential Time Hypothesis. Specifically, we prove the following results:

Theorem 9. *If there is an algorithm for $(a{:}b)$-coloring that runs in time $f(b) \cdot 2^{o(\log b)\cdot n}$, for some computable function $f(b)$, then ETH fails. This holds even if the algorithm is only required to work on instances where $a = \Theta(b^2 \log b)$.*

Corollary 10. *If there is an algorithm for Graph Homomorphism that runs in time $f(h) \cdot 2^{o(\log \log h)\cdot n}$, for some computable $f(h)$, then ETH fails. This holds even if the algorithm is only required to work on instances where H is a Kneser graph $KG_{a,b}$ with $a = \Theta(b^2 \log b)$.*

The bound for $(a{:}b)$-coloring is tight, as the straightforward $O^*(2^n \cdot (b+1)^n) = 2^{O(\log b)\cdot n}$ dynamic programming algorithm already shows. At first glance, one might have suspected that $(a{:}b)$-coloring, as an interpolation between classical coloring and fractional coloring, both solvable in $2^{O(n)}$ time [66], should be just as easy; Theorem 9 refutes this suspicion.

Corollary 10 in particular excludes any algorithm for testing homomorphisms into Kneser graphs with running time $2^{O(n+h)}$. It cannot give a tight lower bound matching the result mentioned in Theorem 5 for general homomorphisms, because $h = |V(KG_{a,b})| = \binom{a}{b}$ is not polynomial in b. On the other hand, it exhibits the first explicit family of graphs H for which the complexity of Graph Homomorphism increases with h.

In the proof, we first show a lower bound for the list variant of the problem, where every vertex is given a list of colors that can be assigned to it (see Section 6.1 for formal definitions). The list version is reduced to the standard version by introducing a large Kneser graph $KG_{a+b,b}$; we need a and b to be really small so that the size of this Kneser graph does not dwarf the size of the rest of the construction. However, this is not necessary for the list version, where we obtain lower bounds for a much wider range of functions $b(n)$.

Theorem 11. *If there is an algorithm for* List $(a{:}b)$-coloring *that runs in time* $2^{o(\log b)\cdot n}$*, then ETH fails. This holds even if the algorithm is only required to work on instances where* $a = \Theta(b^2 \log b)$ *and* $b = \Theta(b(n))$ *for an arbitrarily chosen polynomial-time computable function* $b(n)$ *such that* $b(n) \in \omega(1)$ *and* $b(n) = O(n/\log n)$*.*

Finally, we observe that from our main result one can infer a lower bound for the complexity of the (r, k)-Monomial Testing problem. In the following definition a polynomial is homogeneous if all its monomials have the same total degree k.

(r, k)-Monomial Testing
Input: An arithmetic circuit that evaluates a homogeneous polynomial $P(x_1, x_2, \ldots, x_n)$ over some field $\mathbb{F}$, a parameter r.
Question: Does P have some monomial in which every variable has individual degree not larger than r

Abasi et al. [2] gave a randomized algorithm that solves this problem in time $O^*(2^{O(k\cdot\frac{\log r}{r})})$, where k is the degree of the polynomial, assuming that $\mathbb{F} = \mathrm{GF}(p)$ for a prime $p \leq 2r^2 + 2r$. This algorithm was later derandomized by Gabizon et al. [60] within the same running time, but under the assumption that the circuit is *non-cancelling*: it has only input, addition, and multiplication gates. Abasi et al. [2] and Gabizon et al. [60] gave a number of applications of low-degree monomial detection to concrete problems. For instance, r-Simple k-Path, the problem of finding a walk of length k that visits every vertex at most r times, can be solved in time $O^*(2^{O(k\cdot\frac{\log r}{r})})$. However, for r-Simple k-Path, as well as other problems that can be tackled using this technique, the best known lower bounds under ETH exclude only algorithms with running time $O^*(2^{o(\frac{k}{r})})$. Whether the $\log r$ factor in the exponent is necessary was left open by Abasi et al. and Gabizon et al.

We observe that the List $(a{:}b)$-coloring problem can be reduced to (r, k)-Monomial Testing over the field $\mathrm{GF}(2)$ in such a way that an $O^*(2^{k\cdot o(\frac{\log r}{r})})$-time algorithm for the latter would imply a $2^{o(\log b)\cdot n}$-time algorithm for the former, which would contradict ETH. Thus, we show that the known algorithms for (r, k)-Monomial Testing most probably cannot be sped up in general; nevertheless, the question of lower bounds for specific applications remains open. However, going through List $(a{:}b)$-coloring to establish a lower bound for (r, k)-Monomial Testing is actually quite a detour, because the latter problem has a much larger expressive power. Therefore, we also give a more straightforward reduction that starts from a convenient form of Subset Sum; this reduction also proves the lower bound for a wider range of r, expressed as a function of k.

1.2.5 Minimax Approval Voting

One of the central problems in artificial intelligence and computational social choice is aggregating preferences of individual agents (see the overview of Conitzer [34]). Here we focus on *multi-winner choice*, where the goal is to select a k-element subset of a set of candidates. Given preferences of the agents over the candidates, a multi-winner

voting rule can be used to select a subset of candidates that in some sense are preferred by the agents. This scenario covers a variety of settings: nations elect members of parliament or societies elect committees [23], web search engines choose pages to display in response to a query [50], airlines select movies available on board [123, 52], companies select a group of products to promote [106], etc.

We restrict our attention to approval-based multi-winner rules, i.e., rules where each voter expresses his or her preferences by providing a *subset of the candidates* which he or she approves. Various voting rules are studied in the literature. In the simplest one, Approval Voting (AV), occurrences of each candidate are counted and k most often approved candidates are selected. While this rule has many desirable properties in the single winner case [53], in the multi-winner scenario its merits are often considered less clear [89], e.g., because it fails to reflect the diversity of interests in the electorate [82]. Therefore, numerous alternative rules have been proposed, including Satisfaction Approval Voting, Proportional Approval Voting, and Reweighted Approval Voting (see [82] for details). We study a rule called Minimax Approval Voting (MAV), introduced by Brams et al. [15]. Here, we see the votes and the choice as 0-1 strings of length m (characteristic vectors of the subsets, i.e., the candidate i is approved if the string contains 1 at position i). For two strings x and y of the same length the Hamming distance $\mathcal{H}(x, y)$ is the number of positions where x and y differ, e.g., $\mathcal{H}(011, 101) = 2$. In MAV, we look for a 0-1 string with exactly k ones that minimizes the maximum Hamming distance to a vote. In other words, MAV minimizes the disagreement with the least satisfied voter and thus it is highly egalitarian: no voter is ignored and a majority of voters cannot guarantee a specific outcome [15, 92].

Our focus is on the computational complexity of computing the choice based on the MAV rule. The decision version of MAV is formally stated below.

Minimax Approval Voting
Input: A multiset $S = \{s_1, \ldots, s_n\}$ of 0-1 strings of length m (also called votes), two integers k and d.
Question: Does there exist a string $s \in \{0, 1\}^m$ with exactly k ones such that for every $i = 1, \ldots, n$ we have $\mathcal{H}(s, s_i) \leq d$?

In the optimization version of Minimax Approval Voting we minimize d, i.e., given a multiset S and an integer k as before, the goal is to find a string $s \in \{0, 1\}^m$ with exactly k ones which minimizes $\max_{i=1,\ldots,n} \mathcal{H}(s, s_i)$.

A reader familiar with string problems might recognize that Minimax Approval Voting is tightly connected with the following classical NP-complete problem called Closest String.

Closest String
Input: A multiset $S = \{s_1, \ldots, s_n\}$ of 0-1 strings of length m (also called votes), an integer d.
Question: Does there exist a string $s \in \{0, 1\}^m$ such that for every $i = 1, \ldots, n$ we have $\mathcal{H}(s, s_i) \leq d$?

Indeed, LeGrand et al. [93] showed that Minimax Approval Voting is NP-

complete by a reduction from Closest String with binary alphabet. (First proof of NP-completeness of Minimax Approval Voting was shown using reduction from Vertex Cover [92].) This motivated the study on Minimax Approval Voting in terms of approximability and fixed-parameter tractability.

Previous results on Minimax Approval Voting The first approximation result was a simple 3-approximation algorithm due to LeGrand et al. [93], obtained by choosing an arbitrary vote and taking any k approved candidates from the vote (extending it arbitrarily to k candidates if needed). Next, a 2-approximation was shown by Caragiannis et al. [19] using an LP-rounding procedure. Finally, Byrka et al. [17] presented a polynomial time approximation scheme (PTAS), i.e., an algorithm that for any fixed $\epsilon > 0$ gives a $(1+\epsilon)$-approximate solution in polynomial time. More precisely, their algorithm runs in time $m^{O(1/\epsilon^4)} + n^{O(1/\epsilon^3)}$ which is polynomial in the number of voters n and the number of alternatives m.

The study of FPT algorithms for Minimax Approval Voting was initiated by Misra et al. [116]. They show for example that Minimax Approval Voting parameterized by k (the number of ones in the solution) is $W[2]$-hard, which implies that there is no FPT algorithm, unless there is a highly unexpected collapse in parameterized complexity classes. From a positive perspective, they show that the problem is FPT when parameterized by the maximum allowed distance d or by the number of votes n. Their algorithm runs in time $O^*(d^{2d})$.[3]

Previous results on Closest String It is interesting to compare the known results on Minimax Approval Voting with the corresponding ones on the better researched Closest String. The first PTAS for Closest String was given by Li et al. [94] with running time bounded by $n^{O(1/\epsilon^4)}$ where n is the number of the input strings. This was later improved by Andoni et al. [7] to $n^{O(\frac{\log 1/\epsilon}{\epsilon^2})}$, and then by Ma et al. [107] to $n^{O(1/\epsilon^2)}$.

The first FPT algorithm for Closest String, running in time $O^*(d^d)$ was given by Gramm et al. [64]. This was later improved by Ma et al. [107], who gave an algorithm with running time $O^*(2^{O(d)} \cdot |\Sigma|^d)$, which is more efficient for constant-size alphabets. Further substantial progress is unlikely, since Lokshtanov et al. [103] have shown that Closest String admits no algorithms running in time $O^*(2^{o(d \log d)})$ or $O^*(2^{o(d \log |\Sigma|)})$, unless the Exponential Time Hypothesis (ETH) [76] fails.

The discrepancy between the state of the art for Closest String and Minimax Approval Voting raises interesting questions. First, does the additional constraint on the number of ones in Minimax Approval Voting really make the problem harder and the PTAS has to be significantly slower? Similarly, although in Minimax Approval Voting the alphabet is binary, no $O^*(2^{O(d)})$-time algorithm is known, in

[3] Actually, in the article [116] the authors claim the slightly better running time of $O^*(d^d)$. However, there is a flaw in the analysis [100, 115]: it states that the initial solution v is at distance at most d from the solution, while it can be at distance $2d$ because of what we call here the k-completion operation. This increases the maximum depth of the recursion to d (instead of the claimed $d/2$).

contrast to Closest String. Can we find such an algorithm? Our goal is to answer the latter of these questions.

Our results We show that, unless the ETH fails, there is no algorithm for Minimax Approval Voting running in time $O^*(2^{o(d \log d)})$. In other words, the algorithm of Misra et al. [116] is essentially optimal, and indeed, in this sense Minimax Approval Voting is harder than Closest String.

Consequences The lower bound presented in this thesis first appeared in the article of Cygan, Kowalik, Socała and Sornat [44]. The lower bound motivated further positive results contained in the paper of Cygan et al. [44], namely, a parameterized approximation scheme, i.e., a randomized Monte-Carlo algorithm which, given an instance (S, k, d) and a number $\epsilon > 0$, finds a solution at distance at most $(1+\epsilon)d$ in time $O^*((3/\epsilon)^{2d})$ or reports that there is no solution at distance at most d (with arbitrarily small positive constant probability of error in the case of the negative answer). Note that our lower bound implies that, under (randomized version of) ETH, this is essentially optimal, i.e., there is no parameterized approximation scheme running in time $O^*(2^{o(d \log(1/\epsilon))})$. Indeed, if such an algorithm existed, by picking $\epsilon = 1/(d+1)$ we would get an exact algorithm which contradicts our lower bound.

It is also worthwhile to mention that the parametrized approximation scheme served in the work of Cygan et al. [44] as a tool to get a faster PTAS for Minimax Approval Voting.

Theorem 12 ([44]). *For each $\epsilon > 0$ we can find $(1+\epsilon)$-approximate solution for the* Minimax Approval Voting *problem in time $n^{O\left(\frac{\log 1/\epsilon}{\epsilon^2}\right)} \cdot m^{O(1)}$ with probability at least $1 - r$, for any fixed $r > 0$.*

1.2.6 k-OPT Detection

In the Traveling Salesman Problem (TSP) one is given a complete graph $G = (V, E)$ and a weight function $w : E \to \mathbb{N}$. The goal is to find a Hamiltonian cycle in G (also called a *tour*) of minimum weight. This is one of the central problems in computer science and operation research. It is well known to be NP-hard and has been researched from different perspectives, most notably using approximation [8, 31, 121], exponential-time algorithms [71, 80] and heuristics [119, 97, 36].

In practice, TSP is often solved by means of local search heuristics where we begin from an arbitrary Hamiltonian cycle in G, and then the cycle is modified by means of some local changes in a series of steps. After each step the weight of the cycle should improve; when the algorithm cannot find any improvement it stops. One of the most successful examples of this approach is the k-opt heuristic, where in each step an improving k-move is performed. Given a Hamiltonian cycle H in a graph $G = (V, E)$ a *k-move* is an operation that removes k edges from H and adds k edges of G so that the resulting set of edges H' is a new Hamiltonian cycle. The k-move is *improving* if the weight of H' is smaller than the weight of H. The k-opt heuristic has been introduced in 1958 by Croes [36] for $k = 2$, and then applied for $k = 3$ by Lin [96]

in 1965. Then in 1972 Lin and Kernighan designed a complicated heuristic which uses k-moves for unbounded values of k, though restricting the space of k-moves to search to so-called sequential k-moves. A variant of this heuristic called LKH, implemented by Helsgaun [73], solves optimally instances up to 85 900 cities. Among other modifications, the variant searches for non-sequential 4- and 5-moves. From the theory perspective, the quality of the solutions returned by k-opt, as well as the length of the sequence of k-moves needed to find a local optimum, was studied, among others, by Johnson, Papadimitriou and Yannakakis [78], Krentel [86] and Chandra, Karloff and Tovey [24]. More recently, smoothed analysis of the running time and approximation ratio was investigated by Manthey and Veenstra [88] and Künnemann and Manthey [108].

We study the k-opt heuristic but we focus on its basic ingredient, namely on finding a single improving k-move. We consider the following decision problem.

k-OPT Detection
Input: A TSP tour H in an edge weighted complete graph G.
Question: Does there exist an improving k-move?

In the optimization version, called k-OPT Optimization, the goal is to find a k-move that gives the largest weight improvement, if any. Unfortunately, these are computationally hard problems. Namely, Marx [110] has shown that k-OPT Detection is $W[1]$-hard, which means that it is unlikely to be solvable in $f(k)n^{O(1)}$ time, for any function f. Later Guo, Hartung, Niedermeier and Suchý [67] proved that there is no algorithm running in time $n^{o(k/\log k)}$, unless Exponential Time Hypothesis (ETH) fails. This explains why in practice people use exhaustive search running in $O(n^k)$ time for every fixed k, or faster algorithms which explore only a very restricted subset of all possible k-moves.

Very recently, de Berg, Buchin, Jansen and Woeginger [48] have shown that it is possible to improve over the naive exhaustive search. For every fixed $k \geq 3$ their algorithm runs in time $O(n^{\lfloor 2k/3 \rfloor + 1})$ and uses $O(n)$ space. In particular, it gives $O(n^3)$ time for $k = 4$. Thus, the algorithm of de Berg et al. is of high practical interest: the complexity of the $k = 4$ case now matches the complexity of $k = 3$ case, and hence it seems that one can use 4-opt in all the applications where 3-opt was fast enough. De Berg et al. show also that a progress for $k = 3$ is unlikely, namely k-OPT Detection has an $O(n^{3-\epsilon})$-time algorithm for some $\epsilon > 0$ iff All Pairs Shortest Paths problem can be solved in $O(n^{3-\delta})$-time algorithm for some $\delta > 0$.

Cygan, Kowalik and Socała [43] extend the line of research started in [48] by showing an algorithm running in time $O(n^{(1/4+\epsilon_k)k})$ for every fixed k, where $\lim_{k\to\infty} \epsilon_k = 0$. The values of ϵ_k are computed for every $k \leq 10$ (see Table 1.1 for the values $k \leq 8$). In particular the new algorithm improves over the previous one by de Berg et al. [48] for every $k \geq 5$.

Our Results. We show a good reason why it may be hard to improve over the $O(n^3)$-time algorithm of de Berg et al. for 4-OPT Optimization:

Theorem 13. *If there is $\epsilon > 0$ such that* 4-OPT Detection *admits an algorithm in time $O(n^{3-\epsilon} \cdot (\log M)^{O(1)})$, then there is $\delta > 0$ such that both* Negative Edge-

k	2	3	4	5	6	7	8	...	∞
known algorithm	n^2	n^3	n^3	$n^{3.4}$	n^4	$n^{4.25}$	$n^{4\frac{2}{3}}$	...	$n^{(1/4+\epsilon_k)k}$ for $\epsilon_k \to 0$
source	folk.	folk.	[48]	[43]	[43]	[43]	[43]		[43]
known lower bound	n^2	$n^{3-\epsilon}$	$n^{3-\epsilon}$	n^2					$n^{\Omega(k/\log k)}$
source	folk.	[48]	this thesis	folklore					[67]

Table 1.1: Known running times and lower bounds for k-OPT Detection depending on k. (Up to the big-O and big-Ω respectively.)

Weighted Triangle *and* All Pairs Shortest Paths *admit an algorithm in time* $O(n^{3-\delta} \cdot (\log M)^{O(1)})$*, where in all cases we refer to* n*-vertex input graphs with integer weights from* $\{-M, \ldots, M\}$*.*

Note that although the family of 4-moves contains all 3-moves, it is still possible that there is no improving 3-move, but there is an improving 4-move. Thus the previous lower bound of de Berg et al. does not imply our lower bound, though our reduction is essentially an extension of the one by de Berg et al. [48] with a few additional technical tricks. Table 1.1 summarizes the current state of the art regarding lower and upper bounds for k-OPT Detection.

1.3 Articles

Most of this dissertation comes from the following articles and preprints:

- *Tight Lower Bound for the Channel Assignment Problem*, ACM Trans. Algorithms, 2016 [125]. The extended abstract of the publication was published in the proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, 2015 [124]. The results of this article are contained in Chapter 3.

- *The Hardness of Subgraph Isomorphism*, which is a joint work with Marek Cygan and Jakub Pachocki, CoRR, 2015 [45]. This preprint contains the proofs of Theorems 3 and 4 which are presented in Chapter 4 of this dissertation. Theorems 4, 5 and 6 are contained in the merged paper *Tight bounds for Graph Homomorphism and Subgraph Isomorphism* authored by Cygan, Fomin, Golovnev, Kulikov, Mihajlin, Pachocki and Socała [38]. Its journal version [40] is currently accepted to *Journal of the ACM (JACM)*.

- *On the Fine-Grained Complexity of Rainbow Coloring* which is a joint work with Łukasz Kowalik and Juho Lauri. The extended abstract of this publication was published in the proceedings of the 24th Annual European Symposium on Algorithms, ESA, 2016 [83]. The lower bound results of this article are contained in Chapter 5.

- *Tight lower bounds for the complexity of multicoloring*, which is a joint work with Marthe Bonamy, Łukasz Kowalik, Michał Pilipczuk and Marcin Wrochna, CoRR, 2016 [13]. This work is currently accepted to the 25th Annual European Symposium on Algorithms (ESA 2017). The results of this work are contained in Chapter 6.

- *Approximation and Parameterized Complexity of Minimax Approval Voting* which is a joint work with Marek Cygan, Łukasz Kowalik and Krzysztof Sornat. The extended abstract of this publication was published in the proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, 2017 [44]. The lower bound result of this article is contained in Chapter 7.

- *Improving TSP tours using dynamic programming over tree decomposition*, which is a joint work with Marek Cygan and Łukasz Kowalik, CoRR, 2017 [43]. This work is currently accepted to the 25th Annual European Symposium on Algorithms (ESA 2017). The lower bound result of this work is contained in Chapter 8.

1.4 Acknowledgements

I would like to thank my advisor Łukasz Kowalik for help at each step of the way.

Moreover I would like to thank all my co-authors: Michael Cohen, Michał Farnik, Jesper Nederlof and especially Marthe Bonamy, Marek Cygan, Fedor V. Fomin, Alexander Golovnev, Łukasz Kowalik, Alexander S. Kulikov, Juho Lauri, Ivan Mihajlin, Jakub Pachocki, Michał Pilipczuk, Krzysztof Sornat and Marcin Wrochna who are the co-authors of the core publications my dissertation is based on.

Chapter 2

Notation and Preliminaries

2.1 Notation

For an integer k, by $[k]$ we denote the set $\{1, 2, \ldots, k\}$. By $x^{\underline{k}}$ we denote the falling factorial, i.e., $x^{\underline{k}} = x(x-1)\cdots(x-k+1)$. For a (partial) function c, by $\mathrm{Dom}(c)$ we denote its domain. The $O^*(\ )$ notation suppresses polynomial factors.

If I and J are instances of decision problems P and R, respectively, then we say that I and J are *equivalent*, when either both I and J are YES-instances or both are NO-instances of the respective problems.

For standard graph-theoretic notions, we refer the reader to [41, 49]. For problem definitions, we refer the reader to Appendix A.

2.2 Exponential-Time Hypothesis toolbox

The Exponential Time Hypothesis (ETH) of Impagliazzo et al. [76] states that there exists a constant $c > 0$, such that there is no algorithm solving 3-CNF-SAT in time $O^*(2^{cn})$. During the recent years, ETH became the central conjecture used for proving tight bounds on the complexity of various problems.

2.2.1 Reducing from 3-CNF-SAT

One of the most important results connected to ETH is the *Sparsification Lemma* [77], which essentially gives a reduction from an arbitrary instance of k-CNF-SAT to an instance where the number of clauses is linear in the number of variables.

Lemma 14 (Sparsification Lemma [76])**.** *For each $\varepsilon > 0$ there exist a constants c_ε, such that any* 3-CNF-SAT *formula φ with n variables can be expressed as $\varphi = \vee_{i=1}^{t} \psi_i$, where $t \leq 2^{\varepsilon n}$ and each ψ_i is a* 3-CNF-SAT *formula with the same variable set as φ, but contains at most $c_\varepsilon n$ clauses. Moreover, this disjunction can be computed in time $O^*(2^{\varepsilon n})$.*

The following well-known corollary can be derived by combining ETH with the Sparsification Lemma.

Theorem 15 (see e.g. Theorem 14.4 in [41]). *Unless ETH fails, there is no algorithm for* 3-CNF-SAT *that runs in time* $2^{o(n+m)}$*, on formulas with* n *variables and* m *clauses.*

We need the following regularization result of Tovey [127]. Following Tovey, by (3,4)-CNF-SAT we call the following variant of 3-CNF-SAT.

(3,4)-CNF-SAT
Input: A k-CNF-SAT formula φ where each clause of the input formula contains exactly 3 different variables, and each variable occurs in at most 4 clauses.
Question: Is φ satisfiable?

Lemma 16 ([127]). *Given a* 3-CNF-SAT *formula* φ *with* n *variables and* m *clauses one can transform it in polynomial time into an equivalent* (3,4)-CNF-SAT *instance* φ' *with* $O(n+m)$ *variables and clauses.*

Theorem 15 and Lemma 16 give the following corollary.

Corollary 17. *Unless ETH fails, there is no algorithm for* (3,4)-CNF-SAT *that runs in time* $2^{o(n)}$*, where* n *denotes the number of variables of the input formula.*

(3,4)-CNF-SAT has the following useful property that we exploit in a few of our proofs. For a given formula φ being an instance of (3,4)-CNF-SAT consider a bipartite graph where the vertices of one side are the variables of φ and the vertices of the other side are the clauses of φ. In this graph we put an edge between a variable and a clause if and only if the variable appears in the clause. Such a graph has a bounded degree and therefore for any constant distance $d \geq 1$ we can find in polynomial time a distance-d-coloring with $k = O(1)$ colors. Namely, if two vertices/clauses are colored with the same color it means that there is no path of length d or shorter between them which means that they are fairly independent of each other. This exposes a lot of additional structure of the given (3,4)-CNF-SAT instance that can be used during the reduction.

Note also that if some color classes are larger than, say, $\lceil (n+m)/k \rceil$ we can split them into smaller chunks and this will multiply the total number of colors only by a constant factor and therefore we can assume that all the colors are small.

2.2.2 Reducing from other problems

Sometimes it is more convenient to start a reduction from a problem other than 3-CNF-SAT variations.

3-Coloring
Input: A graph G.
Question: Is it possible to color the vertices of G with three colors in such a way that there are no adjacent vertices of the same color?

There is a well known reduction from 3-CNF-SAT to 3-Coloring (see e.g. [122, 104, 62, 39] for the discussion).

Theorem 18. *There is a polynomial-time reduction that transforms an instance of* 3-CNF-SAT *with n variables and m clauses into an equivalent instance of* 3-Coloring *with $O(n+m)$ vertices and maximum degree four.*

This together with the Sparsification Lemma 14 gives a very useful corollary.

Corollary 19 (folklore)**.** *Assuming ETH, there is no $2^{o(n)}$-time algorithm for* 3-Coloring *on n-vertex graphs of maximum degree four.*

The 3-Coloring problem was used in the papers of Fomin et al. [57] and Cygan et al. [38] to prove Graph Homomorphism and Subgraph Isomorphism lower bounds. It was used also in [103] to prove a lower bound for $k \times k$-Clique which is another convenient starting point.

$k \times k$-Clique
Input: A graph G over the vertex set $V = [k] \times [k]$, i.e., V forms a grid (as a vertex set; the edge set of G is a part of the input and it can be arbitrary) with k rows and k columns.
Question: Is there in G a clique containing exactly one vertex in each row?

Theorem 20 (Lokshtanov et al. [103])**.** *Assuming ETH, there is no $2^{o(k \log k)}$-time algorithm for $k \times k$-Clique.*

In $k \times k$-HittingSet we are given m sets $S_1, S_2, \ldots S_m \subseteq [k] \times [k]$ and the question is whether there is a set $S \subseteq [k] \times [k]$ containing exactly one element from each row such that $S \cap S_i \neq \emptyset$ for every $i \in [m]$.

Theorem 21 (Lokshtanov et al. [103])**.** *Assuming ETH, there is no $2^{o(k \log k)} \cdot n^{O(1)}$-time algorithm for $k \times k$-HittingSet.*

$k \times k$-Clique, $k \times k$-HittingSet and a few similar problems were used in the paper of Lokshtanov et al. [103] for Closest String, Distortion and Disjoint Paths lower bounds. The $k \times k$-Clique problem is also used in Chapter 7 of this dissertation to prove a lower bound for Minimax Approval Voting. In fact in Chapter 7 we present also a reduction from 3-CNF-SAT, hence it is possible to compare them.

Another interesting problem is Carry-Less Subset Sum.

Carry-Less Subset Sum
Input: $n+1$ numbers $s, a_1, \ldots, a_n$, each represented as n decimal digits. For any number x, the j-th decimal digit of x is denoted by $x^{(j)}$. It is assumed that $\sum_{i=1}^{n} a_i^{(j)} < 10$, for every $j = 1, \ldots, n$.
Question: Does there exist a sequence of indices $1 \leq i_1 < \ldots < i_k \leq n$ such that $\sum_{q=1}^{k} a_{i_q} = s$?

Note that by the small sum assumption, this is equivalent to the statement that $\sum_{q=1}^{k} a_{i_q}^{(j)} = s^{(j)}$, for every $j = 1, \ldots, n$. The standard NP-hardness reduction from 3-CNF-SAT to Subset Sum (see e.g. [35]) in fact gives instances of Carry-Less Subset Sum of linear size, yielding the following.

Lemma 22. *Unless ETH fails, the* Carry-Less Subset Sum *problem cannot be solved in* $2^{o(n)}$ *time.*

We provide a bit more detailed proof in Chapter 6 where we use Lemma 22 for proving a lower bound for (r, k)-Monomial Testing.

Chapter 3

Channel Assignment

In this chapter we prove that there is no algorithm solving Channel Assignment in time $2^{o(n \log n)}$ (times a polynomial in the bit size), unless ETH fails.

Organization of the chapter. In Section 3.1 we describe a sequence of reductions starting in 3-CNF-SAT and ending in Equal Weight Matchings and the conclusions on the hardness of Equal Weight Matchings. In Section 3.2 we present a reduction from Equal Weight Matchings to Channel Assignment and prove the hardness of Channel Assignment.

Additional notation for this chapter. We denote an instance of (decisional) Channel Assignment by $I = (G, w, s)$ where the instance is satisfied when $\text{span}(G, w) \leq s$.

3.1 Hardness of Equal Weight Matchings

In this section we describe a sequence of reductions starting in 3-CNF-SAT and ending in Equal Weight Matchings and the consequences of these reductions on the complexity of Equal Weight Matchings. In the second of these two reductions we compress the instance of size $O(n)$ to an instance with $O\left(\frac{n}{\log n}\right)$ vertices, which is an important part of our result.

3.1.1 From 3-CNF-SAT to Family Intersection.

The intuition is that for a given instance of 3-CNF-SAT we consider a set of the *occurrences* of the variables in the formula i.e. we treat any two different occurrences of the same variable as they were two different variables. Note that in a 3-CNF-SAT instance with n variables and m clauses we have $3m$ occurrences of the n variables so there are 2^{3m} assignments of the occurrences.

We would like to represent two useful subsets of the set of all 2^{3m} assignments of the occurrences. The first is the set of the *consistent* assignments i.e. such assignments of the occurrences that all the occurrences of the same variable have the same value.

The second is the set of the assignments of the occurrences such that every clause is satisfied (although they are allowed to have different values for different occurrences of the same variable i.e. they do not need to be consistent). Note that the instance of 3-CNF-SAT is a YES-instance if and only if the intersection of these two sets is nonempty.

To represent those two sets we would like to use the following concept. For a function $f : [a] \times [b] \to \mathbb{N}$ we define $X_f = \{\sum_{i=1}^{a} f(i, \sigma(i)) \,|\sigma : [a] \to [b]\}$. We call this set an *$f$-family*.

We will define a function f such that the elements of the f-family X_f correspond to the assignments of the occurrences such that every two occurrences of the same variable have the same value. Then we define another function g such that the g-family X_g represents the assignments of the occurrences such that every clause is satisfied. Thus we reduce 3-CNF-SAT into the following problem:

FAMILY INTERSECTION
Input: A function $f : [a] \times [b] \to \mathbb{N}$ and a function $g : [c] \times [d] \to \mathbb{N}$.
Question: Is $X_f \cap X_g$ nonempty?

Example 1. *Let us illustrate our approach on a* 2-CNF-SAT *formula* $\varphi = (\alpha \vee \beta) \wedge (\neg\alpha \vee \gamma)$. *So* $n = 3$ *and* $m = 2$. *We can make a distinction between different occurrences of the same variables* $\varphi' = (\alpha_1 \vee \beta_1) \wedge (\neg\alpha_2 \vee \gamma_1)$. *So we have four occurrences of the variables and* $2^4 = 16$ *assignments. We represent those assignments as numbers from the set* $\{0, 1, \ldots, 2^4 - 1\}$ *However, it will be convenient to refer to these numbers as bit vectors of length* 4 *where the* i*-th bit represents the value of the* i*-th occurrence (among all the occurrences of all the variables).*

To represent the set of the consistent assignments of the occurrences we can use a function $f : [n] \times [2] \to \mathbb{N}$ *such that the value of* $f(i, 1)$ *is a bit vector representing all the occurrences of the* i*-th variable and* $f(i, 2) = 0$. *So in our example we have* $f(1, 1) = 1010_2$, $f(2, 1) = 0100_2$, *and* $f(3, 1) = 0001_2$. *Therefore* $X_f = \{0000_2, 0001_2, 0100_2, 0101_2, 1010_2, 1011_2, 1110_2, 1111_2\}$.

To represent the set of the assignments which satisfies all the clauses we can use a function $g : [m] \times [3] \to \mathbb{N}$ *such that* $g(i, j)$ *is a* j*-th assignment (in some fixed order) of the occurrences of the variables in the* i*-th clause which satisfies this clause. Note that every clause in* 2-CNF-SAT *have* 3 *assignments of the occurrences which satisfy this clause. So in our example we have* $g(1, 1) = 1000_2$, $g(1, 2) = 0100_2$, $g(1, 3) = 1100_2$, $g(2, 1) = 0000_2$, $g(2, 2) = 0001_2$ *and* $g(2, 3) = 0011_2$. *Therefore* $X_g = \{0100_2, 0101_2, 0111_2, 1000_2, 1001_2, 1011_2, 1100_2, 1101_2, 1111_2\}$.

The set $X_f \cap X_g = \{0100_2, 0101_2, 1011_2, 1111_2\}$ *is the set of all the consistent assignments of the occurrences such that each clause is satisfied.*

We can formalize our observation as following.

Lemma 23. *There is a polynomial time reduction from a given instance of* 3-CNF-SAT *with* n *variables and* m *clauses into an instance of* FAMILY INTERSECTION *with* $f : [n] \times [2] \to \mathbb{N}$ *and* $g : [m] \times [7] \to \mathbb{N}$ *such that* $\max X_f < 2^{3m}$ *and* $\max X_g < 2^{3m}$.

Proof. Let $V = \{v_1, v_2, \ldots, v_n\}$ and $C = \{c_1, c_2, \ldots, c_m\}$ be the sets of variables and clauses of the input formula, respectively. Let $D = \{d_1, d_2, \ldots, d_{3m}\}$ be the set of all $3m$ occurrences of our n variables in our m clauses. We will treat these occurrences as separate variables. For every variable $v_i \in V$ we define a set $I_i \subseteq [3m]$ such that $j \in I_i$ if and only if d_j is an occurrence of the variable v_i. Similarly for every clause $c_i \in C$ we define a set $J_i \subseteq [3m]$ such that $j \in J_i$ if and only if d_j is an occurrence (of any variable) belonging to the clause c_i. For every $i \in [m]$ we have $|J_i| = 3$.

For every clause c_i we can treat the subsets of J_i as the assignments of the occurrences d_j belonging to the clause c_i. We treat the subset $K \subseteq J_i$ as the assignment of the occurrences in the clause c_i such that the occurrence d_j is set to 1 if and only if $j \in K$, otherwise it is set to 0. We say that $K \subseteq J_i$ satisfies the clause c_i if the corresponding assignment of the occurrences satisfies this clause.

For every clause $c_i \in C$ let us define the set

$$P_i = \{K \subseteq J_i : K \text{ satisfies the clause } c_i\}.$$

Again note that we treat here all the occurrences as the different variables. Note that $|P_i| = 7$ for every i, so we can denote $P_i = \{P_i^1, P_i^2, \ldots, P_i^7\}$.

A number from $0, 1, \ldots, 2^{3m} - 1$ can be interpreted in the binary system as the characteristic vector of length $3m$ of a subset of the indices of the occurrences i.e., that the i-th bit represents if the occurrence d_i belongs to this subset or not. We define a function $f : [n] \times [2] \to \mathbb{N}$ such that for every $i \in [n]$ we set $f(i, 1) = \sum_{j \in I_i} 2^{j-1}$ and $f(i, 2) = 0$. In other words the number $f(i, 1)$ represents the characteristic vector of all the occurrences of the variable v_i. Note that $\{\sigma : [n] \to [2]\}$ corresponds to the set of all assignments of variables. Therefore X_f is the set of all the characteristic vectors which represent all the assignments of the occurrences such that all the occurrences of the same variable have the same value.

We define a function $g : [m] \times [7] \to \mathbb{N}$ such that for every $i \in [m]$ and for every $j \in [7]$ we can set $g(i, j) = \sum_{k \in P_i^j} 2^{k-1}$. Then for every $i \in [m]$ the numbers $g(i, 1), g(i, 2) \ldots, g(i, 7)$ represent the characteristic vectors of all the assignments of the occurrences in the clause c_i which satisfy this clause. Therefore the set X_g is the set of all the characteristic vectors which represents the assignments of all $3m$ occurrences such that all the clauses are satisfied.

It follows that the set $X_f \cap X_g$ is the set of all the characteristic vectors which represent the assignments of the occurrences such that all the occurrences of the same variable have the same value and all the clauses are satisfied. In other words, elements of $X_f \cap X_g$ correspond to satisfying assignments. □

3.1.2 From Family Intersection to Equal Weight Matchings.

Consider an f-family X_f and a g-family X_g for some functions $f : [n] \times [2] \to \mathbb{N}$ and $g : [m] \times [7] \to \mathbb{N}$. In this section we show how to encode X_f in some weighted bipartite graph G_1 so that the set of the weights of the perfect matchings in G_1 will be equal to X_f. Similarly we will encode X_g in some bipartite graph G_2 such that

the set of the weights of the perfect matchings in G_2 will be equal to X_g. So the set $X_f \cap X_g$ is nonempty if and only if G_1 and G_2 contain perfect matchings with the same weight. Moreover the number of the vertices of the graph G_1 will be $O\left(\frac{n}{\log n}\right)$ and the number of the vertices of the graph G_2 will be $O\left(\frac{m}{\log m}\right)$. This is a crucial step of our construction, because the instance size decreases (by a logarithmic factor).

Before we describe the reduction we need two following technical lemmas which describe a construction of permutations with certain properties. The permutations correspond naturally to perfect matchings in bipartite graphs. Elements of $[k]^b$ will be treated as b-character words over alphabet $[k]$, i.e. for $x \in [k]$ and $w \in [k]^b$ by xw we mean the word of length $b+1$ obtained by concatenating x and w. For convenience we define a set $\hat{\mathbb{N}} = \left\{\hat{0}, \hat{1}, \hat{2}, \ldots\right\}$ as a copy of the natural numbers $\mathbb{N}$ and for every $n \in \mathbb{N}$ we define $[\hat{n}] = \left\{\hat{1}, \hat{2}, \ldots, \hat{n}\right\}$. Every set $[\hat{k}]^b$ is just a copy of $[k]^b$ so we refer to bijections between $[k]^b$ and $[\hat{k}]^b$ as to permutations.

The first lemma provides a way of merging k permutations $\phi_1, \phi_2, \ldots, \phi_k : [\hat{k}]^b \to [k]^b$ into one permutation $\phi : [\hat{k}]^{b+1} \to [k]^{b+1}$ in a way specified by a function $\rho : [\hat{k}]^b \to [k]$. The second lemma is using the first one to provide a way of encoding a function with one argument and k values as a permutation of the number of elements that is sublinear with respect to the size of the domain of the function. This leads us later in the reduction to the way of encoding an f-family in a full weighted bipartite graph with sublinear number of the vertices.

Lemma 24. *For every $b \in \mathbb{N}$ and for a given sequence of permutations $\phi_1, \phi_2, \ldots, \phi_k : [\hat{k}]^b \to [k]^b$ and for every function $\rho : [\hat{k}]^b \to [k]$ there is a permutation $\phi : [\hat{k}]^{b+1} \to [k]^{b+1}$ such that*

(i) for every $x \in [k]$ and for every $\hat{w} \in [\hat{k}]^b$ there exists $y \in [k]$ such that $\phi(\hat{x}\hat{w}) = y\phi_x(\hat{w})$ and moreover

(ii) for every $\hat{w} \in [\hat{k}]^b$ we have $\phi\left(\hat{1}\hat{w}\right) = \rho(\hat{w})\,\phi_1(\hat{w})$.

Before we proceed to the proof we suggest the reader to take a look at an example in Figure 3.1 ($b = 1$, $k = 3$).

Proof. We start with the permutation $\hat{x}\hat{w} \mapsto x\phi_x(\hat{w})$ which already satisfies the condition $\exists_y \phi(\hat{x}\hat{w}) = y\phi_x(\hat{w})$. Then we are going to swap the values for some (disjoint) pairs of the arguments in order to fulfill the condition $\phi\left(\hat{1}\hat{w}\right) = \rho(\hat{w})\,\phi_1(\hat{w})$. Such swaps are preserving the condition of being permutation. Moreover we perform only such swaps that preserve also the $\exists_y \phi(\hat{x}\hat{w}) = y\phi_x(\hat{w})$ condition.

For every $\hat{w} \in [\hat{k}]^b$ we need to put $\phi\left(\hat{1}\hat{w}\right) = \rho(\hat{w})\,\phi_1(\hat{w})$. Let us assign $x = \rho(\hat{w})$ and $\hat{u} = \phi_x^{-1}(\phi_1(\hat{w}))$. Note that $\phi_x(\hat{u}) = \phi_1(\hat{w})$. If $x \neq 1$ then we can put $\phi(\hat{x}\hat{u}) = 1\phi_1(\hat{w})$. So we have swapped the values for the arguments $\hat{1}\hat{w}$ and $\hat{x}\hat{u}$. Our function is still a permutation. Note that the condition $\exists_y \phi(\hat{x}u) = y\phi_x\hat{u}$ is still preserved

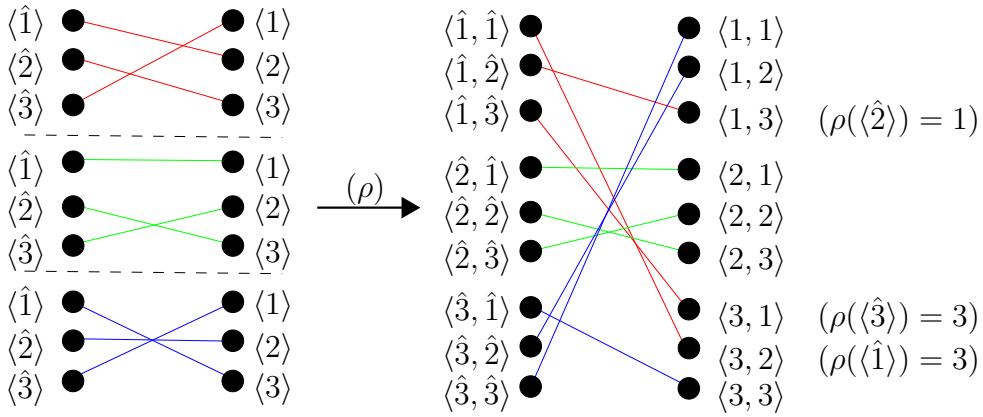

Figure 3.1: Merging three permutations (presented as perfect matchings) with respect to the function ρ such that $\rho\left(\langle\hat{1}\rangle\right) = 3$, $\rho\left(\langle\hat{2}\rangle\right) = 1$ and $\rho\left(\langle\hat{3}\rangle\right) = 3$.

because $\phi_x\left(\hat{u}\right) = \phi_1\left(\hat{w}\right)$. We just need to show that the swaps can be performed independently.

For every $i \in [k]$ a function $\phi_i^{-1} \circ \phi_1$ is a permutation so for every $\hat{w} \in \left[\hat{k}\right]^b$ the values of $\rho\left(\hat{w}\right) \phi_{\rho(\hat{w})}^{-1}\left(\phi_1\left(\hat{w}\right)\right)$ are pairwise different. Indeed for two different $\hat{u}, \hat{w} \in \left[\hat{k}\right]^b$ either the values $\rho\left(\hat{u}\right)$ and $\rho\left(\hat{w}\right)$ are different or $\rho\left(\hat{u}\right) = \rho\left(\hat{w}\right) = x$ for some $x \in [k]$ and then $\left(\phi_x^{-1} \circ \phi_1\right)\left(\hat{u}\right) \neq \left(\phi_x^{-1} \circ \phi_1\right)\left(\hat{w}\right)$ so then the values $\phi_{\rho(\hat{u})}^{-1}\left(\phi_1\left(\hat{u}\right)\right)$ and $\phi_{\rho(\hat{w})}^{-1}\left(\phi_1\left(\hat{w}\right)\right)$ are different. Therefore our pairs of the arguments to swap are pairwise disjoint. Thus all the swaps can be performed independently.

So for every $x \in [k]$ and $\hat{w} \in \left[\hat{k}\right]^b$ we have

$$\phi\left(\hat{x}\hat{w}\right) = \begin{cases} \rho\left(\hat{w}\right)\phi_1\left(\hat{w}\right) & \text{for } \hat{x} = \hat{1} \\ 1\phi_x\left(\hat{w}\right) & \text{for } \hat{x} \neq \hat{1} \wedge \rho\left(\phi_1^{-1}\left(\phi_x\left(\hat{w}\right)\right)\right) = x \\ x\phi_x\left(\hat{w}\right) & \text{in other cases.} \end{cases}$$

□

The intuition of the following lemma is that for every sequence $\hat{w} \in \left[\hat{k}\right]^b$ and for every $i \in [b]$ such that $\hat{w}_i = \hat{1}$ we can pick a value $\alpha\left(\hat{w}, i\right) \in [k]$ and there is a permutation $\phi : \left[\hat{k}\right]^b \to [k]^b$ such that for every sequence $\hat{w} \in \left[\hat{k}\right]^b$ and for every $i \in [b]$ such that $\hat{w}_i = \hat{1}$ we have $\phi\left(\hat{w}\right)_i = \alpha\left(\hat{w}, i\right)$.

Lemma 25. *Let $b \in \mathbb{N}$ and $\alpha : \left[\hat{k}\right]^b \times [b] \to [k] \cup \{\perp\}$ such that for every $\hat{w} \in \left[\hat{k}\right]^b$ and for every $i \in [b]$ holds $\alpha\left(\hat{w}, i\right) \neq \perp$ if and only if $\hat{w}_i = \hat{1}$. There is a permutation $\phi : \left[\hat{k}\right]^b \to [k]^b$ such that for every $\hat{w} \in \left[\hat{k}\right]^b$ and for every $i \in [b]$ if $\hat{w}_i = \hat{1}$ then $\phi\left(\hat{w}\right)_i = \alpha\left(\hat{w}, i\right)$.*

Proof. We will use an induction on b.

For $b = 0$ we have $\phi\left(\varepsilon\right) = \varepsilon$.

	$\langle \hat{1}_{(1)}, \hat{1}_{(2)} \rangle$	$\langle \hat{1}_{(3)}, \hat{2} \rangle$	$\langle \hat{2}, \hat{1}_{(4)} \rangle$	$\langle \hat{2}, \hat{2} \rangle$
$\langle 1_{(i)}, 1_{(ii)} \rangle$	$f(1, 1_{(i)}) + f(2, 1_{(ii)})$	$f(3, 1_{(i)})$	$f(4, 1_{(ii)})$	0
$\langle 1_{(i)}, 2_{(ii)} \rangle$	$f(1, 1_{(i)}) + f(2, 2_{(ii)})$	$f(3, 1_{(i)})$	$f(4, 2_{(ii)})$	0
$\langle 2_{(i)}, 1_{(ii)} \rangle$	$f(1, 2_{(i)}) + f(2, 1_{(ii)})$	$f(3, 2_{(i)})$	$f(4, 1_{(ii)})$	0
$\langle 2_{(i)}, 2_{(ii)} \rangle$	$f(1, 2_{(i)}) + f(2, 2_{(ii)})$	$f(3, 2_{(i)})$	$f(4, 2_{(ii)})$	0

Figure 3.2: The weights on the edges of a graph encoding an f-family for $f : [4] \times [2] \to \mathbb{N}$. The lower indices $(1), (2), (3)$ and (4) are added to indicate the correspondence between the occurrences of $\hat{1}$ and the elements of $[n]$ (the first argument of the function f). The lower indices (i) and (ii) are added to indicate the correspondence between the second argument of the function f and the position in the (two element) sequence $\langle \cdot, \cdot \rangle$.

For $b > 0$ we can define functions $\alpha_1, \alpha_2, \ldots, \alpha_k : [\hat{k}]^{b-1} \times [b-1] \to [k] \cup \{\perp\}$ such that for every $x \in [k]$ every $\hat{w} \in [\hat{k}]^{b-1}$ and every $i \in [b-1]$ we put $\alpha_x(\hat{w}, i) = \alpha(\hat{x}\hat{w}, i+1)$. From the inductive hypothesis for $b-1$ used for every function of $\alpha_1, \alpha_2, \ldots, \alpha_k$ we got the permutations $\phi_1, \phi_2, \ldots, \phi_k : [\hat{k}]^{b-1} \to [k]^{b-1}$ such that for every $x \in [k]$ for every $\hat{w} \in [\hat{k}]^{b-1}$ and for every $i \in [b-1]$ we have that if $\hat{w}_i = \hat{1}$ then $\phi_x(\hat{w})_i = \alpha_x(\hat{w}, i) = \alpha(\hat{x}\hat{w}, i+1)$.

Now we can use Lemma 24 to merge the permutations $\phi_1, \phi_2, \ldots, \phi_k$ using a function $\rho : [\hat{k}]^{b-1} \to [k]$ such that $\rho(\hat{w}) = \alpha(\hat{1}\hat{w}, 1)$ for every $\hat{w} \in [\hat{k}]^{b-1}$. We obtain one permutation $\phi : [\hat{x}]^b \to [x]^b$ such that by Lemma 24 (i) for every $\hat{x} \in [\hat{k}]$, for every $\hat{w} \in [\hat{k}]^{b-1}$ and for every $i \in [b-1]$ we have that $\phi(\hat{x}\hat{w})_{i+1} = \phi_x(\hat{w})_i$. So if $\hat{w}_i = \hat{1}$ then $\phi(\hat{x}\hat{w})_{i+1} = \phi_x(\hat{w})_i = \alpha(\hat{x}\hat{w}, i+1)$. Also by Lemma 24 (ii), for every $\hat{w} \in [\hat{k}]^{b-1}$ we have that $\phi(\hat{1}\hat{w})_1 = \rho(\hat{w}) = \alpha(\hat{1}\hat{w}, 1)$. So for every $\hat{w} \in [\hat{k}]^b$ and for every $i \in [b]$ we have that if $\hat{w}_i = \hat{1}$ then $\phi(\hat{w})_i = \alpha(\hat{w}, i)$, as required. □

Now we can describe the reduction.

Lemma 26. *For $k \geq 2$ and a function $f : [n] \times [k] \to \mathbb{N}$ there is a complete bipartite graph $G = (V_1 \cup V_2, E, w)$ such that*

- *for every $x \in X_f$ there exists a perfect matching M of G such that $w(M) = x$,*
- *for every perfect matching M of G we have $w(M) \in X_f$,*
- $|V_1| = |V_2| = O\left(\frac{nk^2 \log k}{\log n + \log k}\right)$.

Before the proof we present an informal idea. For some number b we define two sets of the vertices of our bipartite graph as $V_1 = [\hat{k}]^b$ and $V_2 = [k]^b$ i.e. as the sets of words of the length b. The number b is chosen in such a way that the sets V_1 and V_2 are small enough and at the same time b is large enough that the total number of occurrences of the character $\hat{1}$ in all the words in V_1 is at least n. Each such an

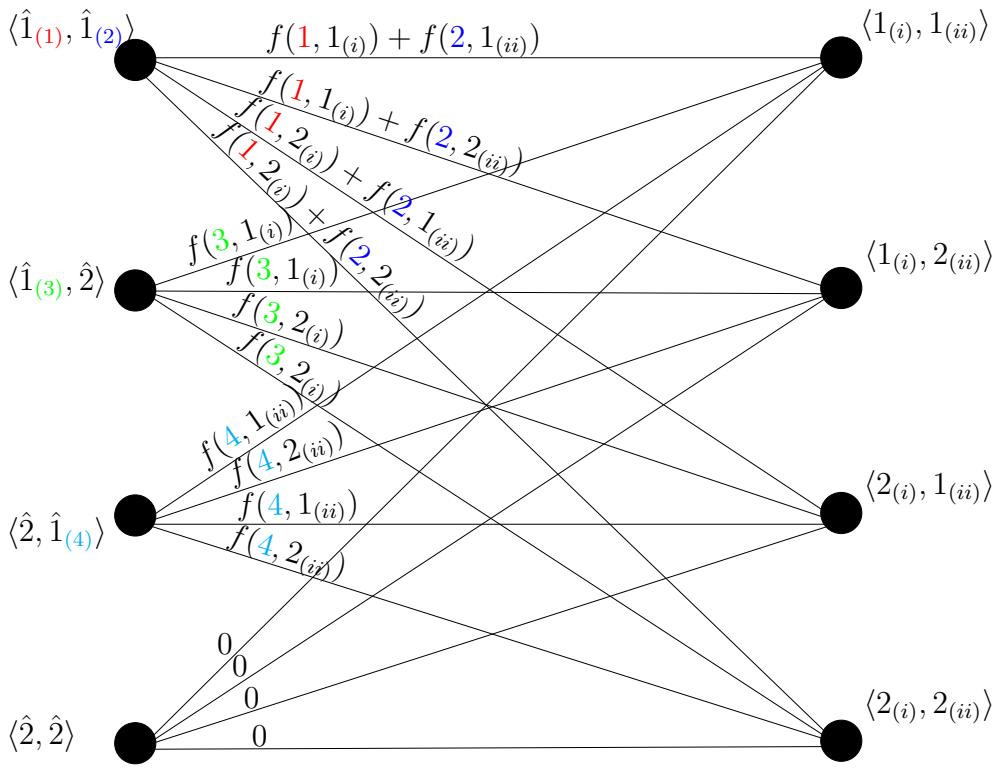

Figure 3.3: The graph encoding an f-family for $f : [4] \times [2] \to \mathbb{N}$. The lower indices $(1), (2), (3)$ and (4) are added to indicate the correspondence between the occurrences of $\hat{1}$ and the elements of $[n]$ (the first argument of the function f). The lower indices (i) and (ii) are added to indicate the correspondence between the second argument of the function f and the position in the (two element) sequence $\langle \cdot, \cdot \rangle$.

occurrence of $\hat{1}$ can be used to encode the values of f for one fixed first argument and therefore we need at least n of them. Then if $\hat{1}$ is the i-th character of some word $v_1 \in V_1$ and c is the i-th character of some word $v_2 \in V_2$ then if this occurrence of $\hat{1}$ encodes the values of f for some fixed first argument j then we add $f(j, c)$ to the weight of the corresponding edge in our bipartite graph. Now we can proceed to the formal proof of the lemma.

Proof. Let us consider the smallest $b \in \mathbb{N}_+$ such that $c = b \cdot k^{b-1} \geq n$. Later we will show that $|V_1| = |V_2| = k^b$ is sufficient. Through out the proof $|V_1| = |V_2| = k^b$.

For convenience we extend our chosen function $f : [n] \times [k] \to \mathbb{N}$ to $f : [c] \times [k] \to \mathbb{N}$ in such a way that for every $i = n+1, n+2, \ldots, c$ and for every $j \in [k]$ we put $f(i, j) = 0$. Note that the f-family X_f does not change after this extension.

Let $V_1 = [\hat{k}]^b$ and $V_2 = [k]^b$ be the sets of words of length b over the alphabets respectively $[\hat{k}]$ and $[k]$. Note that $|V_1| = |V_2| = k^b$. Let $\beta : V_1 \times [b] \to [c] \cup \{\perp\}$ be any function such that if $\hat{w}_j \neq \hat{1}$ then $\beta(\hat{w}, j) = \perp$ and every value from the set $[c]$ is used exactly once, i.e., for every $x \in [c]$ there is exactly one argument $(\hat{w}, j) \in V_1 \times [b]$ such

that $\beta(\hat{w}, j) = x$. Note that such a function always exists because the total number of the occurrences of $\hat{1}$ in all the words in V_1 is exactly $c = b \cdot k^{b-1}$.

Now we define our weight function $w : V_1 \times V_2 \to \mathbb{N}$ as follows

$$w\left(\hat{t}, u\right) = \sum_{\substack{i \in [b] \\ \beta\left(\hat{t}, i\right) \neq \perp}} f\left(\beta\left(\hat{t}, i\right), u_i\right).$$

An example of such weight function can be found in Figure 3.2 (or in Figure 3.3 as a picture of a bipartite graph).

Note that because β picks every value from the set $[c]$ exactly once, then $\beta(\hat{t}, i) \to \phi(\hat{t})_i$ defines a function, say σ, from $[c]$ to $[k]$. Then for every permutation $\phi : V_1 \to V_2$ we have

$$\sum_{\hat{t} \in V_1} w\left(\hat{t}, \phi\left(\hat{t}\right)\right) = \sum_{\hat{t} \in V_1} \sum_{\substack{i \in [b] \\ \beta\left(\hat{t}, i\right) \neq \perp}} f\left(\beta\left(\hat{t}, i\right), \phi\left(\hat{t}\right)_i\right) \in X_f.$$

In other words the set of the weights of all perfect matchings in G is a subset of X_f as required.

We also need to show that for every $x \in X_f$ there exists some permutation $\phi : V_1 \to V_2$ such that $\sum_{\hat{t} \in V_1} w\left(\hat{t}, \phi\left(\hat{t}\right)\right) = x$. This permutation gives us a corresponding perfect matching of weight x in G. Let us take a function $\sigma : [c] \to [k]$ such that $x = \sum_{i \in [c]} f\left(i, \sigma\left(i\right)\right)$, which exists by the definition of X_f. Define $\alpha : \left[\hat{k}\right]^b \times [b] \to [k] \cup \{\perp\}$ as follows:

$$\alpha\left(\hat{u}, i\right) = \begin{cases} \sigma\left(\beta\left(\hat{u}, i\right)\right) & \text{for } \hat{u}_i = \hat{1} \\ \perp & \text{for } \hat{u}_i \neq \hat{1}. \end{cases}$$

Now we can use Lemma 25 with function α to obtain a permutation $\phi : \left[\hat{k}\right]^b \to [k]^b$ such that for every $\hat{u} \in \left[\hat{k}\right]^b$ and for every $i \in [b]$ if $\hat{u}_i = \hat{1}$ then $\phi\left(\hat{u}\right)_i = \sigma\left(\beta\left(\hat{u}, i\right)\right)$. So we have that

$$\sum_{\hat{u} \in \left[\hat{k}\right]^b} w\left(\hat{u}, \phi\left(\hat{u}\right)\right) =$$

$$= \sum_{\hat{u} \in \left[\hat{k}\right]^b} \sum_{\substack{i \in [b] \\ \beta(\hat{u}, i) \neq \perp}} f\left(\beta\left(\hat{u}, i\right), \sigma\left(\beta\left(\hat{u}, i\right)\right)\right) =$$

$$= \sum_{i \in [c]} f\left(i, \sigma\left(i\right)\right) = x.$$

Hence we have shown that X_f is the set of weights of all perfect matchings in graph G. The last thing is to show that the number of the vertices is sufficiently small. Since $bk^b \geq nk$ we must have $b > \log_k(nk / \log_k(nk)) \sim \log_k(nk)$. Then, since $(b-1)k^{b-2} < n$ we must have $k^b < nk^2/(b-1) \leq (1 + o(1))nk^2 / \log_k(nk) = O\left(\frac{nk^2}{\log_k nk}\right) = O\left(\frac{nk^2 \log k}{\log n + \log k}\right)$. So $|V_1| = |V_2| = k^b = O\left(\frac{nk^2 \log k}{\log n + \log k}\right)$, as required. □

Lemma 26 immediately implies the following result.

Lemma 27. *There is a polynomial time reduction that for an instance $I = (f, g)$ of* Family Intersection *with $f : [a] \times [b] \to \mathbb{N}$ and $g : [c] \times [d] \to \mathbb{N}$ reduces it into an instance of* Equal Weight Matchings *$J = (G_1, G_2)$ with $|V(G_1)| = O\left(\frac{ab^2 \log b}{\log a + \log b}\right)$ and $|V(G_2)| = O\left(\frac{cd^2 \log d}{\log c + \log d}\right)$ vertices. The sets of the weights of all perfect matchings in G_1 and in G_2 are equal respectively to X_f and X_g.*

Together with Lemma 23 we obtain the following theorem.

Theorem 28. *There is a polynomial time reduction from a given instance of* 3-CNF-SAT *with n variables and m clauses into an instance of* Equal Weight Matchings *with $|V(G_1)| = O\left(\frac{n}{\log n}\right)$, $|V(G_2)| = O\left(\frac{m}{\log m}\right)$ and the maximum matching weights bounded by 2^{3m}.*

Using Theorem 15 we can prove the following lower bound.

Theorem 2. *Unless ETH fails, there is no algorithm solving* Equal Weight Matchings *in time $2^{o(n \log n)} \cdot r^{O(1)}$ where n is the total number of vertices, and r is the bit size of the input.*

Proof. For a given instance of 3-CNF-SAT with n variables and m clauses we can use the reduction from Theorem 28 to obtain an instance of Equal Weight Matchings. The total number of the vertices in the new instance is

$$|V(G_1)| + |V(G_2)| = O\left(\frac{n}{\log n} + \frac{m}{\log m}\right) =$$

$$= O\left(\frac{n+m}{\log(n+m)} + \frac{n+m}{\log(n+m)}\right) = O\left(\frac{n+m}{\log(n+m)}\right)$$

because the function $\frac{n}{\log n}$ is nondecreasing for the sufficiently big values of n. Weights of the matchings are bounded by 2^{3m} and the bit size of the instance

$$r = O\left(\left(\frac{n+m}{\log(n+m)}\right)^2 \log 2^{3m}\right) = nm^{O(1)}.$$

Then let us assume that there is an algorithm solving Equal Weight Matchings in $2^{o(n \log n)} r^{O(1)}$-time. Then we could solve our instance of 3-CNF-SAT in time

$$2^{o\left(\frac{n+m}{\log(n+m)} \log\left(\frac{n+m}{\log(n+m)}\right)\right)} nm^{O(1)^{O(1)}} =$$

$$= 2^{o\left(\frac{n+m}{\log(n+m)} \log(n+m)\right)} nm^{O(1)} = 2^{o(n+m)}$$

which contradicts ETH by Theorem 15. □

3.2 Hardness of Channel Assignment

Consider two weighted complete bipartite graphs G_1 and G_2. We would like to encode them in a Channel Assignment instance in such a way that this Channel Assignment instance is a YES-instance if and only if there are two perfect matchings, one in G_1 and the other in G_2, of the same weight.

Because Channel Assignment is a natural generalization of a classical graph coloring problem we will refer to the assignments as colorings. It is very convenient because we then can refer to the value assigned to a vertex v as to the *color* of v.

Consider an instance $I = (V, d, s)$ of Channel Assignment where d is our weight function and s is a maximum allowed span. We say that $c : V \to \mathbb{Z}$ is a YES-coloring if c is a proper coloring and has span at most s. Note that an instance of Channel Assignment is a YES-instance if and only if it has a YES-coloring.

Our approach is that we encode those graphs G_1 and G_2 separately in such a way that we have a special vertex v_M whose color in every YES-coloring represents a weight of some perfect matching in G_1 and on the other hand in every YES-coloring its color represents (in a similar way) a weight of some perfect matching in G_2. So a YES-coloring would be possible if and only if the graphs G_1 and G_2 have two perfect matchings, one in G_1 and the other in G_2, with equal weights.

Before we present a way to encode a weighted complete bipartite graph in a Channel Assignment instance we would like to present the two lemmas to merge those two encoded graphs into a one instance of Channel Assignment. In order to do that we use the following concepts.

We say that instance I is (x, y)-*spanned* for some vertices $x, y \in V$ if for every YES-coloring c of I we have $|c(x) - c(y)| = s - 1$. We say that an instance $I = (V, d, s)$ of Channel Assignment is (X, Y)-*spanned* for some nonempty subsets of the vertices $\emptyset \neq X, Y \subseteq V$ if it is (x, y)-spanned for every two vertices $x \in X$ and $y \in Y$.

Lemma 29. *For every (u, v)-spanned instance $I_1 = (V_1, d_1, s)$ and (w, z)-spanned instance $I_2 = (V_2, d_2, s)$ of* Channel Assignment *there is a $(\{u, w\}, \{v, z\})$-spanned instance $I = (V_1 \cup V_2, d, s)$ of* Channel Assignment *such that*

(i) for every YES-coloring c of I the coloring $c|_{V_1}$ is a YES-coloring of I_1 and the coloring $c|_{V_2}$ is a YES-coloring of I_2,

(ii) for every YES-coloring c_1 of I_1 and every YES-coloring c_2 of I_2 such that $c_1(u) = c_2(w)$, $c_1(v) = c_2(z)$ and for every $x \in V_1 \cap V_2$ we have $c_1(x) = c_2(x)$ there exists a YES-coloring c of I such that $c|_{V_1} = c_1$ and $c|_{V_2} = c_2$.

Proof. Let $B = \{u, w\} \times \{v, z\} \cup \{v, z\} \times \{u, w\}$ and let

$$d(x,y) = \begin{cases} s-1 & \text{if } (x,y) \in B \\ \max\{d_1(x,y), d_2(x,y)\} & \text{if } x,y \in V_1 \cap V_2 \\ d_1(x,y) & \text{if } x,y \in V_1 \text{ and} \\ & \{x,y\} \not\subseteq V_1 \cap V_2 \\ d_2(x,y) & \text{if } x,y \in V_2 \text{ and} \\ & \{x,y\} \not\subseteq V_1 \cap V_2 \\ 0 & \text{otherwise.} \end{cases}$$

Our instance is $(\{u, w\}, \{v, z\})$-spanned because for all the pairs in B we set the minimum allowed distance to at least $s-1$. Note that for $i = 1, 2$, for every $x, y \in V_i$ we have $d(x,y) \geq d_i(x,y)$. Hence every proper coloring c of I has the property that $c\,|_{V_1}$ is a proper coloring of I_1 and $c\,|_{V_2}$ is a proper coloring of I_2. Also the maximum allowed spans of I, I_1, I_2 are the same, so for every YES-coloring c of I coloring $c\,|_{V_1}$ is a YES-coloring of I_1 and $c\,|_{V_2}$ is a YES-coloring of I_2. Hence (i) is clear.

For (ii), consider a YES-coloring c_1 of I_1 and a YES-coloring c_2 of I_2 such that $c_1(u) = c_2(w)$ and $c_1(v) = c_2(z)$ and such that for every $x \in V_1 \cap V_2$ we have $c_1(x) = c_2(x)$. Then we define a coloring c

$$c(x) = \begin{cases} c_1(x) & \text{if } x \in V_1 \\ c_2(x) & \text{if } x \in V_2. \end{cases}$$

We know that $c_1(u) = c_2(w)$ and $c_1(v) = c_2(z)$ so all the vertices of $V_1 \cup V_2$ have colors between $c_1(u)$ and $c_1(v)$, i.e., the span of c is at most s as required. It is straightforward to check that c is a proper coloring. □

Lemma 30. *For every (v_L, v_R)-spanned instance $I = (V, d, s)$ of* Channel Assignment *and for every numbers $l, r \in \mathbb{N}$ there exists a (w_L, w_R)-spanned instance $I' = (V \cup \{w_L, w_R\}, d', l + s + r)$ such that*

(i) for every YES-coloring c of I there is a YES-coloring c' of I' such that $c'\,|_V = c$,

(ii) for every YES-coloring c' of I' such that $c'(w_L) \leq c'(w_R)$ we have that

- *a coloring $c'\,|_V$ is a YES-coloring of I,*
- *$c'(v_L) = c'(w_L) + l$ and $c'(v_R) = c'(w_R) - r$.*

Proof. We assume that $w_L, w_R \notin V$. We put

$$d'(x,y) = \begin{cases} l+s-1+r & \text{for } \{x,y\} = \{w_L, w_R\} \\ l & \text{for } \{x,y\} \cap \{w_L, w_R\} = \\ & \{w_L\} \\ r & \text{for } \{x,y\} \cap \{w_L, w_R\} = \\ & \{w_R\} \\ d(x,y) & \text{for } x, y \in V. \end{cases}$$

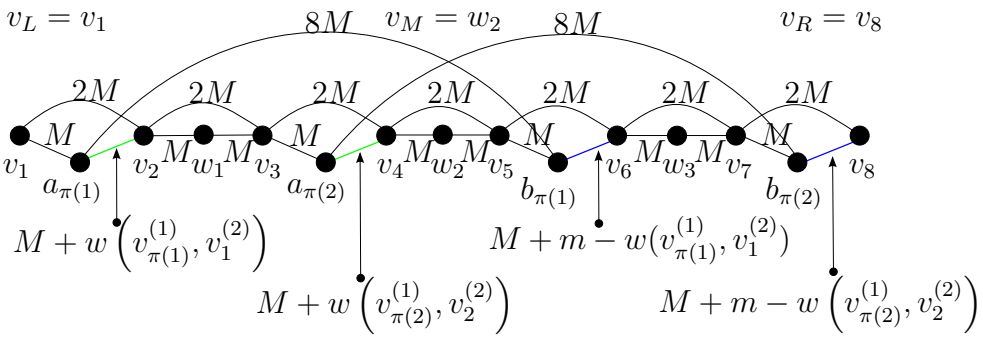

Figure 3.4: A weighted complete bipartite graph (G, w) with $|V_1| = |V_2| = 2$ encoded in a Channel Assignment form. The color of the vertex $v_M = w_2$ corresponds to the weight of the perfect matching in G given by the permutation π and is equal to $c(v_M) = c(v_L) + 7M + w(M_\pi)$. The picture is simplified. Some of the edges and corresponding to them minimum distances are omitted in the picture.

It is straightforward to check that d' satisfies (i) and (ii). □

Lemma 31. *Let $G = (V_1 \cup V_2, E, w)$ be a weighted complete bipartite graph with nonnegative weights and such that $|V_1| = |V_2|$. Let $n = |V_1|$, $m = \max_{e \in E} w(e)$, $M = n \cdot m + 1$, $l = (4n-1) \cdot M$ and $s = (8n-1) \cdot M$. There exists a (v_L, v_R)-spanned instance $I = (V, d, s)$ of* Channel Assignment *with $|V| = O(n)$ and such that for some vertex $v_M \in V$,*

(i) for every YES-coloring c of I such that $c(v_L) \leq c(v_R)$ there exists a perfect matching M_G in G such that $c(v_M) = c(v_L) + l + w(M_G)$ and

(ii) for every perfect matching M_G in G there exists a YES-coloring c of I such that $c(v_L) \leq c(v_R)$ and $c(v_M) = c(v_L) + l + w(M_G)$.

Proof. Let $V_1 = \left\{v_1^{(1)}, v_2^{(1)}, \ldots, v_n^{(1)}\right\}$ and $V_2 = \left\{v_1^{(2)}, v_2^{(2)}, \ldots, v_n^{(2)}\right\}$. We will build our Channel Assignment instance step by step. A simplified picture of the instance can be found in Figure 3.4. The outline of the proof is following:

- We will define the vertices of our Channel Assignment instance and we will force some properties of its colorings by putting appropriate weights on the edges. We specify those properties as Claims.
- We start with defining the vertices $v_L = v_1, v_2, \ldots, v_{4n} = v_R$. These vertices will be used as a backbone of our Channel Assignment instance i.e. we force them to be colored in a strictly ascending way (or strictly descending, but we assume the ascending order without loss on generality). All the following groups of the vertices that we will add later will be colored in relation to the colors of $v_1, v_2, \ldots, v_{4n}$.

Claim 1 For every YES-coloring c and for every $i < j$ we have that $c(v_i) < c(v_j)$.

- Then we add the vertices $w_1, w_2, \ldots, w_{2n-1}$ and interleave them with the vertices $v_1, v_2, \ldots, v_{4n}$.

  Claim 2 For every YES-coloring c the colors of the vertices in the sequence

  $$v_1, v_2, w_1, v_3, v_4, w_2, v_5 \ldots, v_{4n-2}, w_{2n-1}, v_{4n-1}, v_{4n}$$

  are increasing.

- Next we add the vertices $a_1, a_2, \ldots, a_n$ and interleave them with the previous vertices, but in an arbitrary order π_c.

  Claim 3 For every YES-coloring c there is a permutation π_c such that the colors of the vertices of the sequence

  $$v_1, a_{\pi_c(1)}, v_2, w_1, v_3, a_{\pi_c(2)}, v_4, w_2, v_5 \ldots, v_{2n-1}, a_{\pi_c(n)}, v_{2n}$$

  are increasing.

- Similarly we add vertices $b_1, b_2, \ldots, b_n$ in an arbitrary order ρ_c.

  Claim 4 For every YES-coloring c there is a permutation ρ_c such that the colors of the vertices in the sequence

  $$v_{2n+1}, b_{\rho_c(1)}, v_{2n+2}, w_{n+1}, v_{2n+3}, b_{\rho_c(2)}, v_{2n+4}, w_{n+2},$$

  $$v_{2n+5} \ldots v_{4n-1}, b_{\rho_c(n)}, v_{4n}$$

  are increasing.

- Then we force that the vertices $a_1, a_2, \ldots, a_n$ and the vertices $b_1, b_2, \ldots, b_n$ are colored in exactly the same (arbitrary) order i.e. the order π_c is exactly the same as the order ρ_c.

  Claim 5 For every YES-coloring c there is a permutation π_c such that the colors of the vertices in the sequence

  $$v_1, a_{\pi_c(1)}, v_2, w_1, v_3, a_{\pi_c(2)}, v_4, w_2, v_5 \ldots, v_{2n-1}, a_{\pi_c(n)},$$

  $$v_{2n}, w_n, v_{2n+1}, b_{\pi_c(1)}, v_{2n+2}, w_{n+1}, v_{2n+3}, b_{\pi_c(2)}, v_{2n+4},$$

  $$w_{n+2}, v_{2n+5} \ldots v_{4n-1}, b_{\pi_c(n)}, v_{4n}$$

  are increasing.

- Then, with all the above construction specified, we state the dependency between the set of all the possible colors of the vertex v_M and the set of the possible weights of the matchings in our bipartite graph.

  Claim 6 Let $\pi : [n] \to [n]$ be any permutation and

  $$M_\pi = \left\{ v_i^{(1)} v_{\pi(i)}^{(2)} : i \in [n] \right\}$$

  be the corresponding perfect matching in G. There is exactly one YES-coloring c such that $\pi_c = \pi$. Moreover $c(v_M) = c(v_L) + l + w(M_\pi)$.

Now we can present the proof in a formal way. Let us introduce the vertices $v_L = v_1, v_2, \ldots, v_{4n} = v_R$ to set V. Because of the symmetry of colorings we can assume that for every coloring c of our instance we have $c(v_L) \leq c(v_R)$. Indeed we know that for every coloring c a symmetric coloring $c'(v) = 1 + \text{span}(c) - c(v)$ has the same span as c and is proper if and only if c is proper.

We set the minimum distance $d(v_L, v_R) = s - 1$. Then for every YES-coloring c we have that $|c(v_L) - c(v_R)| = s - 1$ so our instance is (v_L, v_R)-spanned. For every $i, j \in [4n]$ such that $i \neq j$ and $\{i, j\} \neq \{1, 4n\}$ we set the minimum distance $d(v_i, v_j) = |i - j| \cdot 2M$. Then we can prove the following claim.

Claim 1 For every YES-coloring c and for every $i < j$ we have that $c(v_i) < c(v_j)$.

Proof of the claim: We have assumed w.l.o.g. that for all colorings we have $c(v_L) \leq c(v_R)$. Note that for $i \neq j$ we have $c(v_i) \neq c(v_j)$. If there are $i < j$ such that $c(v_j) < c(v_i)$ then $c(v_R) - c(v_L) = c(v_R) - c(v_i) + c(v_i) - c(v_j) + c(v_j) - c(v_L) \geq 2M \times ((4n - i) + (j - i) + (j - 1)) \geq 2M \times (4n - 1 + 2(j - i)) \geq 2M \times (4n + 1) > s$ a contradiction. This proves the claim.

Note that for every YES-coloring and for every $i \in [4n - 1]$ we have

$$2M \leq c(v_{i+1}) - c(v_i) \leq 2M + n \cdot m < 3M, \tag{3.1}$$

otherwise $c(v_R) - c(v_L) \geq (4n - 2) \cdot 2M + 2M + n \cdot m + 1 = s$, so c has span at least $s + 1$, a contradiction.

Let us introduce new vertices $w_1, w_2, \ldots, w_{2n-1}$ to set V. For every $i \in [2n - 1]$ and $j \in [4n]$ we set the minimum distances $d(w_i, v_j) = |4i + 1 - 2j| \cdot M$. For every YES-coloring c and for every $i \in [2n - 1]$ we have $c(v_{2i}) + M \leq c(w_i) \leq c(v_{2i+1}) - M$ by (3.1), otherwise we have that $c(v_j) \leq c(w_i) \leq c(v_{j+1})$ for some $j \neq 2i$ (because $c(v_{4n}) - c(v_1) = s - 1$ so every YES-coloring uses only colors from the interval $[c(v_1), c(v_{4n})]$) and then $c(v_{j+1}) - c(v_j) \geq d(v_j, w_i) + d(w_i, v_{j+1})$ and $\{v_j, v_{j+1}\} \neq \{v_{2i}, v_{2i+1}\}$ so at least one of these two distances is at least $3M$ and therefore $c(v_{j+1}) - c(v_j) \geq 3M + M > 3M$, a contradiction with (3.1). Thus infer the following claim.

Claim 2 For every YES-coloring c the colors of the vertices in the sequence

$$v_1, v_2, w_1, v_3, v_4, w_2, v_5 \ldots, v_{4n-2}, w_{2n-1}, v_{4n-1}, v_{4n} \tag{3.2}$$

are increasing.

We introduce new vertices $a_1, a_2, \ldots, a_n$ and for every $i \in [n]$ and $j \in [4n]$ we set the minimum distances

$$d(a_i, v_j) = \begin{cases} M + w\left(v_i^{(1)}, v_{j/2}^{(2)}\right) & \text{when } j \leq 2n \text{ and } 2 \mid j \\ M & \text{when } j \leq 2n \text{ and } 2 \nmid j \\ (j-2n) \cdot 2M + M & \text{when } j > 2n. \end{cases}$$

Then for every YES-coloring c and for every $i \in [n]$ we have $c(a_i) \leq c(v_{2n})$ because in other case we have $c(v_j) \leq c(a_i) \leq c(v_{j+1})$ for some $j \geq 2n$ and then $c(v_{j+1}) - c(v_j) \geq d(v_j, a_i) + d(a_i, v_{j+1}) \geq M + 3M > 3M$, a contradiction with (3.1).

Moreover for every $i \in [n]$ and every $j \in [2n-1]$ we set the minimum distance $d(a_i, w_j) = 2M$. Therefore by (3.1) and (3.2) for every YES-coloring c and every $i \in [n]$ the vertex a_i is colored with the color from one of the intervals $(c(v_{2j-1}), c(v_{2j}))$ for some $j \in [n]$.

Finally for every $i, j \in [n]$ such that $i \neq j$ we set the minimum distance $d(a_i, a_j) = 4M$ so by (3.1) we know that for every YES-coloring c and every $i \in [n]$ exactly one one vertex a_j of the vertices $a_1, a_2, \ldots, a_n$ is colored with the color from the interval $(c(v_{2i-1}), c(v_{2i}))$. The assignment of vertices $a_1, a_2, \ldots, a_n$ to intervals $(c(v_1), c(v_2))$, $(c(v_3), c(v_4)), \ldots, (c(v_{2n-1}), c(v_{2n}))$ determines a permutation $\pi_c : [n] \to [n]$, i.e., $\pi_c(i) = j$ if a_j gets a color from $(c(v_{2i-1}), c(v_{2i}))$. Hence we get the following claim:

Claim 3 For every YES-coloring c there is a permutation π_c such that the colors of the vertices of the sequence

$$v_1, a_{\pi_c(1)}, v_2, w_1, v_3, a_{\pi_c(2)}, v_4, w_2, v_5 \ldots, v_{2n-1}, a_{\pi_c(n)}, v_{2n}$$

are increasing.

Similarly we introduce new vertices $b_1, b_2, \ldots, b_n$ and for every $i \in [n]$ and $j \in [4n]$ we set the minimum distances

$$d(b_i, v_j) = \begin{cases} (2n-j+1) \cdot 2M + M & \text{when } j \leq 2n \\ M + m - w\left(v_i^{(1)}, v_{j/2-n}^{(2)}\right) & \text{when } j > 2n \text{ and } 2 \mid j \\ M & \text{when } j > 2n \text{ and } 2 \nmid j. \end{cases}$$

Also for every $i \in [n]$ and every $j \in [2n-1]$ we set the minimum distance $d(b_i, w_j) = 2M$ and for every $i, j \in [n]$ such that $i \neq j$ we set the minimum distance $d(b_i, b_j) = 4M$. Hence similarly as before, for every YES-coloring c and every $i \in [n]$ exactly one vertex b_j of the vertices $b_1, b_2, \ldots, b_n$ is colored with the color from the interval $(c(v_{2n+2i-1}), c(v_{2n+2i}))$. Analogously as before, the colors of the vertices $b_1, b_2, \ldots, b_n$

determine a permutation $\rho_c : [n] \to [n]$. Thus we have the following claim.

Claim 4 For every YES-coloring c there is a permutation ρ_c such that the colors of the vertices in the sequence

$$v_{2n+1}, b_{\rho_c(1)}, v_{2n+2}, w_{n+1}, v_{2n+3}, b_{\rho_c(2)}, v_{2n+4}, w_{n+2},$$

$$v_{2n+5} \ldots v_{4n-1}, b_{\rho_c(n)}, v_{4n}$$

are increasing.

For every $i \in [n]$ we set the minimum distance $d(a_i, b_i) = n \cdot 4M$. Then we know that for every YES-coloring c we have $\pi_c^{-1}(i) \leq \rho_c^{-1}(i)$ otherwise we can take $j = 2\pi_c^{-1}(i) - 1$ and $k = 2n + 2\rho_c^{-1}(i)$ and then $(c(b_i) - c(a_i)) + 2M \leq c(v_j) - c(v_k)$ and $k - j \leq 2n$ so the sequence $v_1, v_2, \ldots, v_j, v_k, \ldots, v_{4n}$ has at least $4n - 2n + 1 = 2n + 1$ elements so $c(v_{4n}) - c(v_1) \geq (2n-1) \cdot 2M + (c(v_k) - c(v_j)) \geq (2n-1) \cdot 2M + (c(b_i) - c(a_i)) + 2M \geq (2n-1) \cdot 2M + n \cdot 4M + 2M = n \cdot 8M > (n-1) \cdot 8M - 1 = s - 1$, a contradiction. Since π_c and ρ_c are permutations, we further infer that for every YES-coloring c we have $\pi_c = \rho_c$. Hence we have the following claim.

Claim 5 For every YES-coloring c there is a permutation π_c such that the colors of the vertices in the sequence

$$v_1, a_{\pi_c(1)}, v_2, w_1, v_3, a_{\pi_c(2)}, v_4, w_2, v_5 \ldots, v_{2n-1}, a_{\pi_c(n)},$$

$$v_{2n}, w_n, v_{2n+1}, b_{\pi_c(1)}, v_{2n+2}, w_{n+1}, v_{2n+3}, b_{\pi_c(2)}, v_{2n+4},$$

$$w_{n+2}, v_{2n+5} \ldots v_{4n-1}, b_{\pi_c(n)}, v_{4n}$$

are increasing.

This ends the description of the instance I. Note that I is (v_L, v_R)-spanned because $d(v_L, v_R) = s - 1$. Let us put $v_M = w_n$. We are going to show the following claim.

Claim 6 Let $\pi : [n] \to [n]$ be any permutation and $M_\pi = \left\{ v_i^{(1)} v_{\pi(i)}^{(2)} : i \in [n] \right\}$ be the corresponding perfect matching in G. There is exactly one YES-coloring c such that $\pi_c = \pi$. Moreover $c(v_M) = c(v_L) + l + w(M_\pi)$.

Proof of the claim: Let us consider a sequence of the vertices

$$v_1, a_{\pi(1)}, v_2, w_1, v_3, a_{\pi(2)}, v_4, w_2, v_5 \ldots, v_{2n-1}, a_{\pi(n)},$$

$$v_{2n}, w_n, v_{2n+1}, b_{\pi(1)}, v_{2n+2}, w_{n+1}, v_{2n+3}, b_{\pi(2)}, v_{2n+4},$$

$$w_{n+2}, v_{2n+5} \ldots v_{4n-1}, b_{\pi(n)}, v_{4n}$$

and the coloring c implied by the minimum distances of pairs of consecutive elements in this sequence, i.e., $c(v_1) = 1$, $c\left(a_{\pi(1)}\right) = c(v_1) + d\left(v_1, a_{\pi(1)}\right)$, $c(v_2) = c\left(a_{\pi(1)}\right) + d\left(a_{\pi(1)}, v_2\right)$, $c(w_1) = c(v_2) + d(v_2, w_1)$, $c(v_3) = c(w_1) + d(w_1, v_3)$, $\ldots$, $c(v_{4n}) = c\left(b_{\pi(n)}\right) + d\left(b_{\pi(n)}, v_{4n}\right)$. We need to check that all the minimum distance constraints d are satisfied and that the span of this coloring is not greater than s.

Note that for every $i \in [4n]$ and for every vertex $x \in V$ such that $v_i \neq x$ we have $d(x, v_i) \geq M$. Therefore for every $i \in [4n-1]$ we have $c(v_{i+1}) - c(v_i) = (c(v_{i+1}) - c(x)) + (c(x) - c(v_i)) = d(x, v_{i+1}) + d(v_i, x) \geq 2M$ where x is the vertex separating v_i and v_{i+1} in the sequence. Thus for every $i, j \in [4n]$ we have $|c(v_i) - c(v_j)| \geq |i-j| \cdot 2M$ so if $\{i, j\} \neq \{1, 4n\}$ then $|c(v_i) - c(v_j)| \geq d(v_i, v_j)$. Hence also for every $i \in [2n-1]$ and $j \in [4n]$ we have $|c(w_i) - c(v_j)| = |c(w_i) - c(v_k)| + |c(v_k) - c(v_j)| \geq M + |k-j| \cdot 2M$ where in case that $j \leq 2i$ we have $k = 2i$ and in this case $M + |k-j| \cdot 2M = |4i - 2j + 1| \cdot M$ and in case that $j > 2i$ we have $k = 2i + 1$ and in this case $M + |k-j| \cdot 2M = |2j - 4i - 1| \cdot M = |4i - 2j + 1| \cdot M$ so in both cases $|c(w_i) - c(v_j)| \geq |4i - 2j + 1| \cdot M = d(w_i, v_j)$. We will check the distance between $v_L = v_1$ and $v_R = v_{4n}$ later.

For every $i \in [n]$ and vertex $a_{\pi(i)}$ the closest vertex v_j to the left is v_{2i-1} and to the right is v_{2i}. They are immediate neighbors of $a_{\pi(i)}$ in the sequence so from the definition of c we have $|c(a_{\pi(i)}) - c(v_{2i-1})| = d(v_{2i-1}, a_{\pi(i)})$ and $|c(v_{2i}) - c(a_{\pi(i)})| = d(a_{\pi(i)}, v_{2i})$. Note that for every $j \in [2n]$ we have $d(a_{\pi(i)}, v_j) \leq 2M$ and then for every $j \in [2i-2]$ we have $|c(a_{\pi(i)}) - c(v_j)| = (c(v_{2i-1}) - c(v_j)) + (c(a_{\pi(i)}) - c(v_{2i-1})) \geq 2M + M > d(a_{\pi(i)}, v_{2i-1})$. Similarly for every $2i+1 \leq j \leq 2n$ we have $|c(a_{\pi(i)}) - c(v_j)| = ((c(v_{2i}) - c(a_{\pi(i)})) + (c(v_j) - c(v_{2i})) \geq M + 2M > d(a_{\pi(i)}, v_j)$. For every $2n+1 \leq j \leq 4n$ we have $|c(v_j) - c(a_{\pi(i)})| = (c(v_{2i}) - c(a_{\pi(i)})) + (c(v_{2n}) - c(v_{2i})) + (c(v_j) - c(v_{2n})) \geq M + 0 + (j - 2n) \cdot 2M = d(a_{\pi(i)}, v_j)$. Because π is a permutation thus we obtain that for every $i \in [n]$ and for every $j \in [4n]$ we have $|c(a_i) - c(v_j)| \geq d(a_i, v_j)$.

For every $i \in [n]$ and $j \in [2n-1]$ there is at least one vertex v_k with color between the colors $c(a_i)$ and $c(w_j)$ so $|c(a_i) - c(w_j)| = |c(a_i) - c(v_k)| + |c(v_k) - c(w_j)| \geq 2M = d(a_i, w_j)$. For every $i, j \in [n]$ such that $\pi^{-1}(i) < \pi^{-1}(j)$ there are at least two vertices v_k, v_{k+1} with colors $c(a_i) \leq c(v_k) \leq c(v_{k+1}) \leq c(a_j)$. Therefore $|c(a_j) - c(a_i)| = (c(v_k) - c(a_i)) + (c(v_{k+1}) - c(v_k)) + (c(a_j) - c(v_{k+1})) \geq M + 2M + M = 4M = d(a_i, a_j)$.

Similarly we can check that for every $i \in [n]$ and $j \in [4n]$ we have $|c(b_i) - c(v_j)| \geq d(b_i, v_j)$, that for every $i \in [n]$ and $j \in [2n-1]$ we have $|c(b_i) - c(w_j)| \geq d(b_i, w_j)$ and for every $i, j \in [n]$ such that $i \neq b$ we have $|c(b_i) - c(b_j)| \geq d(b_i, b_j)$.

We need also to check that for every $i \in [n]$ we have $|c(a_i) - c(b_i)| \geq n \cdot 4M = d(a_i, b_i)$. Indeed $|c(b_i) - c(a_i)| = (c(v_{2i}) - c(a_i)) + (c(v_{2n+2i-1}) - c(v_{2i})) + (c(b_i) - c(v_{2n+2i-1})) \geq M + (2n-1) \cdot 2M + M = n \cdot 4M = d(a_i, b_i)$.

Now we are going to deal with the distances between v_L, v_M and v_R. The sum of the minimum color distances of neighboring elements in the prefix of our sequence:

$$v_1, a_{\pi(1)}, v_2, w_1, v_3, a_{\pi(2)}, v_4, w_2, v_5 \ldots, v_{2n-1}, a_{\pi(n)}, v_{2n},$$

$$w_n, v_{2n+1}$$

is exactly $2n \cdot 2M + w(M_\pi)$. The sum of the minimum color distances of neighboring elements in the suffix of our sequence:

$$v_{2n+1}, b_{\pi(1)}, v_{2n+2}, w_{n+1}, v_{2n+3}, b_{\pi(2)}, v_{2n+4}, w_{n+2},$$

$$v_{2n+5} \ldots v_{4n-1}, b_{\pi(n)}, v_{4n}$$

is exactly $(2n-1) \cdot 2M + n \cdot m - w(M_\pi)$. So the total sum for the whole sequence is exactly $(4n-1) \cdot 2M + n \cdot m = s - 1$ and it does not depend on the permutation π.

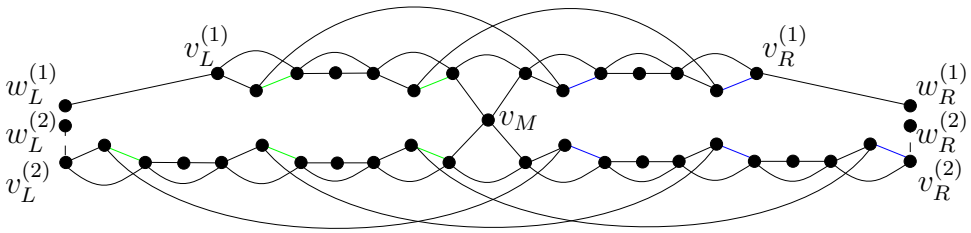

Figure 3.5: Two weighted complete bipartite graphs (G_1, w_1) (with $n_1 = 2$) and (G_2, w_2) (with $n_2 = 3$) encoded in a Channel Assignment form. The color of the vertex $v_M = w_2$ corresponds to the weight of some perfect matching in G_1 and to the weight of some perfect matching in G_2. These two weights have to be equal. The picture is simplified. Some of the edges are omitted in the picture. Note that the values M and m can be different for (G_1, w_1) and for (G_2, w_2).

Therefore $|c(v_R) - c(v_L)| = s - 1 = d(v_L, v_R)$. This was the last constraint to check and hence we have shown that c is proper. On the other hand the span of c is s so c is a YES-coloring. Moreover we have $c(v_M) = c(v_L) + (4n-1) \cdot M + w(M_{\pi_c}) = c(v_L) + l + w(M_{\pi_c})$. Note that all the distances of pairs of consecutive elements of (the whole) sequence are tight, i.e., these distances are equal to the minimum allowed distances for these pairs of the vertices and therefore we cannot decrease any of these distances. On the other hand the span of c is maximum so we cannot increase any of these distances without exceeding the maximum span or violating some of the constraints provided by d. Therefore c is the only one YES-coloring for which the colors of the vertices of this sequence are increasing. Hence c is the only one YES-coloring such that $\pi_c = \pi$. This ends the proof of the claim.

Thus there is a one-to-one correspondence between permutations and YES-colorings. Moreover we know that for every YES-coloring c we have $c(v_M) = c(v_L) + l + w(M_{\pi_c})$ where M_{π_c} is the perfect matching in G corresponding to permutation π_c. Hence we have shown (i) and (ii) as required. □

Lemma 32. *There is a polynomial time reduction such that for a given instance $I = (G_1, G_2)$ of* Equal Weight Matchings *with $n_1 = |V(G_1)|, n_2 = |V(G_2)|$ and such that the weight functions of G_1 and G_2 are bounded by respectively m_1 and m_2 reduces it into an instance of* Channel Assignment *with $O(n_1 + n_2)$ vertices and the maximum edge weight in $O(n_1^2 m_1 + n_2^2 m_2)$.*

In the proof we use Lemma 31 to encode G_1 and G_2 in two instances of Channel Assignment, then we extend them to the common length using Lemma 30 and finally we merge them using Lemma 29. A simplified picture of the obtained Channel Assignment instance can be found in Figure 3.5.

Proof. Let use Lemma 31 on graph G_1 to obtain a $\left(v_L^{(1)}, v_R^{(1)}\right)$-spanned Channel Assignment instance $I_1 = (V_1, d_1, s_1)$ with $l_1 = O(n_1^2 m_1)$, $s_1 = 2l_1 + n_1 \cdot m_1 = O(n_1^2 m_1)$ and with the vertex $v_M^{(1)}$ (as in the statement of Lemma 31). The number of the vertices in V_1 is $O(n_1)$. Similarly, let $I_2 = (V_2, d_2, s_2)$ be a $\left(v_L^{(2)}, v_R^{(2)}\right)$-spanned

Channel Assignment instance with $l_2 = O\left(n_2^2 m_2\right)$, $s_2 = 2l_2 + n_2 \cdot m_2 = O\left(n_2^2 m_2\right)$ and with the vertex $v_M^{(2)}$ obtained from Lemma 31 from graph G_2. The number of the vertices in V_2 is $O\left(n_2\right)$.

Let us identify vertices $v_M^{(1)}$ and $v_M^{(2)}$, i.e., $v_M^{(1)} = v_M^{(2)} = v_M$ and $V_1 \cap V_2 = \{v_M\}$. And let $l_{\max} = \max\{l_1, l_2\} = O\left(n_1^2 m_1 + n_2^2 m_2\right)$ and $s = l_{\max} + \max\{s_1 - l_1, s_2 - l_2\} = O\left(n_1^2 m_1 + n_2^2 m_2\right)$. Our span will be s. Note that then every edge with a weight greater than $s - 1$ forces that our instance is a NO-instance. So if we have an edge with a weight greater that s we can replace it with the same edge with but a weight equal to s and the instance will be still a NO-instance. Therefore weights of all our edges will be bounded by $O\left(n_1^2 m_1 + n_2^2 m_2\right)$.

We can use Lemma 30 with $l = l_{\max} - l_1$ and with $r = s - (l_{\max} + s_1 - l_1)$ for extending the instance I_1 into a $\left(w_L^{(1)}, w_R^{(1)}\right)$-spanned instance

$$I_1'\left(V_1' = V_1 \cup \left\{w_L^{(1)}, w_R^{(1)}\right\}, d_1', s\right)$$

of Channel Assignment. For every YES-coloring c_1' of I_1' we know that $c_1'|_{V_1}$ is a YES-coloring of I_1 and for every YES-coloring c_1 of I_1 there exists a YES-coloring c_1' of I_1' such that $c_1'|_{V_1} = c_1$ so from the properties of I_1 (obtained from Lemma 31) we know that

- for every YES-coloring c_1' of I_1' such that $c_1'\left(w_L^{(1)}\right) \leq c_1'\left(w_R^{(1)}\right)$ there exists a perfect matching M_1 in G_1 such that $c_1'\left(v_M\right) = c_1'\left(w_L^{(1)}\right) + l_{\max} + w_1\left(M_1\right)$ and
- for every perfect matching M_1 in G_1 there exists a YES-coloring c_1' of I_1' such that $c_1'\left(w_L^{(1)}\right) \leq \left(w_R^{(1)}\right)$ and $c_1'\left(v_M\right) = c_1'\left(w_L^{(1)}\right) + l_{\max} + w_1\left(M_1\right)$.

Similarly we can use Lemma 30 with $l = l_{\max} - l_2$ and with $r = s - (l_{\max} - l_2 + s_2)$ for extending the instance I_2 into a $\left(w_L^{(2)}, w_R^{(2)}\right)$-spanned instance $I_2'(V_2' = V_2 \cup \{w_L^{(2)}, w_R^{(2)}\}, d_2', s)$ of Channel Assignment such that

- for every YES-coloring c_2' of I_2' such that $c_2'\left(w_L^{(2)}\right) \leq c_2'\left(w_R^{(2)}\right)$ there exists a perfect matching M_2 in G_2 such that $c_2'\left(v_M\right) = c\left(w_L^{(2)}\right) + l_{\max} + w\left(M_2\right)$ and
- for every perfect matching M_2 in G_2 there exists a YES-coloring c_2' of I_2' such that $c_2'\left(w_L^{(2)}\right) \leq \left(w_R^{(2)}\right)$ and $c_2'\left(v_M\right) = c_1'\left(w_L^{(2)}\right) + l_{\max} + w_1\left(M_1\right)$.

Now we can use Lemma 29 to merge the instances I_1' and I_2' into a one $(\{w_L^{(1)}, w_L^{(2)}\}, \{w_R^{(1)}, w_R^{(2)}\})$-spanned instance $I' = (V_1' \cup V_2', d, s)$. A simplified picture of the obtained instance can be found in Figure 3.5. Note that $V_1' \cap V_2' = \{v_M\}$ so

- for every YES-coloring c of I' such that $c\left(w_L^{(1)}\right) \leq c\left(w_R^{(1)}\right)$ there exist perfect matchings M_1 in G_1 and M_2 in G_2 such that $c\left(v_M\right) = c\left(w_L^{(1)}\right) + l_{\max} + w_1\left(M_1\right) = c\left(w_L^{(1)}\right) + l_{\max} + w_2\left(M_2\right)$, so $w_1\left(M_1\right) = w_2\left(M_2\right)$ and

- for every two perfect matchings M_1 in G_1 and M_{G_2} in G_2 such that $w_1(M_1) = w_2(M_2)$ there is a YES-coloring c of I such that $c\left(w_L^{(1)}\right) \leq c\left(w_R^{(1)}\right)$ and $c(v_M) = c\left(w_L^{(1)}\right) + l_{\max} + w_1(M_1)$.

Therefore the Channel Assignment instance I' has a YES-coloring if and only if there are two perfect matchings M_1 in G_1 and M_2 in G_2 such that $w_1(M_1) = w_2(M_2)$. By Lemma 29 and Lemma 30 we know that I' has $O(n_1 + n_2)$ vertices. □

Now we can use the results of Section 3.1 to get the following two theorems.

Theorem 1. *Unless ETH fails, there is no algorithm solving* Channel Assignment *in time* $2^{o(n \log n)} \cdot r^{O(1)}$ *where* n *is the number of the vertices and* r *is the bit size of the instance.*

Proof. For a given instance of Equal Weight Matchings with n vertices and the weights bounded by m we can transform it by Lemma 32 into an instance of Channel Assignment with $n' = O(n)$ vertices and the weights bounded by $\ell = O(n^2 m)$. Note that for the bit size r' of the new instance we have $r'^{O(1)} = (n')^2 \ell^{O(1)} = n, m^{O(1)} = r^{O(1)}$. Let us assume that we can solve Channel Assignment in $2^{o(n \log n)} r^{O(1)}$-time. Then we can solve our instance in time $2^{o(n' \log n')} r'^{O(1)} = 2^{o(n \log n)} r^{O(1)}$ which contradicts ETH by Theorem 2. □

Theorem 33. *Unless ETH fails, there is no algorithm solving* Channel Assignment *in time* $2^{n \cdot o(\log \log \ell)} \cdot r^{O(1)}$ *where* n *is the number of the vertices,* ℓ *is a maximum weight of the edge and* r *is the bit size of the instance.*

Proof. For a given instance of 3-CNF-SAT with n variables and m clauses we use the reduction from Theorem 28 to obtain an instance of Equal Weight Matchings with $|V_1| = O\left(\frac{n}{\log n}\right)$, $|V_2| = O\left(\frac{m}{\log m}\right)$ and the maximum matching weights bounded by 2^{3m}. Then we use the reduction from Lemma 32 to obtain an instance of Channel Assignment with

$$n' = O\left(\frac{n}{\log n} + \frac{m}{\log m}\right) = O\left(\frac{n+m}{\log(n+m)}\right)$$

vertices and the weights on the edges bounded by

$$\ell = O\left(\left(\frac{n}{\log n} + \frac{m}{\log m}\right)^2 \cdot 2^{3m}\right).$$

Then $\log \ell = O(n+m)$ and $r = O((n')^2 \cdot \log \ell) = O((n+m)^3)$. Let us assume that there is an algorithm solving Channel Assignment in $2^{n \cdot o(\log \log \ell)} r^{O(1)}$-time then we can solve our instance in time

$$2^{O\left(\frac{n+m}{\log(n+m)}\right) \cdot o(\log O(n+m))} O((n+m)^3)^{O(1)} =$$

$$= 2^{O\left(\frac{n+m}{\log(n+m)}\right) \cdot o(\log(n+m))} n + m^{O(1)} = 2^{o(n+m)}$$

which contradicts ETH by Theorem 15. □

Chapter 4

Subgraph Isomorphism

In this chapter we prove that there is no algorithm solving Subgraph Isomorphism in time $2^{o(n\sqrt{\log n})}$, unless ETH fails.

Additional notation for this chapter. All the graphs used in this chapter are undirected, however in edge colored graphs there might be several parallel edges between the same pair of vertices.

For a CNF-SAT formula φ let $\mathrm{Var}(\varphi)$ be the set of variables of φ, whereas $\mathrm{Clauses}(\varphi)$ is the set of clauses of φ.

4.1 Overview of the reduction

Note that due to the Corollary 17 we can start our reduction from (3,4)-CNF-SAT instead of more general 3-CNF-SAT. In Sections 4.2 -4.4 we present our reduction from (3,4)-CNF-SAT to Subgraph Isomorphism, which is self-contained and independent of [56]. Throughout the reduction we define the *size* of a Subgraph Isomorphism instance to be the total number of vertices in the pattern and host graphs. Our reduction can be broken into three steps:

- First, in Section 4.2, we preprocess the given (3,4)-CNF-SAT formula and pack its variables and clauses into groups of logarithmic size. Importantly, we ensure that there is only a limited interaction between the groups by marking variables with colors – applying further steps of the reduction for an arbitrary grouping would not yield a superexponential lower bound for Subgraph Isomorphism.
- Next, in Section 4.3, we use the packing to create $2^{O(n/\log n)}$ smaller instances of a variant of the Subgraph Isomorphism problem, where additionally vertices and edges have colors which have to be preserved by the mapping. This proves that the color variant of Subgraph Isomorphism admits a tight lower bound of $2^{\Omega(n \log n)}$ under the Exponential Time Hypothesis. In this step, we use a simple technique of guessing preimage sizes, which allows us to circumvent the usual technical difficulties of encoding valuations by permutations.
- Finally, in Section 4.4 we reduce the color version of Subgraph Isomorphism to the original variant, incurring an $O(\sqrt{\log n})$ increase in the instance size.

Definition 34. *We define the* (c,t)-Subgraph Isomorphism *problem as a generalization of* Subgraph Isomorphism *where every vertex of the pattern and host graphs is colored in one of* c *colors, and every edge is colored in one of* t *colors, and the mapping is restricted to preserving vertex and edge colors.*

In particular, Subgraph Isomorphism is the same as $(1,1)$-Subgraph Isomorphism. The pipeline of our lower bound consists of two steps. First, in Lemma 35, given a (3,4)-CNF-SAT formula with n variables we construct a set of $2^{O(n/\log n)}$ instances of $(O(1), O(\log n))$-Subgraph Isomorphism of $O(n/\log n)$ size each. Note that the number of vertex colors is constant, whereas the number of edge colors is logarithmic. In the second step (Lemma 45) we reduce to the original variant of Subgraph Isomorphism, with an additional increase in the instance size by a factor of $O(\sqrt{\log n})$, leading to a final size of $O(n/\sqrt{\log n})$, which is sublinear.

Lemma 35. *Given a* (3,4)-CNF-SAT *formula* φ *with* n *variables one can in* $2^{O(n/\log n)}$ *time create a set* $\mathcal{S}$ *of* $2^{O(n/\log n)}$ *instances of* $(O(1), O(\log n))$-Subgraph Isomorphism *of size* $O(n/\log n)$*, such that* φ *is satisfiable iff any instance in* $\mathcal{S}$ *is satisfiable, and the host graph and the pattern graph have the same number of vertices for every instance in* $\mathcal{S}$*.*

Lemma 36. *An instance of* (c,t)-Subgraph Isomorphism*, where the host graph and the pattern graph have the same number of vertices, can be reduced to an equivalent instance of* Subgraph Isomorphism *with* $O(c\sqrt{t})$ *times more vertices.*

Having the two lemmas above, which we prove in the remainder of this chapter, we can prove Theorem 3.

Theorem 3. *There is no algorithm which solves* Subgraph Isomorphism *in time* $2^{o\left(n\sqrt{\log n}\right)}$*, unless the Exponential Time Hypothesis fails.*

Proof. Assume that a $2^{o(n\sqrt{\log n})}$ time algorithm exists for the Subgraph Isomorphism problem, where $n = |V(G)| + |V(H)|$. We show an algorithm solving a given (3,4)-CNF-SAT formula φ with n variables in time $2^{o(n)}$, leading to a contradiction with the Exponential Time Hypothesis by Corollary 17.

Indeed, the prerequisites of Lemma 35 are satisfied, and in $2^{O(n/\log n)}$ time we can obtain a corresponding set $\mathcal{S}$ of $2^{O(n/\log n)}$ instances of $(O(1), O(\log n))$-Subgraph Isomorphism of size $O(n/\log n)$ each. Next, we apply Lemma 45 to transform each instance in $\mathcal{S}$ into an instance of Subgraph Isomorphism of size $O(n/\sqrt{\log n})$, obtaining the set $\mathcal{S}'$. Finally, we apply the hypothetical $2^{o(n\sqrt{\log n})}$-time algorithm to the instances in $\mathcal{S}'$, leading to $2^{O(n/\log n)}2^{o(n)} = 2^{o(n)}$ running time. □

We prove Lemma 35 in Section 4.3 and Lemma 45 in Section 4.4. However, before we describe the reduction, in Section 4.2 we present how to group clauses of a given (3,4)-CNF-SAT formula in a way that allows a sublinear reduction to Subgraph Isomorphism.

4.2 Grouping clauses

As we already mentioned, when proving superexponential lower bounds based on the Exponential Time Hypothesis, we need to come up with a reduction producing an instance of Subgraph Isomorphism of sublinear number of vertices. In this section we show how to preprocess a given (3,4)-CNF-SAT formula and partition its clauses into groups of logarithmic size. Our grouping is far from arbitrary, as we need to precisely control the interactions between clauses sharing the same variables.

Before we arrive at our main structural lemma, we need a simple step in which we assign colors to variables so that no clause contains two variables of the same color and moreover the counts of variables in each color are balanced.

First we show the existence of a potentially unbalanced 9-coloring.

Lemma 37. *Given a* (3,4)-CNF-SAT *formula* φ *with* n *variables we can color the variables of* φ *in polynomial time using at most* 9 *colors, so that no clause contains two variables of the same color.*

Proof. Construct an auxiliary graph G_φ, the vertex set of which is the set of variables of S, where two vertices of G_φ are adjacent iff they both appear in at least one of the clauses of φ. Note that the maximum degree of G_φ is bounded by 8, as each variable appears in at most 4 clauses and each clause contains at most 3 literals. Consequently, we can color G_φ with at most 9 colors in a greedy manner. □

Lemma 38. *Given an integer* $k > 9$ *and a* (3,4)-CNF-SAT *formula* φ *with* n *variables we can color the variables of* φ *in polynomial time using at most* k *colors, so that no more than* $\lceil n/(k-9) \rceil$ *variables share the same color and no clause contains two variables of the same color.*

Proof. First, color the variables into 9 colors using Lemma 37. Then, while there exists a color with more than $\lceil n/(k-9) \rceil$ variables assigned to it, separate $\lceil n/(k-9) \rceil$ of them to form a new color. This can occur at most $k-9$ times, and the lemma follows. □

Having Lemma 38 we are ready to pack the clauses of a given (3,4)-CNF-SAT formula into 2^k groups, which is the main structural insight in our reduction. It is important that no two clauses from the same group contain variables of the same color.

Lemma 39. *Given a* (3,4)-CNF-SAT *formula* φ *with* $n \geq 16$ *variables one can in polynomial time construct:*

- *a coloring* $l : \mathrm{Var}(\varphi) \to [k]$ *of the variables in* φ *into* k *colors, such that no two variables contained in a clause of* φ *share the same color, and*
- *a packing* $h : \mathrm{Clauses}(\varphi) \to [2^k]$ *of the clauses into* 2^k *groups indexed by* $\{0, \ldots, 2^k - 1\}$*, such that for any* $i \in [2^k]$ *no two clauses that are mapped to* i *contain variables of the same color,*

where $k := \lceil \log n - \log\log n \rceil + 9$.

Proof. Let l be the coloring guaranteed by Lemma 38. We slightly overload the notation and by $l(C)$ denote the set of colors of variables in $C \in \text{Clauses}(\varphi)$.

We construct the packing h in a greedy manner. Consider all the clauses of $\text{Clauses}(\varphi)$ one by one in an arbitrary order. When a clause $C \in \text{Clauses}(\varphi)$ is processed, we find any group $i \in [2^k]$, such that the set of colors of variables appearing in clauses already assigned to i is disjoint from $l(C)$. If several such sets i exist, we pick an arbitrary one and assign $h(C) := i$.

It remains to prove that such an i always exists for the value of k as stated in the lemma. We prove this by contradiction: suppose that at some point, for some clause C, for every i one of the colors in $l(C)$ is already present in a clause already assigned to i. Let $m_{l(C)}$ be the number of clauses of φ containing at least one color from $l(C)$. As there are exactly 2^k groups, and we cannot assign C to any of them, it means that

$$m_{l(C)} \geq 2^k \geq 512n/\log n\,, \tag{4.1}$$

since each of the 2^k groups is blocked by a different clause containing at least one color from $l(C)$.

On the other hand we have only 3 colors in $l(C)$ and we know by Lemma 38, that no more than $\lceil n/(k-9) \rceil$ variables are assigned to any color, and by the upper bound on the frequency of each variable of φ we know that no variable occurs in more than 4 clauses. Consequently, the number of clauses having at least one common color with C is upper bounded by

$$\begin{aligned} m_{l(C)} &\leq 3 \cdot \lceil n/(k-9) \rceil \cdot 4 \leq 12 \cdot (n/(k-9)+1) \\ &\leq 12 \cdot (n/(\log n - \log\log n)+1) \leq 12 \cdot (2n/\log n + 1) \\ &\leq 12 \cdot (2n/\log n + 0.5n/\log n) \leq 30n/\log n\,, \end{aligned} \tag{4.2}$$

where in the last two inequalities we have used that $\log n - \log\log n \geq 0.5 \log n$ and $n/\log n \geq 2$ for $n \geq 16$. Note that (4.2) yields a contradiction with (4.1), and the lemma follows. □

4.3 From (3,4)-CNF-SAT to Subgraph Isomorphism with colors

The technical crux of our result is a method of encoding information in permutations – mappings from the pattern graph to the host graph. The intuition behind this technique is that the number of permutations of an n element set is $n! = 2^{\Theta(n \log n)}$ and therefore a single permutation carries $\Theta(n \log n)$ bits of information. This means that from the information-theoretic perspective it should be possible to encode an assignment of Boolean values to n variables using a permutation of $O(n/\log n)$ elements.

Every element in a permutation is responsible for encoding some number of bits, forming what we call a *pack* of bits. We do not restrict ourselves to packs of constant

size, but each pack we create is of size no greater than logarithmic. The position of an element in a permutation should uniquely determine the values of all the bits from its pack. The problem is, however, that in a permutation no two elements can be mapped to the same position, which potentially might make it impossible to assign the same valuation to two different packs of bits.

Here, we present a new and simple way of circumventing this obstacle by guessing the sizes of preimages in a mapping corresponding to a satisfying assignment. Less formally, what we do is replicate some positions and remove other ones, so that in some branch our guess will transform a mapping we had in mind into a permutation.

We would like to note that encoding groups of bits by a position in a permutation was already used by Marx, Lokshtanov and Saurabh [103] in the $k \times k$-Permutation Clique problem, as well as in Chapter 3 of this dissertation in the lower bound for the Channel Assignment problem.

In the remainder of this section we prove Lemma 35, that is show how to transform a (3,4)-CNF-SAT formula φ into $2^{O(n/\log n)}$ instances of $(O(1), O(\log n))$-Subgraph Isomorphism with $O(n/\log n)$ vertices. In order to do this we need to introduce notation for binary strings. Assume for a moment, that n is a power of two, i.e., $n = 2^k$ for $k \in \mathbb{N}$. One can view elements in a permutation as integers between 0 and $n-1$, denoted as $[n]$, but also as a set of binary strings of length k – being the binary representations of numbers from $[n]$, denoted as $2^{[k]}$. We will use the two conventions interchangeably and for this reason we need the following notation regarding binary strings. Let $\mathcal{B} := \{0,1\}^*$ be the set of all binary strings. and $\mathcal{B}_k := \{0,1\}^k$ be the set of binary strings of size exactly k. For a binary string s, let $|s|$ be its length. We denote the i-th digit (starting from 0) of a binary string s as s_i.

Lemma 35. *Given a* (3,4)-CNF-SAT *formula φ with n variables one can in $2^{O(n/\log n)}$ time create a set $\mathcal{S}$ of $2^{O(n/\log n)}$ instances of $(O(1), O(\log n))$-*Subgraph Isomorphism *of size $O(n/\log n)$, such that φ is satisfiable iff any instance in $\mathcal{S}$ is satisfiable, and the host graph and the pattern graph have the same number of vertices for every instance in $\mathcal{S}$.*

Proof. Assume we are given a (3,4)-CNF-SAT formula φ with n variables. Define $k := \lceil \log n - \log\log n \rceil + 9$. We prove that solving φ can be reduced to solving less than $2^{2^{k+1}} = 2^{O(n/\log n)}$ instances of $(3,k)$-Subgraph Isomorphism, with vertex colored denoted as red, green and blue, and edge colors denoted by $[k]$, where the number of vertices of both the pattern and host graph equals

$$2^k + 8 \cdot \binom{k}{3} + 1 = O(n/\log n)\,.$$

Satisfying assignment gadget.
The assignment gadget G consists of a path on $8 \cdot \binom{k}{3}$ red vertices, with a single green vertex appended at one end. The red vertices will be uniquely identifiable based on the distance from the green vertex. Each red vertex will correspond to a choice of 3 distinct indices from $[k]$ and an assignment of binary values to each of them:

$$(i_1, i_2, i_3, b_1, b_2, b_3) \in [k]^3 \times \mathcal{B}_3\,,$$
$$i_1 < i_2 < i_3\,.$$

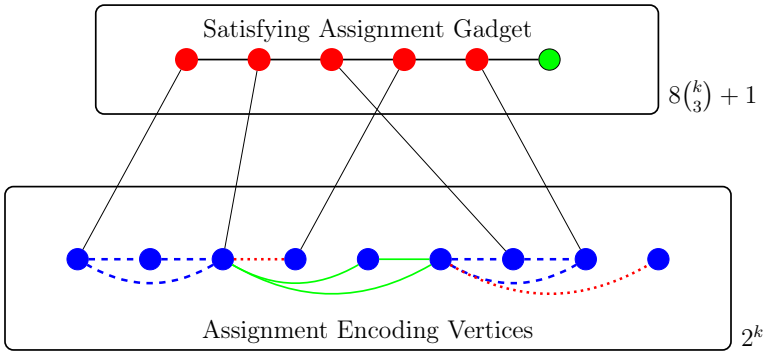

Figure 4.1: A simplified view of the pattern graph.

Intuitively, an edge between one of the clause vertices and a red vertex will indicate that 'in this pack of clause valuations, the variables at positions i_1, i_2, i_3 are *not* assigned values b_1, b_2, b_3 at the same time'. All edges in G are of color 0.

Pattern graph construction.

The pattern graph will be constant across all the created instances. The pattern graph P consists of 2^k blue vertices corresponding to packs of clauses and a copy of the satisfying assignment gadget G. First, find the coloring l and packing h guaranteed by Lemma 39. We associate each blue vertex of P with a different group in the image of h.

For every variable x in S and every two distinct clauses C_1, C_2 containing x, we add an edge of color $l(x)$ between the blue vertices corresponding to $h(C_1)$ and $h(C_2)$. Intuitively, these edges signify that x has to have a consistent valuation when choosing valuations of variables in packs containing C_1 and C_2.

Additionally, for every clause C in φ we add an edge of color 0 between $h(C)$ (i.e., the pack containing C) and the red vertex $(i_1, i_2, i_3, b_1, b_2, b_3)$, where $i_1 < i_2 < i_3$ are the colors of variables in C and (b_1, b_2, b_3) is their only valuation that does not satisfy C.

Host graph construction.

We will generate a different host graph for every sequence of preimage sizes of the valuations of the groups. Fix a sequence $s_0, s_1, \ldots, s_{2^k-1}$, such that $s_i \geq 0$ for all i and $\sum s_i = 2^k$.

The number of possible such sequences s is

$$\binom{2^{k+1}-1}{2^k-1} \leq 2^{2^{k+1}}.$$

The host graph H_s consists of 2^k blue vertices corresponding to valuations of the groups of clauses and a copy of the satisfying assignment gadget G. For the binary string of length k corresponding to $i \in [2^k]$, we generate s_i vertices corresponding to it.

For $j \in \mathbb{Z}_k$, we join two blue vertices u, v in H_s with an edge of color j iff $u_j = v_j$, that is iff the j-th bit in both strings is the same. Intuitively, lack of an edge of color

j between two blue vertices u, v in H_s disallows assigning two packs of clauses to vertices u and v when the variable of color j in both packs is the same, as it would lead to inconsistent valuation.

For every blue vertex u in H_s and red vertex $v = (i_1, i_2, i_3, b_1, b_2, b_3)$ in G, we connect u and v with an edge of color 0 iff $u_{i_j} \neq b_j$ for some $j \in \mathbb{Z}_3$. Less formally, lack of an edge between a blue vertex u and a red vertex $v = (i_1, i_2, i_3, b_1, b_2, b_3)$ means that a pack of clauses can be assigned to u, only if the valuation corresponding to the bit string associated with u only if there is no clause such that assigning values b_1, b_2, b_3 to variables of colors i_1, i_2, i_3, respectively, would cause some clause from the pack to be unsatisfied.

Proof of correctness.
As the construction can be carried out in polynomial time per instance and both the host and pattern graphs have $O(n/\log n)$ vertices as promised, it remains to prove that φ is satisfiable iff for some instance the pattern graph P is a subgraph of the host graph H_s.

Claim 40. *If φ is satisfiable, then for some sequence of preimage sizes s, P is a subgraph of H_s.*

Proof: First, assume that φ is satisfiable and let $\mathrm{val} : \mathrm{Var}(\varphi) \to \{\mathbf{true}, \mathbf{false}\}$ be a satisfying assignment. We construct a mapping $g : V(P) \to V(H)$ as follows. For a group $i \in [2^k]$ let Var_i be the set of variables occurring in all the clauses assigned to i by the packing h. If any colors do not occur in $l(\mathrm{Var}_i)$, add arbitrary variables to Var_i so that $l(\mathrm{Var}_i) = [k]$. Define $f(i) = \mathrm{val}|_{\mathrm{Var}_i}$, i.e., the bit string representing valuation of variables from Var_i by val. Let s be the sequence of preimage sizes of f. A bijection g corresponding to f exists between the blue vertices of P and H_s. We extend g to all the vertices of P by mapping each vertex of the satisfying assignment gadget G in P to its corresponding copy in H_s, obtaining a bijection $g' : V(P) \to V(H_s)$.

It remains to check that b' preserves all the edges. Clearly, the edges within the satisfying assignment gadget G are preserved. Consider any edge of color $c \in [k]$ in the pattern graph between two blue vertices u, v, corresponding to groups i and j. By construction, this means that the packs $h^{-1}(i)$ and $h^{-1}(j)$ share a variable of color c, which means that by the definition of f the bit strings $f(i)$ and $f(j)$ assign the same value to the index corresponding to this variable. As g extends f, we have $g(i) = f(i)$ and $g(j) = f(j)$, hence the bit strings corresponding to $g'(u)$ and $g'(v)$ have the same value on the c-th position, hence by construction of the host graph $g'(u)$ and $g'(v)$ are connected by an edge of color c. Finally, we inspect the edges between blue vertices and red vertices. Consider a blue vertex u associated with a set $A \subseteq [k]$, which is connected to some red vertex $v = (i_1, i_2, i_3, b_1, b_2, b_3)$, because of a clause $C \in h^{-1}(A)$. As val is a satisfying assignment and g extends bit strings assigned by f, we infer that the vertex $g(u)$ is connected to the red vertex v. Consequently, P is a subgraph of H_s witnessed by the mapping g'. ⌟

Now we prove the following claim.

Claim 41. *If for any s it holds that P is a subgraph of H_s, then φ is satisfiable.*

Proof: Let g be a mapping from P to H_s witnessing the fact that P is a subgraph of H_s. As g respects colors, we infer that the single green vertex in P is mapped to the single green vertex in H_s. Similarly all the red vertices of P have to be mapped to red vertices of H_s. Additionally the distance between each red vertex v and the green vertex in P cannot be smaller than the distance between $g(u)$ and the green vertex in H_s. As red vertices induce a path, and the green vertex is pendant to one if its ends, we infer that g assigns each vertex of the satisfying-assignment-gadget in P to its copy in H_s (in short, by construction there are no non-trivial automorphisms of the gadget).

Construct an assignment $\mathrm{val} : \mathrm{Var}(\varphi) \to \{\mathbf{true}, \mathbf{false}\}$ as follows. For a variable $x \in \mathrm{Var}(\varphi)$ find any clause C that contains x and assign $\mathrm{val}(x)$ to **true** iff $g(h(C))_{l(x)} = 1$, where $h(C)$ is the blue vertex associated with C and $l(x)$ is the color of the variable x. Note that by construction the assignment val is well-defined, as edges between blue vertices guarantee consistence. Consider a clause C. The edges between $h(C)$ and red vertices in the pattern graph P have to be preserved by g, and we already observed that g maps red vertices of P to their corresponding copies in H_s. Hence, we infer that there is an edge between $g(h(C))$ and the red vertex $v = (i_1, i_2, i_3, b_1, b_2, b_3)$, where b_1, b_2, b_3 is the only assignment to variables of C, where $l(C) = \{i_1, i_2, i_3\}$, which does not satisfy C. This in turn implies that for at least one variable of C the assignment val assigns a different value than the one corresponding to the appropriate bit from $\{b_1, b_2, b_3\}$. Consequently, val is a satisfying assignment. ⌟

Claims 40 and 41 prove equivalence of the formula φ and created instances of $(3, k)$-Subgraph Isomorphism, hence the proof of Lemma 35 follows. □

We would like to note that Lemma 35 implies a tight bound for the auxiliary version of Subgraph Isomorphism with colors, even in the case when the number of vertex colors is constant and the number of edge colors is logarithmic.

Corollary 42. *There is no $2^{o(n \log n)}$ time algorithm for the $(O(1), O(\log n))$-Subgraph Isomorphism problem, unless the Exponential Time Hypothesis fails.*

4.4 Removing the colors

In this section we prove Lemma 45, first by showing how to remove colors from edges, and next by removing colors from vertices.

Lemma 43. *An instance (G, H) of (c, t)-Subgraph Isomorphism such that $|V(G)| = |V(H)|$ can be reduced to an instance (G', H') of $(c+1, 1)$-Subgraph Isomorphism with $O(\sqrt{t})$ times more vertices such that $|V(G')| = |V(H')|$.*

Proof. Let (G, H) be an instance of (c, t)-Subgraph Isomorphism such that $|V(G)| = |V(H)|$. Assume that none of the vertices of the instance (G, H) was colored yellow. Let $t' := 2\lceil\sqrt{t}\rceil$. Note that for $t \geq 1$ we have $\lceil\sqrt{t}\rceil \geq 1$ and then

$$\binom{t'}{2} = \frac{2\lceil\sqrt{t}\rceil \cdot (2\lceil\sqrt{t}\rceil - 1)}{2} \geq \lceil\sqrt{t}\rceil^2 \geq t.$$

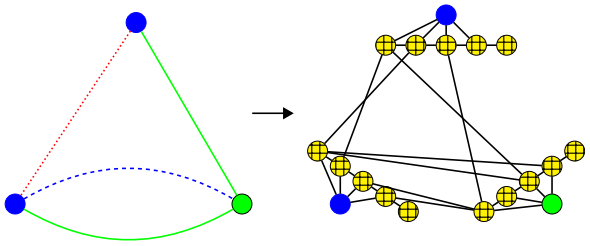

Figure 4.2: The reduction described in Lemma 43. In the example, $t = 3$, $t' = 2\lceil\sqrt{t}\rceil = 4$, $p(\text{red}) = (1,2)$, $p(\text{green}) = (1,3)$ and $p(\text{blue}) = (1,4)$.

Therefore for each color $x \in [t]$ we can pick a different pair $p(x) := (i,j)$ where $1 \leq i < j \leq t'$.

For every vertex u in either the pattern or the host graph, we replace it by a gadget consisting of $t'+2$ vertices (see Fig. 4.2):

- a *center vertex* u'_0 of the same color as u, and
- a path on $t'+1$ yellow vertices $u'_1, \ldots, u'_{t'+1}$, the first t' of which are connected to the center vertex.

For every edge (u,v) of color x in either the pattern or the host graph, we replace it by the edges (u'_i, v'_j) and (u'_j, v'_i) in the modified graph, where $(i,j) = p(x)$. We denote this new instance of $(c+1,1)$-Subgraph Isomorphism as (G', H'). Note that $|V(G')| = (t'+2)\cdot|V(G)|$ and $|V(H')| = (t'+2)\cdot|V(H)|$ hence $|V(G')| = |V(H')|$ and also $|V(G')| = O(\sqrt{t})\cdot|V(G)|$ and $|V(H')| = O(\sqrt{t})\cdot|V(H)|$.

It remains to prove that G is a subgraph of H iff G' is a subgraph of H'. If G is a subgraph of H then there exists an injective function $f : V(G) \to V(H)$ such that edges and colors are preserved. Note that every vertex of the instance (G', H') is of the form u'_i for some vertex u of the instance (G, H). Let $g : V(G') \to V(H')$ be a function such that $g(u'_i) = f(u)'_i$. The function g is an injection because the function f is an injection. The function g preserves the colors of the vertices because $col(g(u'_0)) = col(f(u)'_0) = col(f(u)) = col(u) = col(u'_0)$ and for $i > 0$ the color of u'_i is always yellow. The function g preserves also the edges. Let $u'_i v'_j$ be an edge in G' such that $i \leq j$ (we can assume this w.l.o.g.). If $u = v$ then there exists also an edge $f(u)'_i f(u)'_j = g(u'_i)g(v'_j)$ in H' because all the gadgets have exactly the same structure of the internal edges i.e. if there exists an edge $u'_i u'_j$ in a gadget for any vertex u then for every vertex v there exists an edge $v'_i v'_j$ in a gadget for vertex v. If $u \neq v$ then $1 \leq i < j \leq t'$ and there exists an edge uv of the color $x = p^{-1}(i,j)$ in G and then there exists an edge $f(u)f(v)$ of the color x in H and (because $(i,j) = p(x)$) we know that there exists an edge $f(u)'_i f(v)'_j = g(u'_i)g(v'_j)$ in H'. The edges of (G', H') have only one color thus g preserves the colors of the edges trivially. Hence G' is a subgraph of H'.

If G' is a subgraph of H' then there exists an injective function $g : V(G') \to V(H')$ such that edges and colors are preserved. The vertices of the form u'_0 are the only vertices of G' and H' which are not yellow. Therefore if $g(u'_i) = v'_j$, then $i = 0$ iff

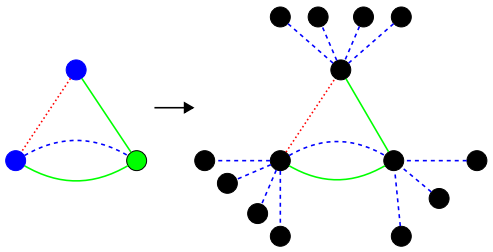

Figure 4.3: The reduction described in Lemma 44. In the example, the green and blue colors of the vertices represent the numbers 2 and 3 respectively and the blue color of the edges represent the number 1.

$j = 0$. If $g(u'_0) = v'_0$ then for every u'_i such that $1 \leq i \leq t'$ we have $g(u'_i) = v'_j$ for some $1 \leq j \leq t'$ because the vertices $u'_1, u'_2, \ldots, u'_{t'}$ are yellow neighbors of u'_0 and the vertices $v'_1, v'_2, \ldots, v'_{t'}$ are the only yellow neighbors of v'_0. On the other hand we know that $|V(G)| = |V(H)|$ and then the number of the vertices of the form u'_i for $1 \leq i \leq t'$ is the same in G' and in H'. Therefore for every vertex v'_i in H' such that $1 \leq i \leq t'$ there exists a vertex u'_j in G' such that $1 \leq j \leq t'$ and $g(u'_j) = v'_i$. Therefore if $g(u'_i) = v'_j$ then $i = t' + 1$ iff $j = t' + 1$. Moreover the vertices $u'_1, u'_2, \ldots, u'_{t'}$ create a path (in this order) and the only directed paths containing exactly the vertices $\{v'_1, v'_2, \ldots, v'_{t'}\} = g(\{u'_1, u'_2, \ldots, u'_{t'}\})$ are $v'_1, v'_2, \ldots, v'_{t'}$ and $v'_{t'}, v'_{t'-1}, \ldots, v'_1$. But the vertex $u'_{t'}$ is a neighbor of the vertex $u'_{t'+1}$ and the vertex v'_1 has no neighbor of the form $w'_{t'+1}$ for any vertex w in H. On the other hand the vertex $u'_{t'+1}$ has to be mapped to a vertex of the form $w'_{t'+1}$ for some vertex w in H. Therefore the path $u'_1, u'_2, \ldots, u'_{t'}$ is mapped to the path $v'_1, v'_2, \ldots, v'_{t'}$ i.e. for every vertex u'_i such that $1 \leq i \leq t'$ we have $g(u'_i) = v'_i$. Let $f : V(G) \to V(H)$ be a function such that $f(u) = v$ iff $g(u'_0) = v'_0$. (note that then $f(u)'_0 = v'_0 = g(u'_0)$). The function f is an injection because the function g is an injection. The function f preserves the colors of the vertices because $col(f(u)) = col(f(u)'_0) = col(g(u'_0)) = col(u'_0) = col(u)$. The function f preserves also the edges with their colors because if there is an edge uv of the color x in the graph G then for $(i, j) = p(x)$ there is an edge $u'_i v'_j$ in the graph G' and therefore there is an edge $g(u'_i)g(v'_j) = f(u)'_i f(v)'_j$ in the graph H' and then there is an edge $f(u)f(v)$ of the color x in the graph H. Hence G is a subgraph of H. □

Having reduced the number of edge colors down to one, it remains to reduce the number of vertex colors. Note that in the following lemma it would be enough to assume $t = 1$, however we prove the lemma in a more general form as it does not affect the complexity of the proof.

Lemma 44. *An instance (G, H) of (c, t)-*Subgraph Isomorphism *such that $|V(G)| = |V(H)|$ can be reduced to an instance (G', H') of $(1, t)$-*Subgraph Isomorphism *with $O(c)$ times more vertices such that $|V(G')| = |V(H')|$.*

Proof. Let (G, H) be an instance of (c, t)-Subgraph Isomorphism such that $|V(G)| = |V(H)|$. Number the vertex colors arbitrarily from 1 to c and number the edge colors

arbitrarily from 1 to t. We can assume that for every vertex color the number of the vertices in this color in G and in H is the same because otherwise we can produce a trivial NO instance as (G', H'). In both pattern and host graphs, for each vertex v, attach $i+1$ new leaves $v_1, v_2, \ldots, v_{i+1}$ to it, where i is the color of v, using edges of color 1 (or any fixed color from 1 to t). We also denote $v_0 = v$. Consider the $(1,t)$-Subgraph Isomorphism instance (G', H') on the new graphs. For every vertex color the number of the vertices in that color in G is the same as in H and therefore the number of added leaves is the same in G' as in H'. Hence $|V(G')| = |V(H')|$.

If G is a subgraph of H then there exists an injective function $f : V(G) \to V(H)$ such that edges and colors are preserved. Note that every vertex of the instance (G', H') is of the form v_i for some vertex v of the instance (G, H). Let $g : V(G') \to V(H')$ be a function such that $g(v_i) = f(v)_i$ which is a correctly defined function because $col(v) = col(f(v))$ and therefore v_0 has the same number of leaves in the graph G' as $f(v)_0$ in the graph H'. The function g is an injection because the function f is an injection. The function g preserves the colors of the vertices trivially. We show that the function g preserves also edges and their colors. Let assume that there is an edge $u_i v_j$ for $i \leq j$ (we can assume that w.l.o.g) of the color x in the graph G'. If $j > 0$ then $u = v$, $i = 0$ and $x = 1$ and there exists also an edge $f(u)_0 f(u)_j = g(u_i)g(v_j)$ of the color $1 = x$ in the graph H'. Otherwise we have $i = j = 0$ and then there exists an edge uv of the color x in the graph G thus there exists an edge $f(u)f(v)$ of the color x in the graph H hence there exists an edge $f(u)_0 f(v)_0 = g(u_i)g(v_j)$ of the color x in the graph H'. Therefore G' is a subgraph of G.

If G' is a subgraph of H' then there exists an injective function $g : V(G') \to V(H')$ such that edges and colors are preserved. All vertices from the original pattern graph have to be matched to vertices from the original host graph, as they are the only ones of degree greater than 1 in the new graphs. But the number of the vertices of the form u_0 is the same in G' as in H' because $|V(G)| = |V(H)|$. Therefore for every vertex of the form v_0 in H' there exists a vertex of the form u_0 in G' such that $g(u_0) = v_0$. Hence, the leaves have to map to leaves. But the number of leaves is the same in G' as in H'. Thus for every leaf v_i in H' there exists a leaf u_j in G' such that $g(u_j) = v_i$. Hence all the leaves are used and then for every vertex u_0 in G' the number of leaves of u_0 in G' is the same as the number of leaves of $g(u_0)$ in H'. Let us consider a function $f : V(G) \to V(H)$ such that $f(u) = v$ iff $g(u_0) = v_0$ (then $f(u)_0 = v_0 = g(u_0)$). Note that $f = g|_{V(G)}$. The function f is an injection because the function g is an injection. The function f preserves the colors of the vertices because for every v in the graph G we have that v_0 has exactly $col(v)+1$ leaves as neighbors in the graph G' and then $g(v_0) = f(v)_0$ has also exactly $col(v)+1$ leafs as neighbors. But on the other hand the vertex $f(v)_0$ has exactly $col(f(v))+1$ leafs as neighbors in the graph H' and therefore $col(f(v)) = col(v)$. The function f preserves also the edges with their colors because for every edge uv of a color x in the graph G there exists an edge $u_0 v_0$ of the color x in the graph G' and therefore there exists an edge $g(u_0)g(v_0) = f(u)_0 f(v)_0$ of the color x in the graph H' and hence there exists an edge $f(u)f(v)$ of the color x in the graph H. Therefore G is a subgraph of H. $\square$

Lemma 45. *An instance of* (c,t)-Subgraph Isomorphism*, where the host graph and the pattern graph have the same number of vertices, can be reduced to an equivalent instance of* Subgraph Isomorphism *with* $O(c\sqrt{t})$ *times more vertices.*

Proof. The claim follows directly from consecutive application of Lemmas 43 and 44. □

4.5 From Graph Homomorphism to Subgraph Isomorphism

In this section we present a reduction which shows that one can solve the Graph Homomorphism problem by solving $2^{|V(G)|+|V(H)|}$ instances of the Subgraph Isomorphism problem, demonstrating that the lower bound of $2^{\Omega(|V(G)|\log|V(H)|)}$ of Fomin et al. [57, 38] implies a $2^{\Omega(n\log n)}$ lower bound under the Exponential Time Hypothesis for the Subgraph Isomorphism problem, where $n=|V(G)|+|V(H)|$.

Graph Homomorphism
Input: undirected graphs G, H.
Question: Is there a homomorphism from G to H, i.e., does there exist a function $h: V(G)\to V(H)$, such that for each edge $uv\in E(G)$ we have $h(u)h(v)\in E(H)$.

Theorem 4. *Given an instance* (G,H) *of* Graph Homomorphism *one can in* $O(2^n n^{O(1)})$ *time create* 2^n *instances of* Subgraph Isomorphism *with* n *vertices, where* $n=|V(G)|+|V(H)|$*, such that* (G,H) *is a yes-instance iff at least one of the created instances of* Subgraph Isomorphism *is a yes-instance.*

Proof. Let (G,H) be an instance of Graph Homomorphism and denote $n=V(G)+V(H)$. Note that any homomorphism h from G to H can be associated with some sequence of non-negative numbers $(|h^{-1}(v)|)_{v\in V(H)}$, being the numbers of vertices of G mapped to particular vertices of H. The sum of the numbers in such a sequence equals exactly $|V(G)|$. As the number of such sequences is $\binom{V(G)+V(H)-1}{V(H)-1}\leq 2^n$, we can enumerate all such sequences in time $2^n n^{O(1)}$. For each such sequence $(a_v)_{v\in V(H)}$ we create a new instance (G',H') of Subgraph Isomorphism, where the pattern graph remains the same, i.e., $G'=G$, and in the host graph H' each vertex of $v\in V(H)$ is replicated exactly a_v times (possibly zero). Observe that $|V(H')|=|V(G')|$.

We claim that G admits a homomorphism to H iff for some sequence $(a_v)_{v\in V(H)}$ the graph G' is a subgraph of H'. First, assume that G admits a homomorphism h to H. Consider the instance (G',H') created for the sequence $a_v=|h^{-1}(v)|$ and observe that we can create a bijection $h': V(G')\to V(H')$ by assigning $v\in V(G')$ to its private copy of $h(v)$. As h is a homomorphism, so is h', and as h' is at the same time a bijection, we infer that G' is a subgraph of H'.

On the other hand if for some sequence $(a_v)_{v\in V(H)}$ the constructed graph G' is a subgraph of H', then projecting the witnessing injection $g: V(G')\to V(H')$ so that $g'(v)$ is defined as the prototype of the copy $g(v)$ gives a homomorphism from G to H, as copies of each $v\in V(H)$ form independent sets in H'. □

Chapter 5

Rainbow Coloring

In this chapter we prove that there is no algorithm solving Rainbow k-Coloring in time $2^{o(n^{3/2})}$ for any $k \geq 2$, unless ETH fails.

Organization of the chapter. The main difficulties we encountered are sketched in Section 5.1. The further sections contain consecutive steps of our reduction.

Additional notation for this chapter. All graphs we consider in this chapter are simple and undirected. We denote $\Delta_1(G) = \max\{\Delta(G), 1\}$.

By $\bar{E}$ we denote the set of anti-edges, i.e., $\bar{E} = \binom{V}{2} \setminus E$. When $G = (V, E)$ is a graph then $\bar{G} = (V, \bar{E})$ is its *complement graph*.

5.1 Overview

The main goal of this chapter is to show that for any $k \geq 2$ Rainbow k-Coloring does not admit an algorithm running in time $2^{o(n^{3/2})}$, unless the Exponential Time Hypothesis fails. Let us give a high-level overview of our proof. A natural idea would be to begin with a 3-CNF-SAT formula ϕ with n variables and then transform it in time $2^{o(n)}$ to an equivalent instance $G = (V, E)$ of Rainbow k-Coloring with $O(n^{2/3})$ vertices. Then indeed a $2^{o(|V|^{3/2})}$-time algorithm that solves Rainbow 2-Coloring can be used to decide 3-CNF-SAT in time $2^{o(n)}$. Note that in a typical NP-hardness reduction, we observe some polynomial blow-up of the instance size. For example, one can verify that in the reduction of Chakraborty *et al.* [22] showing the NP-hardness of Rainbow k-Coloring, the initial 3-CNF-SAT formula with n variables and m clauses is transformed into a graph with $\Theta(n^4+m^4)$ vertices and edges. In our case, instead of a blow-up we aim at *compression*: the number of vertices needs to be much smaller than the number of variables in the input formula ϕ. As usual in reductions, variables and clauses in ϕ are going to correspond to some structures in G, called gadgets. The compression requirement means that our gadgets need to share vertices. To make our lives slightly easier, we apply Sparsification Lemma 14, which allows for assuming that the number of clauses is $O(n)$.

Note that by using the Sparsification Lemma we tweak our general plan a bit: instead of creating one equivalent instance, we are going to create $2^{\varepsilon n}$ instances (for arbitrarily small ϵ), each with $O(n^{2/3})$ vertices. Then we use Lemma 16 to further

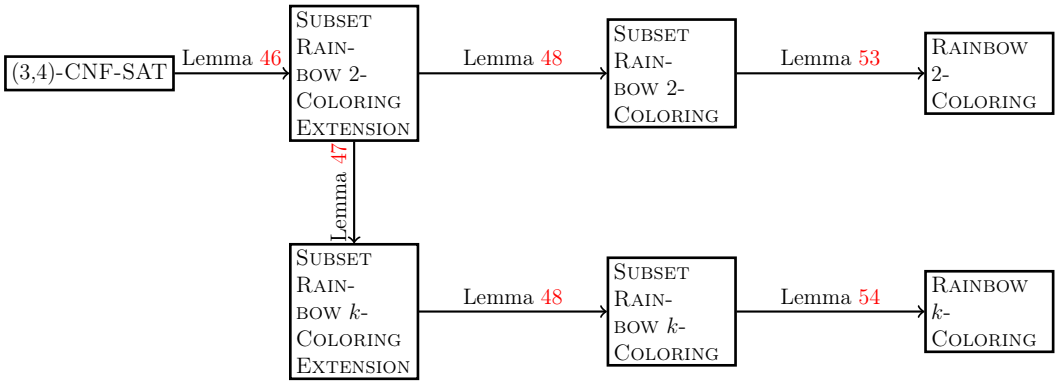

Figure 5.1: The road map of our reductions.

simplify the instance in such a way that each variable occurs in at most 4 clauses, indeed, we obtain an instance of (3,4)-CNF-SAT.

Now our goal is to transform a (3,4)-CNF-SAT formula ϕ with n variables to a graph with $O(n^{2/3})$ vertices — an equivalent instance of Rainbow k-Coloring. We do it in four steps (see Fig 5.1).

In the first step we transform ϕ to an instance $I = (G, S, c_0)$ of Subset Rainbow 2-Coloring Extension, which is a generalization of Subset Rainbow 2-Coloring, where c_0, called a *precoloring*, is a partial coloring of the edges of G into two colors and the goal is to determine if there is an edge-coloring of $E(G)$ which extends c_0 and such that all pairs of S are satisfied. The first step is crucial, because here the compression takes place: $|V(G)| = O(n^{2/3})$ and $E(G) = O(n)$. The major challenge in the construction is avoiding interference between gadgets that share a vertex: to this end we define various conflict graphs and we show that they can be vertex-colored in a few colors. This reduction is described in Section 5.2.

In the second step (Lemma 47) we reduce Subset Rainbow 2-Coloring Extension to Subset Rainbow k-Coloring Extension, for every $k \geq 3$. In the third step (Lemma 48) an instance of Subset Rainbow k-Coloring Extension is transformed to an instance of Subset Rainbow k-Coloring, for every $k \geq 2$. The number of the vertices in the resulting instance does not increase more than by a constant factor. These steps are rather standard, though some technicalities appear because we need to guarantee additional properties of the output instance, which are needed by the reduction in the fourth step.

The last step (Section 5.3), where we reduce an instance $(G = (V, E), S)$ of Subset Rainbow k-Coloring to an instance G' of Rainbow k-Coloring, is yet another challenge. We would like to get rid of the set of requests somehow. For simplicity, let us focus on the $k = 2$ case now. Here, the natural idea, used actually by Chakraborty *et al.* [22] is to create, for every $\{u, v\} \notin S$, a path (u, x_{uv}, v) through a new vertex x_{uv}. Such a path cannot help any of the requests $\{u', v'\} \in S$ to get satisfied (since if it creates a new path P' between u' and v', then P' has length at least 3), and by coloring it into two different colors we can satisfy $\{u, v\}$. Unfortunately, in our case we cannot afford for creating a new vertex for every such $\{u, v\}$, because that would

result in a quadratic blow up in the number of vertices. However, one can observe that for any biclique (a complete bipartite subgraph) in the graph $(V, \binom{V}{2} \setminus S)$ it is sufficient to use just one such vertex x (connected to all the vertices of the biclique). By applying a result of Jukna [79] we can show that in our specific instance of Subset Rainbow 2-Coloring which results from a (3,4)-CNF-SAT formula, the number of bicliques needed to cover all the pairs in $\binom{V}{2} \setminus S$ is small enough. We show a $2^{|V(G)|}|V(G)|^{O(1)}$-time algorithm to find such a cover. Although this algorithm does not seem fast, in our case $|V(G)| = O(n^{2/3})$, so this complexity is *subexponential* in the number of variables of the input formula, which is enough for our goal. The case of $k \geq 3$ is similar, i.e., we also use the biclique cover. However, the details are much more technical because for each biclique we need to introduce a much more complex gadget.

5.2 From (3,4)-CNF-SAT to Subset Rainbow k-Coloring

Let Subset Rainbow k-Coloring Extension be a generalization of Subset Rainbow k-Coloring, where c_0 is a partial k-coloring of the edges of G and the goal is to determine if there is an edge-coloring of $E(G)$ which extends c_0 and such that all pairs of S are satisfied. In the beginning of this section we show a reduction (Lemma 46) from (3,4)-CNF-SAT to Subset Rainbow 2-Coloring Extension.

For an instance $I = (G, S, c_0)$ of Subset Rainbow k-Coloring Extension (for any $k \geq 2$), let us define a *precoloring conflict graph* CG_I. Its vertex set is the set of colored edges, i.e., $V(CG_I) = \mathrm{Dom}(c_0)$. Two different colored edges e_1 and e_2 (treated as vertices of CG_I) are adjacent in CG_I when they are incident in G or there is a pair of endpoints $u \in e_1$ and $v \in e_2$ such that $uv \in E(G) \cup S$.

In what follows the reduction in Lemma 46 is going to be pipelined with further reductions going through Subset Rainbow k-Coloring Extension and Subset Rainbow k-Coloring to Rainbow k-Coloring. In these three reductions we need to keep the instance small. To this end, the instance of Subset Rainbow 2-Coloring Extension resulting in Lemma 46 has to satisfy some additional properties, which are formulated in the claim of Lemma 46. Their role will become clearer later on.

Lemma 46. *Given a* (3,4)-CNF-SAT *formula φ with n variables one can construct in polynomial time an equivalent instance (G, S, c_0) of* Subset Rainbow 2-Coloring Extension *such that G has $O(n^{2/3})$ vertices and $O(n)$ edges. Moreover, $\Delta(G) = O(n^{1/3})$, $\Delta(V(G), S) = O(n^{1/3})$, $|\mathrm{Dom}(c_0)| = O(n^{2/3})$ and along with the instance $I = (G, S, c_0)$ the algorithm constructs a proper vertex 4-coloring of $(V(G), E \cup S)$ (so also of $(V(G), S)$) and a proper vertex $O(n^{1/3})$-coloring of the precoloring conflict graph CG_I.*

Proof. Let m denote the number of clauses in φ. Observe that $m \leq \frac{4}{3}n$. Let Var and Cl denote the sets of variables and clauses of φ. For more clarity, the two colors of the

partial coloring c_0 will be called T and F. Let us describe the graph G along with a set of anti-edges S. Graph G consists of two disjoint vertex subsets: the variable part and the clause part. The intuition is that in any 2-edge coloring of G that extends c_0 and satisfies all pairs in S

- edge colors in the variable part represent an assignment of the variables of φ,
- edge colors in the clause part represent a choice of literals that satisfy all the clauses, and
- edge colors between the two parts make the values of the literals from the clause part consistent with the assignment represented by the variable part.

The variable part. The vertices of the variable part consist of the *middle set* M and $\lceil n^{1/3} \rceil$ layers $L_1 \cup L_2 \cdots \cup L_{\lceil n^{1/3} \rceil}$. The middle set M consists of vertices m_i for each $i = 1, \ldots, \lceil n^{2/3} \rceil + 9$. For every $i = 1, \ldots, \lceil n^{1/3} \rceil$ the layer L_i consists of two parts: upper $L_i^{\uparrow} = \{u_{i,j} \,:\, j = 1, \ldots, \lceil n^{1/3} \rceil + 3\}$ and lower $L_i^{\downarrow} = \{l_{i,j} \,:\, j = 1, \ldots, \lceil n^{1/3} \rceil + 3\}$.

We are going to define four functions: $\mathrm{mid} : \mathrm{Var} \to M$, $\mathrm{lay}, \mathrm{up}, \mathrm{low} : \mathrm{Var} \to [\lceil n^{1/3} \rceil]$. Then, for every variable $x \in \mathrm{Var}$ we add two edges

$$u_{\mathrm{lay}(x),\mathrm{up}(x)}\mathrm{mid}(x), \quad \mathrm{mid}(x) l_{\mathrm{lay}(x),\mathrm{low}(x)}.$$

Moreover, we add the pair $p_x = \{u_{\mathrm{lay}(x),\mathrm{up}(x)}, l_{\mathrm{lay}(x),\mathrm{low}(x)}\}$ to S. In other words, x corresponds to the 2-path $u_{\mathrm{lay}(x),\mathrm{up}(x)}\mathrm{mid}(x) l_{\mathrm{lay}(x),\mathrm{low}(x)}$. Now we describe a careful construction of the four functions, that guarantee several useful properties (for example edge-disjointness of paths corresponding to different variables).

Let us define the *variable conflict graph* $G_V = (\mathrm{Var}, E_{G_V})$, where for two variables $x, y \in \mathrm{Var}$ we have xy are adjacent iff they both occur in the same clause. Since every variable occurs in at most 4 clauses, $\Delta(G_V) \leq 8$. It follows that there is a proper vertex 9-coloring $\alpha : Var \to [9]$ of G_v, and it can be found by a simple linear time algorithm. Next, each of the 9 color classes $\alpha^{-1}(i)$ is partitioned into $\left\lceil |\alpha^{-1}(i)| / \lceil n^{1/3} \rceil \right\rceil$ disjoint groups, each of size at most $\lceil n^{1/3} \rceil$. It follows that the total number n_g of groups is at most $\lceil n^{2/3} \rceil + 9$. Let us number the groups arbitrarily from 1 to n_g and for every variable $x \in \mathrm{Var}$, let $g(x)$ be the number of the group that contains x. Then we define $\mathrm{mid}(x) = m_{g(x)}$. Since any group contains only vertices of the same color we can state the following property:

(P_1) If variables x and y occur in the same clause then $\mathrm{mid}(x) \neq \mathrm{mid}(y)$.

Now, for every variable x we define its layer, i.e., the value of the function $\mathrm{lay}(x)$. Recall that for every $i = 1, \ldots, \lceil n^{2/3} \rceil + 9$ the i-th group $\mathrm{mid}^{-1}(m_i)$ contains at most $\lceil n^{1/3} \rceil$ variables. Inside each group, number the variables arbitrarily and let $\mathrm{lay}(x)$ be the number of variable x in its group, $\mathrm{lay}(x) \in [n^{1/3}]$. This implies another important property.

(P_2) If variables x and y belong to the same layer then $\mathrm{mid}(x) \neq \mathrm{mid}(y)$.

Observe that every layer gets assigned at most $\lceil n^{2/3} \rceil + 9$ variables. For every layer L_i pick any injective function $h_i : \mathrm{lay}^{-1}(i) \to [\lceil n^{1/3} \rceil + 3]^2$. Then, for every variable $x \in \mathrm{Var}$ we put $(\mathrm{up}(x), \mathrm{low}(x)) = h_{\mathrm{lay}(x)}(x)$. Note that by (P_2) we have the following.

- (P_3) For every variable x there is exactly one 2-path in G connecting p_x, namely $(u_{\mathrm{lay}(x),\mathrm{up}(x)}, \mathrm{mid}(x), l_{\mathrm{lay}(x),\mathrm{low}(x)})$.
- (P_4) For every pair of variables x, y the two unique paths connecting p_x and p_y are edge-disjoint.

Although we are going to add more edges and vertices to G, none of these edges has any endpoint in $\bigcup_i L_i$, so P_3 will stay satisfied.

The clause part. The vertices of the clause part are partitioned into $O(m^{1/3})$ clusters. Similarly as in the case of variables, each clause is going to correspond to a pair of vertices in the same cluster. Again, the assignment of clauses to clusters has to be done carefully. To this end we introduce the *clause conflict graph* $G_C = (\mathrm{Cl}, E_{G_C})$. Two different clauses C_1 and C_2 are adjacent in G_C if C_1 contains a variable x_1 and C_2 contains a variable x_2 such that $\mathrm{mid}(x_1) = \mathrm{mid}(x_2)$. Fix a variable x_1. Since $|\mathrm{mid}^{-1}(\mathrm{mid}(x_1))| \leq \lceil n^{1/3} \rceil$, there are at most $\lceil n^{1/3} \rceil$ variables x_2 such that $\mathrm{mid}(x_1) = \mathrm{mid}(x_2)$. Since every clause contains 3 variables, and each of them is in at most 4 clauses, $\Delta(G_C) \leq 12 \lceil n^{1/3} \rceil$. It follows that in polynomial time we can find a proper coloring β of the vertices of G_C into at most $12 \lceil n^{1/3} \rceil + 1$ colors. Moreover, if for any color j its color class $\beta^{-1}(j)$ is larger than $\lceil n^{2/3} \rceil$ we partition it into $\lceil |\beta^{-1}(j)| / \lceil n^{2/3} \rceil \rceil$ new colors. Clearly, in total we produce at most $\frac{4}{3} \lceil n^{1/3} \rceil$ new colors in this way because $m \leq \frac{4}{3} n \leq \frac{4}{3} \lceil n^{1/3} \rceil \cdot \lceil n^{2/3} \rceil$. Hence, in what follows we assume that each color class of β is of size at most $\lceil n^{2/3} \rceil$, and the total number of colors $s \leq 14 \lceil n^{1/3} \rceil + 1$. In what follows we construct s clusters $Q_1, \ldots, Q_s$. Every clause $C \in \mathrm{Cl}$ is going to correspond to a pair of vertices in the cluster $Q_{\beta(C)}$.

Fix $i = 1, \ldots, s$. Let us describe the subgraph induced by cluster Q_i. Define *cluster conflict graph* $G_i = (\beta^{-1}(i), E_{G_i})$. Two different clauses $C_1, C_2 \in \beta^{-1}(i)$ are adjacent in G_i if there are three variables x_1, x_2, and x_3 such that (i) C_1 contains x_1, (ii) C_2 contains x_2, (iii) $(\mathrm{lay}(x_1), \mathrm{up}(x_1)) = (\mathrm{lay}(x_3), \mathrm{up}(x_3))$ and (iv) $\mathrm{mid}(x_2) = \mathrm{mid}(x_3)$. Fix a variable x_1 which appears in a clause $C_1 \in \beta^{-1}(i)$. By our construction, there are at most $\lceil n^{1/3} \rceil + 2$ other variables x_3 that map to the same pair as x_1 by functions lay and up. For each such x_3 there are at most $\lceil n^{1/3} \rceil$ variables x_2 such that $\mathrm{mid}(x_2) = \mathrm{mid}(x_3)$; however, at most one of these variables belongs to a clause C_2 from the same cluster $\beta^{-1}(i)$, by the definition of the coloring β. It follows that $\Delta(G_i) \leq 12(\lceil n^{1/3} \rceil + 2)$. Hence in polynomial time we can find a proper coloring γ_i of the vertices of G_i into at most $12(\lceil n^{1/3} \rceil + 2) + 1$ colors. Similarly as in the case of the coloring β, we can assume that each of the color classes of γ_i has at most $\lceil n^{1/3} \rceil$ clauses, at the expense of at most $\lceil n^{1/3} \rceil$ additional colors. It follows that we can construct in polynomial time a function $g : \mathrm{Cl} \to [\lceil n^{1/3} \rceil]$ such that for every cluster $i = 1, \ldots, s$ and for every color class S of γ_i g is injective on S. Let $n_i \leq 13 \lceil n^{1/3} \rceil + 25$

be the number of colors used by γ_i. For notational convenience, let us define a function $\gamma : \mathrm{Cl} \to [\max_i n_i]$ such that for any clause C we have $\gamma(C) = \gamma_{\beta(C)}(C)$.

We are ready to define the vertices and edges of Q_i. It is a union of three disjoint vertex sets A_i, B_i, and C_i. We have $A_i = \{a_{i,j} \,:\, j = 1, \ldots, \lceil n^{1/3} \rceil\}$, $B_i = \{b^k_{i,j} \,:\, j = 1, \ldots, n_i, k = 1, 2, 3\}$, and $C_i = \{c_{i,j} \,:\, j = 1, \ldots, n_i\}$. For every $j = 1, \ldots, n_i$ and for every $k = 1, 2, 3$ we add edge $c_{i,j}b^k_{i,j}$ to G, and we color it by c_0 to color F. (These are the only edges pre-colored in the whole graph G.) For every clause $C \in \beta^{-1}(i)$ we do the following. For each $k = 1, 2, 3$, add the edge $(a_{i,g(C)}, b^k_{i,\gamma(C)})$ to G. Finally, add the pair $\{a_{i,g(C)}, c_{i,\gamma(C)}\}$ to S. Clearly, the following holds:

(P_5) Let C be any clause. Let $i = \beta(C)$ and let $j = g(C)$. Then there are exactly three 2-paths between $a_{\beta(C),g(C)}$ and $c_{\beta(C),\gamma(C)}$, each going through $b^k_{\beta(C),\gamma(C)}$ for $k = 1, 2, 3$.

Connections between the two parts. Consider a clause $C = \{\ell_1, \ell_2, \ell_3\}$ and its k-th literal ℓ_k for each $k = 1, 2, 3$. Then for some variable x we have $\ell_k = x$ or $\ell_k = \bar{x}$. We add the edge $b^k_{\beta(C),\gamma(C)}\mathrm{mid}(x)$ and we add the pair $\{\mathrm{mid}(x), a_{\beta(C),g(C)}\}$ to S. If $\ell_k = x$, we also add the pair $\{b^k_{\beta(C),\gamma(C)}, u_{\mathrm{lay}(x),\mathrm{up}(x)}\}$ to S; otherwise we add the pair $\{b^k_{\beta(C),\gamma(C)}, l_{\mathrm{lay}(x),\mathrm{low}(x)}\}$ to S. We claim the following.

(P_6) Every edge between the two parts was added exactly once, i.e., for every edge uv such that u is in the clause part and v is in the variable part, there is exactly one clause C and exactly one literal $\ell_k \in C$ such that $u = b^k_{\beta(C),\gamma(C)}$ and $v = \mathrm{mid}(x)$, where x is the variable in ℓ_k.

Indeed, assume for a contradiction that there is a clause C_1 with its k_1-th literal containing x_1 and a clause C_2 with its k_2-th literal containing x_2 such that $b^{k_1}_{\beta(C_1),\gamma(C_1)} = b^{k_2}_{\beta(C_2),\gamma(C_2)}$ and $\mathrm{mid}(x_1) = \mathrm{mid}(x_2)$. Then $C_1 \neq C_2$ by (P_1). Since $\mathrm{mid}(x_1) = \mathrm{mid}(x_2)$, C_1 and C_2 are adjacent in the clause conflict graph G_C. It follows that $\beta(C_1) \neq \beta(C_2)$, so two different clusters share a vertex, a contradiction.

This finishes the description of the instance (G, S, c_0). (See Fig. 5.2.)

Size and time. The construction clearly takes polynomial time. In the variable part we have $|M| = \lceil n^{2/3} \rceil + 9$ and each of the $\lceil n^{1/3} \rceil$ layers contains $2\lceil n^{1/3} \rceil + 6$ vertices. It follows that the variable part contains $O(n^{2/3})$ vertices. In the clause part we have $s = O(n^{1/3})$ clusters. For each $i = 1, \ldots, s$, the cluster Q_i has $4n_i + \lceil n^{1/3} \rceil = O(n^{1/3})$ vertices. Hence the clause part also has $O(n^{2/3})$ vertices.

In the variable part there are two edges per variable, so $2n$ in total. In the clause part there are $3n_i$ edges in i-th cluster, i.e., $O(n^{2/3})$ in total, and 3 edges per clause, i.e., $3m = O(n)$ in total. Finally, for every clause we added 3 edges between the variable part and the clause part, so $3m = O(n)$ in total. It follows that $|E(G)| = O(n)$.

Maximum degree of G. Consider a vertex $u_{i,j} \in L^{\uparrow}_i$, for some $i = 1, \ldots, \lceil n^{1/3} \rceil$ and $j = 1, \ldots, \lceil n^{1/3} \rceil + 3$. The only edges incident to $u_{i,j}$ are those of the form $u_{i,j}\mathrm{mid}(x)$, for some variable x such that $\mathrm{lay}(x) = i$ and $\mathrm{up}(x) = j$. Since h_i is injective, there

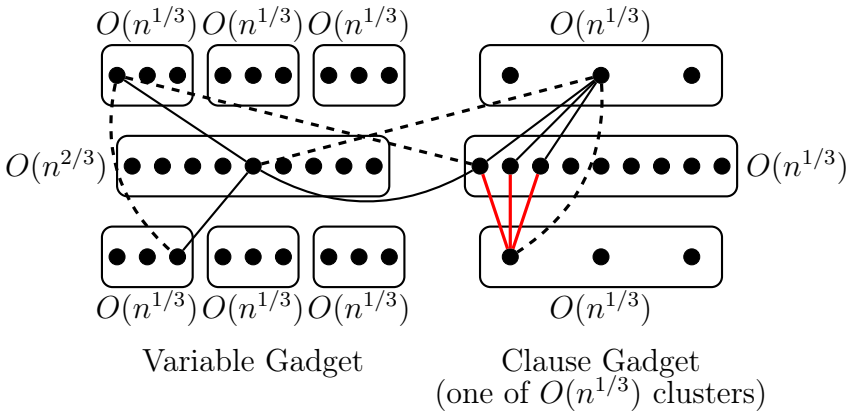

Figure 5.2: A simplified view of the obtained instance. Edges (solid lines) and requests (dashed lines) representing one variable and one clause that contains this variable are presented on the picture.

are at most $\lceil n^{1/3} \rceil + 3$ variables in $\text{lay}^{-1}(i) \cap \text{up}^{-1}(j)$. Hence $\deg_G(u_{i,j}) \leq \lceil n^{1/3} \rceil + 3$. Analogously, the same bound holds for vertices in $\bigcup_i L_i^{\downarrow}$. Consider a vertex m_j in M. For every variable $x \in \text{mid}^{-1}(m_j)$ vertex m_j is adjacent with exactly two vertices in the variable part (one in $\bigcup_i L_i^{\uparrow}$, one in $\bigcup_i L_i^{\downarrow}$), which results in $2\lceil n^{1/3} \rceil$ incident edges in total. Moreover, for every variable $x \in \text{mid}^{-1}(m_j)$ vertex m_j has at most 4 edges to the clause part, since x occurs in at most 4 clauses. It follows that $\deg_G(m_j) \leq 6\lceil n^{1/3} \rceil$. Now we focus on the clause part. Since every cluster has $O(n^{1/3})$ vertices, and there are no edges between the clusters, every vertex in the graph induced by the clause part has degree $O(n^{1/3})$. Vertices in $\bigcup_i A_i \cup \bigcup_i C_i$ have no more edges in G. It remains to consider an arbitrary vertex $b_{i,j}^k$ and count the edges connecting it to the variable part. These edges are of the form $b_{i,j}^k \text{mid}(x)$, for some variable x that occurs in a clause C such that $\beta(C) = i$ and $\gamma_i(C) = j$. Since $|\gamma_i^{-1}(j)| \leq n_i$, there are $O(n^{1/3})$ such clauses C, and each of them contains three variables. Hence there are $O(n^{1/3})$ edges connecting $b_{i,j}^k$ and the variable part and $\deg_G(b_{i,j}^k) = O(n^{1/3})$. To sum up, $\Delta(G) = O(n^{1/3})$, as required.

Maximum degree of $G_S = (V, S)$. Let us inspect each kind of vertices in V separately.

First consider a vertex $u_{i,j} \in L_i^{\uparrow}$ in the variable part. It is incident with two kinds of edges in G_S. The edges of the first kind are of the form $\{u_{i,j}, l_{i,\text{low}(x)}\}$, for some variable x such that $\text{lay}(x) = i$ and $\text{up}(x) = j$. Since $|\text{up}^{-1}(j)| \leq \lceil n^{1/3} \rceil + 3$, we get that $u_{i,j}$ is incident with at most $\lceil n^{1/3} \rceil + 3$ edges of the first kind. The edges of the second kind are of the form $\{b_{\beta(C),\gamma(C)}^k, u_{i,j}\}$, for some variable x in k-th literal of a clause C such that $\text{lay}(x) = i$ and $\text{up}(x) = j$. Since $|\text{up}^{-1}(j)| \leq \lceil n^{1/3} \rceil + 3$ and x is in at most 4 clauses, we get that $u_{i,j}$ is incident with at most $4(\lceil n^{1/3} \rceil + 3)$ edges of the second kind. It follows that $\deg_{G_S}(u_{i,j}) = O(n^{1/3})$. In an analogous way we can bound the degree of vertices in sets $L_i^{\downarrow}$.

Consider a vertex $m_j \in M$ in the variable part. It is incident with edges of

the form $\{m_j, a_{\beta(C),g(C)}\}$, for some variable x from a clause C such that $\mathrm{mid}(x) = m_j$. Since $|\mathrm{mid}^{-1}(m_j)| \leq \lceil n^{1/3} \rceil$ and every variable is in at most 4 clauses, we get $\deg_{G_S}(m_j) = O(n^{1/3})$.

Consider a vertex $a_{i,j}$ in the clause part. It is incident with at most $n_i = O(n^{1/3})$ edges of the form $\{a_{i,j}, c_{i,\gamma(C)}\}$, since $|C_i| = n_i$. It is also incident with edges of the form $\{\mathrm{mid}(x), a_{i,j}\}$, where x is a variable in a clause C such that $\beta(C) = i$ and $g(C) = j$. Since g is injective on every color class of γ_i, so $|g^{-1}(j) \cap \beta^{-1}(i)|$ is bounded by the number of colors in γ_i, which is $O(n^{1/3})$. Hence there are $O(n^{1/3})$ clauses C with $g(C) = j$ and each of them contains three variables, so $\deg_{G_S}(a_{i,j}) = O(n^{1/3})$.

Consider a vertex $b^k_{i,j}$ in the clause part. It is incident with at most $|\gamma^{-1}(j)| \leq n^{1/3}$ edges of the form $\{b^k_{i,j}, u_{\mathrm{lay}(x),\mathrm{up}(x)}\}$ or $\{b^k_{i,j}, l_{\mathrm{lay}(x),\mathrm{low}(x)}\}$, where x is the variable in the k-the literal of a clause C such that $\beta(C) = i$ and $\gamma(C) = j$. It follows that $\deg_{G_S}(b^k_{i,j}) \leq n^{1/3}$.

Finally consider a vertex $c_{i,j}$ in the clause part. It is incident with at most $\lceil n^{1/3} \rceil$ edges of the form $\{a_{i,g(C)}, c_{i,j}\}$, since $|A_i| = \lceil n^{1/3} \rceil$. Hence $\deg_{G_S}(c_{i,j}) \leq \lceil n^{1/3} \rceil$.

To sum up, $\Delta(G_S) = O(n^{1/3})$, as required.

Additional properties. We have $|\mathrm{Dom}(c_0)| = \sum_i 3|C_i| = O(n^{2/3})$. Notice that the vertices of $(V(G), E \cup S)$ can be partitioned into four independent sets as follows: $I_1 = \bigcup_i L_i^{\uparrow} \cup \bigcup_i A_i$, $I_2 = M$, $I_3 = \bigcup_i B_i$, $I_4 = \bigcup_i L_i^{\downarrow} \cup \bigcup_i C_i$. This defines the desired 4-coloring of $(V(G), E \cup S)$. Now consider the precoloring conflict graph CG_{G,S,c_0}. Notice that for every $i = 1, \ldots, s$ cluster Q_i has $3n_i = O(n^{1/3})$ colored edges. For every $i = 1, \ldots, s$, color the vertices of CG_{G,S,c_0} corresponding to the colored edges of Q_i using different colors from 1 to $3n_i$. Notice that two colored edges $e_1, e_2 \in \mathrm{Dom}(c_0)$ that belong to different clusters are not adjacent in the precoloring conflict graph CG_{G,S,c_0}. Hence we defined a proper $O(n^{1/3})$-coloring of CG_{G,S,c_0}.

From an assignment to a coloring. Let $\xi : \mathrm{Var} \to \{T, F\}$ be a satisfying assignment of φ. We claim that there is a coloring c of $E(G)$ which extends c_0 and satisfies all pairs in S. We define c as follows. Denote $\overline{F} = T$, $\overline{T} = F$ and $\xi(\overline{x}) = \overline{\xi(x)}$. For every variable $x \in \mathrm{Var}$ we put $c(u_{\mathrm{lay}(x),\mathrm{up}(x)}\mathrm{mid}(x)) = \xi(x)$ and $c(\mathrm{mid}(x)l_{\mathrm{lay}(x),\mathrm{low}(x)}) = \overline{\xi(x)}$. By (P_3) and (P_4) each edge is colored exactly once. Note that it satisfies all the pairs in S between vertices in the variable part.

For each clause C and each of its literals ℓ_k do the following. Let us color the edge $a_{\beta(C),g(C)}b^k_{\beta(C),\gamma(C)}$ with the color $\xi(\ell_k)$. Since g is injective on color classes of $\gamma_{\beta(C)}$, after processing all the literals in all the clauses, no edge is colored more than once. Recall that for every clause C we added exactly one pair to S, namely $\{a_{\beta(C),g(C)}, c_{\beta(C),\gamma(C)}\}$. Pick any of C's satisfied literals, say ℓ_k. Note that the pair $\{a_{\beta(C),g(C)}, c_{\beta(C),\gamma(C)}\}$ is then satisfied, because edge $a_{\beta(C),g(C)}b^k_{\beta(C),\gamma(C)}$ is colored by T and $b^k_{\beta(C),\gamma(C)}c_{\beta(C),\gamma(C)}$ is colored by F. Hence all the pairs in S between vertices in the clause part are satisfied.

Now let us color the edges between the clause part and the variable part. Consider any such edge uv, i.e., u is in the clause part and v is in the variable part. By (P_6), there is exactly one clause C and exactly one literal $\ell_k \in C$ such that $u = b^k_{\beta(C),\gamma(C)}$ and $v = \mathrm{mid}(x)$, where x is the variable in ℓ_k. Color the edge

$b^k_{\beta(C),\gamma(C)}\mathrm{mid}(x)$ with the color $\overline{\xi(\ell_k)}$. Then the pair $\{\mathrm{mid}(x), a_{\beta(C),g(C)}\}$ is satisfied by the path $(\mathrm{mid}(x), b^k_{\beta(C),\gamma(C)}, a_{\beta(C),g(C)})$, since $c(b^k_{\beta(C),\gamma(C)}a_{\beta(C),g(C)}) = \xi(\ell_k)$. Assume $\ell_k = x$. Then the pair

$$\{b^k_{\beta(C),\gamma(C)}, u_{\mathrm{lay}(x),\mathrm{up}(x)}\}$$

is satisfied by the path $(b^k_{\beta(C),\gamma(C)}, \mathrm{mid}(x), u_{\mathrm{lay}(x),\mathrm{up}(x)})$, since its first edge is colored by $\overline{\overline{\xi(\ell_k)}} = \overline{\overline{\xi(x)}}$ and its second edge is colored by $\xi(x)$. Analogously, when $\ell_k = \bar{x}$, then the pair $\{b^k_{\beta(C),\gamma(C)}, l_{\mathrm{lay}(x),\mathrm{low}(x)}\}$ is satisfied by the path $(b^k_{\beta(C),\gamma(C)}, \mathrm{mid}(x), l_{\mathrm{lay}(x),\mathrm{low}(x)})$, since its first edge is colored by $\overline{\xi(\ell_k)} = \xi(x)$ and its second edge is colored by $\overline{\xi(x)}$.

It follows that we colored all the edges and all the pairs in S are satisfied, so (G, S, c_0) is a YES-instance, as required.

From a coloring to an assignment. Let $c : E(G) \to \{T, F\}$ be a coloring which extends c_0 and satisfies all pairs in S. Consider the following variable assignment: for every $x \in \mathrm{Var}$, we put $\xi(x) = c(u_{\mathrm{lay}(x),\mathrm{up}(x)}\mathrm{mid}(x))$. We claim that ξ satisfies all the clauses of φ. Consider an arbitrary clause $C = \{\ell_1, \ell_2, \ell_3\}$.

Since the pair $\{a_{\beta(C),g(C)}, c_{\beta(C),\gamma(C)}\}$ is satisfied, there is a 2-color 2-path P between $a_{\beta(C),g(C)}$ and $c_{\beta(C),\gamma(C)}$. Recall that $N(c_{\beta(C),\gamma(C)}) = \{b^k_{\beta(C),\gamma(C)} \ : \ k = 1, 2, 3\}$, so there is $k = 1, 2, 3$ such that $b^k_{\beta(C),\gamma(C)}$ is the internal vertex on P. Since c extends c_0 and $c_0(b^k_{\beta(C),\gamma(C)}c_{\beta(C),\gamma(C)}) = F$, we infer that $c(a_{\beta(C),g(C)}b^k_{\beta(C),\gamma(C)}) = T$. Let x be the variable in the literal ℓ_k.

Since the pair $\{\mathrm{mid}(x), a_{\beta(C),g(C)}\}$ is satisfied, there is a 2-color 2-path Q between $\mathrm{mid}(x)$ and $a_{\beta(C),g(C)}$. Then the internal vertex of Q is $b^{k'}_{\beta(C'),\gamma(C')}$, for some clause C' and integer $k' = 1, 2, 3$. Let y be the variable in the k'-th literal of C'. Since there is an edge between $\mathrm{mid}(x)$ and $b^{k'}_{\beta(C'),\gamma(C')}$, from (P_6) we infer that $\mathrm{mid}(y) = \mathrm{mid}(x)$. If $C = C'$ and $k' \neq k$, then by (P_1) we get that $\mathrm{mid}(x) \neq \mathrm{mid}(y)$, a contradiction. If $C \neq C'$, since $\mathrm{mid}(y) = \mathrm{mid}(x)$, the clauses C and C' are adjacent in the clause conflict graph G_C, so $\beta(C') \neq \beta(C)$. However, then the edge $b^{k'}_{\beta(C'),\gamma(C')}a_{\beta(C),g(C)}$ of Q goes between two clusters, a contradiction. Hence $C' = C$ and $k' = k$, i.e., $Q = (\mathrm{mid}(x), b^k_{\beta(C),\gamma(C)}, a_{\beta(C),g(C)})$. Since $c(b^k_{\beta(C),\gamma(C)}a_{\beta(C),g(C)}) = T$, we get $c(\mathrm{mid}(x)b^k_{\beta(C),\gamma(C)}) = F$. Now assume w.l.o.g. that $\ell_k = x$, the case $\ell_k = \bar{x}$ is analogous.

Since the pair $\{b^k_{\beta(C),\gamma(C)}, u_{\mathrm{lay}(x),\mathrm{up}(x)}\}$ is satisfied, there is a 2-color 2-path R between $b^k_{\beta(C),\gamma(C)}$ and $u_{\mathrm{lay}(x),\mathrm{up}(x)}$. Then the internal vertex z of R belongs to M. By (P_6) there is a literal ℓ_k which belongs to a clause C_2 and contains a variable x_2 such that $z = \mathrm{mid}(x_2)$ and $b^k_{\beta(C),\gamma(C)} = b^k_{\beta(C_2),\gamma(C_2)}$. In particular, $\beta(C) = \beta(C_2)$ and $\gamma(C) = \gamma(C_2)$. Assume $C_2 \neq C$. There is a variable, say x_3, corresponding to edge $\mathrm{mid}(x_2)u_{\mathrm{lay}(x),\mathrm{up}(x)}$, i.e., $\mathrm{mid}(x_2) = \mathrm{mid}(x_3)$ and $u_{\mathrm{lay}(x),\mathrm{up}(x)} = u_{\mathrm{lay}(x_3),\mathrm{up}(x_3)}$. It follows that C and C_2 are adjacent in $G_{\beta(C)}$, which contradicts the fact that $\gamma(C) = \gamma(C_2)$. Hence $C_2 = C$, i.e., there is exactly one 2-path between $b^k_{\beta(C),\gamma(C)}$ and $u_{\mathrm{lay}(x),\mathrm{up}(x)}$, and it goes through $\mathrm{mid}(x)$. Since $c(\mathrm{mid}(x)b^k_{\beta(C),\gamma(C)}) = F$ and the path is 2-color, we get that $c(u_{\mathrm{lay}(x),\mathrm{up}(x)}\mathrm{mid}(x)) = T$. Hence $\xi(\ell_k) = \xi(x) = T$, so clause C is satisfied, as required. It finishes the proof. □

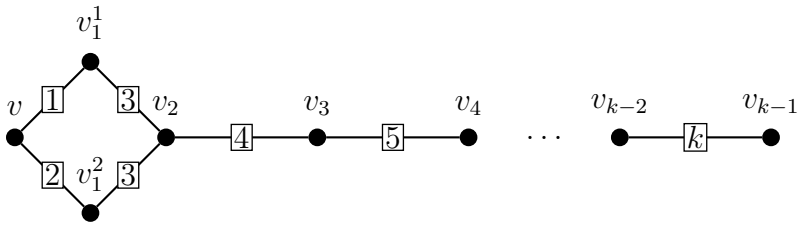

Figure 5.3: The gadget added to every vertex $v \in V$ in Lemma 47.

Lemma 47. *For any fixed $k \geq 3$, there is a polynomial time algorithm which given an instance $I = (G = (V, E), S, c_0)$ of* Subset Rainbow 2-Coloring Extension *constructs an equivalent instance $I' = (G' = (V', E'), S', c_0')$ of* Subset Rainbow k-Coloring Extension *such that $|V'| = O(k|V|)$, $|E'| = |E| + O(k|V|)$, $|S'| = |S| + |E|$, $\Delta(G') \leq \Delta_1(G) + 2$ and $|\mathrm{Dom}(c_0')| = |\mathrm{Dom}(c_0)| + O(k|V|)$. Let $G_S = (V, S)$ and $G_{S'} = (V', S')$. Then $\Delta(G_{S'}) \leq \Delta(G) + \Delta(G_S)$. Moreover, given a proper vertex p-coloring of G_S the algorithm outputs also a $(p+1)$-coloring of $G_{S'}$. Also, given a proper vertex q-coloring of $G_{E \cup S} = (V, E \cup S)$ and a proper vertex ℓ-coloring of the precoloring conflict graph CG_I the algorithm outputs a proper $(\ell + O(q))$-coloring of the precoloring conflict graph $CG_{I'}$.*

Proof. **Construction.** Let us denote the colors of c_0 by 1 and 2. Let

$$V' = V \cup \bigcup_{v \in V} \{v_1^1, v_1^2, v_2, v_3, \ldots, v_{k-1}\},$$

$$E' = E \cup \bigcup_{v \in V} \{vv_1^1, v_1^1v_2, vv_1^2, v_1^2v_2, v_2v_3, v_3v_4, \ldots, v_{k-2}v_{k-1}\},$$

$$S' = S \cup \{u_{k-1}v \ : \ uv \in E \text{ and } u < v\}$$

and $\mathrm{Dom}(c_0') = \mathrm{Dom}(c_0) \cup (E' \setminus E)$. Let $c_0'|_{\mathrm{Dom}(c_0)} = c_0$ and let for every $v \in V$ $c_0'(vv_1^1) = 1$, $c_0'(vv_1^2) = 2$, $c_0'(v_1^1v_2) = 3$, $c_0'(v_1^2v_2) = 3$, and $c_0'(v_iv_{i+1}) = i + 2$ for every $i \in \{2, 3, \ldots, k-2\}$. (See Fig 5.3.) Note that for every vertex u_{k-1} such that $u \in V$ we have $\deg_{G_{S'}}(u_{k-1}) \leq \deg_G(u)$ and for every vertex $v \in V$ we have $\deg_{G_{S'}}(v) \leq \deg_{G_S}(v) + \deg_G(v)$. Hence $\Delta(G_{S'}) \leq \Delta(G) + \Delta(G_S)$, as required.

Equivalence. Assume $(G = (V, E), S, c_0)$ is a YES-instance of Subset Rainbow 2-Coloring Extension and let c be the corresponding 2-coloring. Define a coloring c' of $E(G')$ as $c'|_E = c$ and $c'|_{E' \setminus E} = c_0'|_{E' \setminus E}$. Note that c' is an extension of c_0'. Moreover, c' satisfies all the pairs of S using rainbow paths in E because c satisfies S. All the pairs of the form $\{u_{k-1}, v\}$ for some $uv \in E$ where $u < v$ are satisfied because one of the paths $vuu_1^1u_2u_3 \ldots u_{k-1}$ and $vuu_1^2u_2u_3 \ldots u_{k-1}$ is rainbow. Hence (G', S', c_0') is a YES-instance of Subset Rainbow k-Coloring Extension.

Now assume that $(G' = (V', E'), S', c_0')$ is a YES instance of Subset Rainbow k-Coloring Extension and let c' be the corresponding k-coloring. Then we claim that $c = c'|_E$ is a coloring of $E(G)$ that satisfies S and extends c_0. It is easy to see that c extends c_0, because c_0' extends c_0 and c' extends c_0'.

Now we show that $c(E) \subseteq \{1, 2\}$. Indeed, for every edge $uv \in E$ such that $u < v$ the distance between the vertices u_{k-1} and v is k. Hence all rainbow paths between u_{k-1} and v are of length exactly k. There are exactly two paths of length k between them, namely $vuu_1^1u_2u_3 \dots u_{k-1}$ and $vuu_1^2u_2u_3 \dots u_{k-1}$. Exactly one of them is rainbow, so $c(uv) = 2$ (if the first one is rainbow) or $c(uv) = 2$ (otherwise), as required.

Finally we show that all pairs in S are satisfied by c. Pick any $\{u, v\} \in S$. Note that no (u, v)-path in G' visits a vertex from $V' \setminus V$ because every vertex $v \in V$ separates the vertices in $\{v_1^1, v_1^2, v_2, v_3, \dots, v_{k-1}\}$ from the rest of the graph. It follows that the rainbow path that satisfies $\{u, v\}$ in c' also satisfies $\{u, v\}$ in c. This finishes the proof that (G, S, c_0) is a YES-instance of Subset Rainbow 2-Coloring Extension.

Additional Properties. The vertex $(p+1)$-coloring of $G_{S'} = (V', S')$ can be easily obtained from the p coloring of $G_S = (V, S)$. Indeed, all the independent sets of $G_S = (V, S)$ are also independent in $G_{S'} = (V', S')$ and all the added vertices $V' \setminus V$ form an independent set in $G_{S'} = (V', S')$. Therefore it is sufficient to extend the input p-coloring with one additional color for the vertices of $V' \setminus V$.

Let $\alpha : \mathrm{Dom}(c_0) \to [\ell]$ be the given vertex ℓ-coloring of CG_I, and let $h : V \to [q]$ be the given vertex q-coloring of the graph $G_{E \cup S} = (V, E \cup S)$. Note that in G' we have not added any edge or requirement between any two vertices of V. Therefore $CG_{I'}[\mathrm{Dom}(c_0)] = CG_I$. Hence it suffices to extend α to a coloring α' of $CG_{I'}$. The remaining elements, i.e., elements from $\mathrm{Dom}(c_0') \setminus \mathrm{Dom}(c_0)$ get new colors, from the set $\{\ell + 1, \ell + 2, \dots \ell + O(q)\}$, as follows. For every vertex $v \in V$ we put $\alpha'(vv_1^1) = \ell + 2h(v) - 1$ and $\alpha'(vv_1^2) = \ell + 2h(v)$. Thus we added $2q$ new colors. Note that their corresponding color classes are independent sets in $CG_{I'}$. In what follows they stay independent because will not use these $2q$ colors any more. Next, for every vertex $v \in V$ we put $\alpha'(v_1^1v_2) = \ell + 2q + 1$ and $\alpha'(v_1^2v_2) = \ell + 2q + 2$. Again, the corresponding color classes are independent in $CG_{I'}$. If $k = 3$ then we have just properly colored all the vertices of $CG_{I'}$, i.e., the edges of $\mathrm{Dom}(c_0')$. If $k > 3$ then for every vertex $v \in V$ we color the remaining path $v_3v_4 \dots v_{k-1}$ using alternating sequence of colors $\ell + 2q + 3, \ell + 2q + 4, \ell + 2q + 5, \ell + 2q + 3, \ell + 2q + 4, \dots$. Thus we have obtained a proper vertex $(\ell + 2q + 5)$-coloring of the graph $CG_{I'}$, as required. □

Lemma 48. *For any fixed $k \geq 2$, given an instance $(G = (V, E), S, c_0)$ of* Subset Rainbow k-Coloring Extension *together with a proper vertex ℓ-coloring of the precoloring conflict graph CG_I one can construct in polynomial time an equivalent instance $(G' = (V', E'), S')$ of* Subset Rainbow k-Coloring *such that $|V'| = |V| + O(k^2\ell)$, $|E'| = |E| + |\mathrm{Dom}(c_0)| + O(k^2\ell)$, $|S'| = |S| + 2|\mathrm{Dom}(c_0)| + O(k^2\ell)$. Let $G_S = (V, S)$ and $G_{S'} = (V', S')$. Then $\Delta(G_{S'}) = O(\Delta(G_S) + \Delta(G) + |\mathrm{Dom}(c_0)|/\ell)$. Moreover if we are given a proper vertex p-coloring of the graph $G_S = (V, S)$ then we can output also a proper vertex $(p + 2)$-coloring of the graph $G_{S'} = (V', S')$.*

Proof. **Construction.** Let $f : \mathrm{Dom}(c_0) \to [\ell]$ be the vertex ℓ-coloring of the precoloring conflict graph CG_I. If there is a color class larger than $\lceil |\mathrm{Dom}(c_0)|/\ell \rceil$ we split it into two colors: one of size $\lceil \mathrm{Dom}(c_0)/\ell \rceil$ and the rest. We repeat this procedure

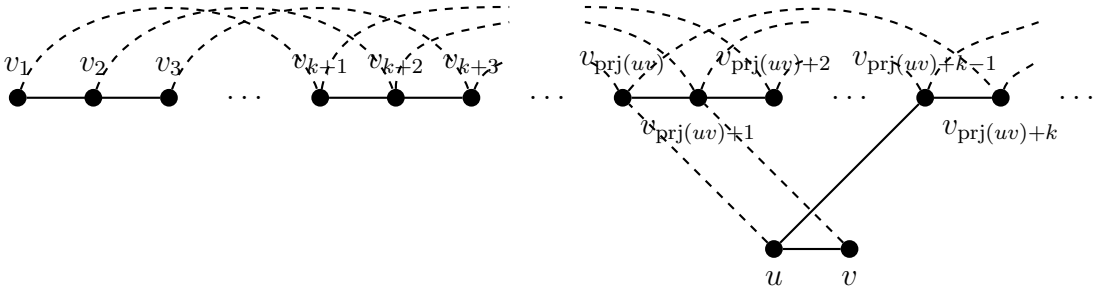

Figure 5.4: Construction in Lemma 48. Dashed lines denote requests.

until none of the color classes is larger than $\lceil |\mathrm{Dom}(c_0)|/\ell \rceil$. This process introduces at most ℓ new colors, since $\ell \cdot \lceil |\mathrm{Dom}(c_0)|/\ell \rceil \geq |\mathrm{Dom}(c_0)|$. Hence in what follows we assume that f is an ℓ'-coloring of CG_I such that each color class is of size at most $\lceil |\mathrm{Dom}(c_0)|/\ell \rceil$, where $\ell' \leq 2\ell$.

Let $V' = V \cup \{v_i : i \in [3k^2\ell']\}$. For every $\alpha \in [\ell']$, $\beta \in [k]$ and $\gamma \in [3k]$ let us denote $\mathrm{id}(\alpha, \beta, \gamma) = (\alpha-1)\cdot 3k^2 + (\beta-1)\cdot 3k + \gamma$. Note that for every $i \in [3k^2\ell']$ there is exactly one triple $(\alpha, \beta, \gamma) \in [\ell'] \times [k] \times [3k]$ such that $i = \mathrm{id}(\alpha, \beta, \gamma)$. Moreover, for every edge $uv \in \mathrm{Dom}(c_0)$, such that $u < v$, we denote $\mathrm{prj}(uv) = \mathrm{id}(f(uv), c_0(uv), c_0(uv))$. Let

$$E' = E \cup \{v_i v_{i+1} : i \in [3k^2\ell' - 1]\} \cup \{v_{\mathrm{prj}(uv)+k-1} u : uv \in \mathrm{Dom}(c_0) \text{ and } u < v\}$$

and

$$\begin{aligned} S' = S \cup \{&\{v_i, v_{i+k}\} : i \in [3k^2\ell' - k]\} \\ &\cup \{\{v_{\mathrm{prj}(uv)}, u\}, \{v_{\mathrm{prj}(uv)+1}, v\} : uv \in \mathrm{Dom}(c_0) \text{ and } u < v\}. \end{aligned}$$

From the above, if we have $uv, u'v' \in \mathrm{Dom}(c_0)$ such that $u < v$, $u' < v'$ and $\mathrm{prj}(uv) = \mathrm{prj}(u'v')$ then $f(uv) = f(u'v')$ and $c_0(uv) = c_0(u'v')$.

Equivalence. Assume c is a k-coloring of E that satisfies all the constraints of S and extends the coloring c_0. We define a coloring $c' : E' \to [k]$ as follows. For every edge $xy \in E$ we put $c'(xy) = c(xy)$. Let us define

$$x \bmod_1 y = 1 + (x-1) \bmod y.$$

The edges of the path $(v_1, \ldots, v_{3k^2\ell'})$ are colored with the sequence $(1, \ldots, k)$ repeated, i.e., for every $i = 1, \ldots, 3k^2\ell' - 1$ we put $c'(v_i v_{i+1}) = i \bmod_1 k$. Finally, for every edge $uv \in E$, such that $u < v$, we put

$$c'(v_{\mathrm{prj}(uv)+k-1} u) = (c_0(uv) - 1) \bmod_1 k.$$

We claim that c' is a k-coloring of E' that satisfies all the constraints of S'. Indeed, every constraint of S is satisfied because c' extends c. Every constraint of the form $\{v_i, v_{i+k}\}$ is satisfied because of the path $(v_i, v_{i+1}, \ldots v_{i+k})$. Finally, note that

$$c'(v_{\mathrm{prj}(uv)}, v_{\mathrm{prj}(uv)+1}) = c_0(uv) \bmod_1 k = c_0(uv)$$

and the path $(v_{\mathrm{prj}(uv)}, v_{\mathrm{prj}(uv)+1}, \ldots, v_{\mathrm{prj}(uv)+k-1})$ uses all the colors except for $(\mathrm{prj}(uv)+k-1) \bmod_1 k = (c_0(uv)-1) \bmod_1 k$. But $c'(v_{\mathrm{prj}(uv)+k-1}u) = (c_0(uv)-1) \bmod_1 k$. So, for every $uv \in \mathrm{Dom}(c_0)$ such that $u < v$ the constraints $\{v_{\mathrm{prj}(uv)}, u\}$ and $\{v_{\mathrm{prj}(uv)+1}, v\}$ are satisfied because of the paths $P^1_{uv} = (v_{\mathrm{prj}(uv)}, v_{\mathrm{prj}(uv)+1}, \ldots, v_{\mathrm{prj}(uv)+k-1}, u)$ and $P^2_{uv} = (v_{\mathrm{prj}(uv)+1}, v_{\mathrm{prj}(uv)+2}, \ldots,$
$v_{\mathrm{prj}(uv)+k-1}, u, v)$, respectively.

Now assume c' is a k-coloring of E' that satisfies all the constraints of S'. The distance between $i = \mathrm{id}(a, s, s)$ and $j = \mathrm{id}(a, s+1, s+1)$ is equal $3k+1$. Moreover the distance between $i = \mathrm{id}(a, k, k)$ and $j = \mathrm{id}(a+1, 1, 1)$ is equal $2k+1$. So if for some different $i, j = 1, \ldots, 3k^2\ell'$ such that $i < j$ both i and j have neighbors in V, then $|i-j| \geq 2k+1$. This has three consequences:

(*i*) for any $i \in [3k^2\ell' - k]$, there is exactly one path of length at most k between v_i and v_{i+k}, namely $(v_i, v_{i+1}, \ldots, v_{i+k})$,

(*ii*) for any $uv \in \mathrm{Dom}(c_0)$, $u < v$, there is exactly one path of length at most k between $v_{\mathrm{prj}(uv)}$ and u, namely P^1_{uv}, and exactly one path of length at most k between $v_{\mathrm{prj}(uv)+1}$ and v, namely P^2_{uv},

(*iii*) for any pair of vertices $u, v \in V$ every path of length at most k between u and v does not contain any edge v_iv_{i+1}, for $i \in [3k^2\ell' - 1]$.

From (*i*) it follows that $(c'(v_1v_2), c'(v_2v_3), \ldots, c'(v_kv_{k+1}))$ is a permutation of all k colors, and this permutation repeats $3k\ell'$ times, i.e., for every $i \in [3k^2\ell' - k - 1]$ we have $c'(v_iv_{i+1}) = c'(v_{k+i}v_{k+i+1})$. Let c'' be the coloring of E' obtained from c' by permuting the colors so that for every $i = 1, \ldots, 3k^2\ell' - 1$ we have $c''(v_iv_{i+1}) = i \bmod_1 k$. Obviously, c'' satisfies S', since c' does. We claim that the coloring $c = c'' \mid_E$ satisfies all the constraints of S and extends c_0.

Consider an arbitrary edge $uv \in \mathrm{Dom}(c_0)$, with $u < v$. Note that $c''(v_{\mathrm{prj}(uv)}, v_{\mathrm{prj}(uv)+1})$ $= \mathrm{prj}(uv) \bmod_1 k = c_0(uv)$. Hence the path $(v_{\mathrm{prj}(uv)}, v_{\mathrm{prj}(uv)+1}, \ldots, v_{\mathrm{prj}(uv)+k-1})$ uses all the colors except for $(c_0(uv)-1) \bmod_1 k$. From (*ii*) and the fact that $\{v_{\mathrm{prj}(uv)}, u\} \in S'$ we infer that P^1_{uv} is rainbow, which implies $c''(v_{\mathrm{prj}(uv)+k-1}u) = (c_0(uv)-1) \bmod_1 k$. Hence the path $(v_{\mathrm{prj}(uv)+1}, \ldots v_{\mathrm{prj}(uv)+k-1}, u)$ uses all the colors except for $c_0(uv)$. From (*ii*) and the fact that $\{v_{\mathrm{prj}(uv)+1}, v\} \in S'$ we infer that P^2_{uv} is rainbow, which implies that $c''(uv) = c_0(uv)$. This shows that c'' extends c_0, and hence so does c.

In the previous paragraph we showed that for every edge $uv \in \mathrm{Dom}(c_0)$, with $u < v$, we have $c''(v_{\mathrm{prj}(uv)+k-1}u) = (c_0(uv) - 1) \bmod_1 k$. Note that if there is an edge $v_{\mathrm{prj}(uv)+k-1}u'$ for some $u' \in V$, it means that there is $u'v' \in \mathrm{Dom}(c_0)$ such that $u' < v'$ and $\mathrm{prj}(uv) = \mathrm{prj}(u'v')$. Hence $c_0(u'v') = c_0(uv)$, and by the previous paragraph, $c''(v_{\mathrm{prj}(uv)+k-1}u') = (c_0(u'v') - 1) \bmod_1 k = (c_0(uv) - 1) \bmod_1 k = c''(v_{\mathrm{prj}(uv)+k-1}u)$. In other words, for every $i = 1, \ldots, 3k^2\ell'$, all the edges between v_i and V have the same color in c''. This, combined with (*iii*) means that for every pair of vertices $u, v \in V$ every rainbow path in c'' is contained in E. Hence, for any $\{u, v\} \in S$ the rainbow path between them in c'' is also a rainbow path in c, so c satisfies $\{u, v\}$. This ends the proof of the equivalence.

Additional properties. Note that for every vertex $v \in V' \setminus V$ we have $\deg_{G_{S'}}(v) \leq 2+\lceil |\mathrm{Dom}(c_0)|/\ell' \rceil$ and for every vertex $v \in V$ we have $\deg_{G_{S'}}(v) \leq \deg_{G_S}(v)+\deg_G(v)$. Hence $\Delta(G_{S'}) = O(\Delta(G_S) + \Delta(G) + |\mathrm{Dom}(c_0)|/\ell)$, as required.

Note that in the graph $G_{S'}[V' \setminus V]$ all the connected components are paths. Hence the vertices of $V' \setminus V$ can be colored using two colors in $G_{S'}$. By merging this coloring with the given p-coloring of the vertices of graph G_S we obtain a proper vertex $(p+2)$-coloring of $G_{S'}$. □

5.3 From Subset Rainbow k-Coloring to Rainbow k-Coloring

The basic idea of our reduction from Subset Rainbow k-Coloring to Rainbow k-Coloring is to modify the graph so that the pairs of vertices from $\bar{E} \setminus S$ can be somehow trivially satisfied, without affecting the satisfiability of S. To this end we use a notion of biclique covering number (called also bipartite dimension). The *biclique covering number* $\mathrm{bc}(G)$ of a graph G is the smallest number of biclique subgraphs of G that cover all edges of G. The following proposition is well-known.

Proposition 49 (Folklore)**.** *It holds that* $\mathrm{bc}(K_n) = \lceil \log n \rceil$*, and the corresponding cover can be constructed in polynomial time.*

Proof. Assume $V(K_n) = \{0, \ldots, n-1\}$. The i-th biclique contains edges between the vertices that have 0 at the i-th bit and the vertices that have 1 at the i-th bit. □

Let $G = (V_1, V_2, E)$ be a bipartite graph. Then $\hat{G}$ denotes the bipartite complement of G, i.e, the bipartite graph

$$(V_1, V_2, \{v_1v_2 \ : \ v_1 \in V_1, \quad v_2 \in V_2, \quad \text{and} \quad v_1v_2 \notin E\}).$$

We will use the following result of Jukna. Recall that we denote $\Delta_1(G) = \max\{\Delta(G), 1\}$.

Theorem 50 (Jukna [79])**.** *If G is an n-vertex bipartite graph, then* $\mathrm{bc}(\hat{G}) = O(\Delta_1(G) \cdot \log n)$.

Let us call the cover from Theorem 50 the *Jukna cover*. In our application we need to be able to *compute* the Jukna cover fast.

Lemma 51. *The Jukna cover can be constructed in (i) expected polynomial time, or (ii) deterministic $2^n n^{O(1)}$ time.*

Proof. Denote $\Delta = \Delta(G)$. If $\Delta = 0$ the claim follows from Proposition 49, so in what follows assume $\Delta \geq 1$. Jukna [79] shows a simple worst-case linear time algorithm which samples a biclique in G. Then it is proved that after sampling t bicliques, the probability that there is an edge not covered by one of the bicliques is at most $n^2 e^{-t/(\Delta e)}$. It follows that the probability that more than $\Delta e(2 \ln n + 1)$ samples are needed is at most e^{-1}. If after $\Delta e(2 \ln n + 1)$ samples some edges is not covered, we

discard all the bicliques found and repeat the whole algorithm from the scratch. The expected number of such restarts is $1/(1-e^{-1}) = O(1)$.

Now we proceed to the second part of the claim. Let $G = (V_1, V_2, E)$. For every subset $A \subseteq V_1$ we define the biclique $B_A = (A, B, E_A)$, where B is the set of vertices of V_2 adjacent in $\hat{G}$ to all vertices of A. Clearly, B_A is a subgraph of $\hat{G}$ and for every subset $A \subseteq V_1$ it can be found in time linear in the size of $\hat{G}$. Our deterministic algorithm works as follows: as long as not all edges of $\hat{G}$ are covered, it picks the biclique B_A which maximizes the number of new covered edges of $\hat{G}$. Since all the bicliques in the set $\{B_A \ : \ A \subseteq V_1\}$ can be listed in time $O(2^n|E(\hat{G})|)$, the total running time is $t2^n n^{O(1)}$, where t is the size of the returned cover. It suffices to show that $t = O(\Delta \log n)$.

Jukna [79] shows that if set A is chosen by picking every vertex of V_1 independently with probability $\frac{1}{\Delta}$, then for any edge $uv \in E(\hat{G})$, $\Pr[uv \in E_A] \geq \frac{1}{\Delta e}$. Consider any step of our algorithm and let $R \subseteq E(\hat{G})$ be the set of the edges of $\hat{G}$ which are not covered yet. By the bound on $\Pr[uv \in E_A]$ and the linearity of expectation a set A sampled as described above covers at least $|R|/(\Delta e)$ new edges in expectation. In particular, it implies that there exists a set $A \subseteq V_1$ that covers at least $|R|/(\Delta e)$ new edges. Let $\alpha = (1 - \frac{1}{\Delta e})^{-1}$. By the Taylor expansion of $\log(1-x)$, it follows that $t = O(\log_\alpha |E(\hat{G})|) = O(\log n / \log \alpha) = O(\Delta \log n)$. □

Lemma 52. *Let G be an n-vertex graph with a given proper vertex p-coloring. Then the edges of $\bar{G}$ can be covered by $O(p^2\Delta_1(G)\log n)$ bicliques from $\bar{G}$ so that any edge of G and any biclique have at most one common vertex. This cover can be constructed in (i) expected polynomial time, or (ii) deterministic $2^n n^{O(1)}$ time.*

Proof. The edges of $\bar{G}$ between the vertices of any color class form a clique, so by Proposition 49 we can cover its edges using $O(\log n)$ bicliques. If an edge of G has both endpoints in such a biclique, these endpoints have the same color, contradiction. For two different colors i and j the edges of G between their color classes form a bipartite graph of maximum degree at most $\Delta(G)$. Hence by Lemma 51 we can cover the edges of its bipartite complement using $O(\Delta_1(G)\log n)$ bicliques. If an edge uv of G has both endpoints in such a biclique, then either (i) these endpoints have the same color, contradiction, or (ii) these endpoints belong to two different parts of the biclique, so uv is in the biclique and hence $uv \in E(\bar{G})$, a contradiction. Summing over all color classes and pairs of color classes, we use $O(p^2\Delta_1(G)\log n)$ bicliques, as required. □

Now we proceed to the actual reduction. Somewhat surprisingly, the $k = 2$ requires a slightly different construction than the $k \geq 3$ case, so we partition the proof into two lemmas.

Lemma 53. *Given an instance $(G = (V, E), S)$ of* Subset Rainbow 2-Coloring *together with a proper p-coloring of the graph $G_S = (V, S)$, one can construct an equivalent instance G' of* Rainbow 2-Coloring *such that $|V(G')| = O(|V| + p^2\Delta_1(G_S) \cdot \log|V|)$, $|E(G')| = O(|E(G)| + (|V| + p^2\Delta_1(G_S)\log|V|) \cdot p^2\Delta_1(G_S)\log|V|)$. The construction algorithm can run in (i) expected polynomial time, or (ii) deterministic $2^{|V|}|V|^{O(1)}$ time.*

Proof. Let us consider a biclique covering of the complement of the graph G_S with $q = O(p^2\Delta_1(G_S)\log n)$ bicliques $(U_1, V_1; E_1), (U_2, V_2; E_2), \ldots, (U_q, V_q; E_q)$ as in Lemma 52. Let $W = \{w_1, w_2, \ldots, w_q\}$, $T = \{t_1, t_2, t_3\}$, $V(G') = V \cup W \cup T$ and $E(G') = E(G) \cup (W \times W) \cup (T \times T) \cup (\{t_2\} \times W) \cup (\{t_3\} \times (V \cup W)) \cup \left(\bigcup_{1 \le i \le q} \{w_i\} \times (U_i \cup V_i)\right)$ (we abuse the notation assuming that $\times$ operator returns *unordered* pairs minus loops).

If (G, S) is a YES-instance then there exists a coloring c_S such that all the constraints in S are satisfied. We extend this coloring to a coloring of $E(G')$ as follows.

$$c(e) = \begin{cases} c_S(e) & \text{for } e \in E(G), \\ 1 & \text{for } e \in (W \times W) \cup (T \times T) \cup (\{t_3\} \times W) \\ & \qquad \cup \left(\bigcup_{1 \le i \le q} \{w_i\} \times U_i\right), \\ 0 & \text{for } e \in (\{t_2\} \times W) \cup (\{t_3\} \times V) \cup \left(\bigcup_{1 \le i \le q} \{w_i\} \times V_i\right). \end{cases}$$

It suffices to show that all anti-edges of G' are satisfied by the coloring c. An anti-edge uv inside the set of the vertices V either belongs to S and it is satisfied by a path in G or it belongs to one of the bicliques $(U_i, V_i; E_i)$ and then it is satisfied by a path uw_iv. Inside W and T all the vertices are connected directly. An anti-edge vw between V and W is connected by a path vt_3w. The vertices of T are connected with V via $\{t_3\}$ and with W via $\{t_2\}$ ($\{t_3\}$ is also connected directly to W). So G' is a YES-instance.

If G' is a YES-instance then there exists a coloring c such that all the anti-edges in G' are satisfied. Note that $S \cap E(G') = \emptyset$. An anti-edge belonging to S cannot be satisfied by a path using any vertex from W because it is not covered by any of the added bicliques. It cannot be also satisfied by a path using vertex t_3 because t_3 is the only common neighbor of t_1 and the vertices of V. Therefore in c all the edges connecting V with t_3 have to be in the same color, i.e., the color different from $c(t_1t_3)$. Moreover t_3 is the only vertex of T that is adjacent to V. Hence every anti-edge in S is satisfied using only paths inside G. Therefore $c\,|_V$ is also a coloring satisfying an instance (G, E, S).

We added only $O(p^2\Delta_1(G_S)\log|V|)$ new vertices and $O((p^2\Delta_1(G_S)\log|V|)^2 + |V|p^2\Delta_1(G_S)\log|V|) = O((|V| + p^2\Delta_1(G_S)\log|V|) \cdot p^2\Delta_1(G_S)\log|V|)$ edges. □

Lemma 54. *For any fixed $k \ge 3$, given an instance $(G = (V, E), S)$ of* Subset Rainbow k-Coloring *together with a p-coloring of the graph $G_S = (V, S)$, one can construct an equivalent instance G' of* Rainbow k-Coloring *such that $|V(G')| = O(|V| + kp^2\Delta_1(G_S)\log|V|)$, $|E(G')| = O(|E(G)| + |V|p^2\Delta_1(G_S)\log|V|)$. The construction algorithm can run in (i) expected polynomial time, or (ii) deterministic $2^{|V|}|V|^{O(1)}$ time.*

Proof. In what follows we assume that G contains an isolated vertex v^*, for otherwise we can just add it (without changing S) and get an equivalent instance of Subset Rainbow k-Coloring. Let us consider a biclique cover of the complement of the graph G_S with $q = O(p^2\Delta_1(G_S)\log|V|)$ bicliques $B_1 = (U_1, V_1; E_1), B_2 = (U_2, V_2; E_2), \ldots, B_q = (U_q, V_q; E_q)$ as in Lemma 52. Note that because of the existence

of v^*, every vertex of G belongs to at least one biclique. We construct a graph G' as follows. Begin with $G' = G$. Next, for every biclique B_i, $i = 1, \ldots, q$ we add

- a $2(k-1)$-cycle $C_i = (v_{i,0}, v_{i,1}, \ldots, v_{i,k-2}, w_{i,k-3}, \ldots, w_{i,1})$,
- an edge $uv_{i,0}$ for every $u \in U_i$,
- an edge $vv_{i,k-2}$ for every $v \in V_i$.

We denote $w_{i,0} = v_{i,0}$ and $w_{i,k-2} = v_{i,k-2}$. For every $i = 1, \ldots, q$, the cycle C_i partitions into two paths $P_i = (v_{i,0}, v_{i,1}, \ldots, v_{i,k-2})$ and $Q_i = (w_{i,0}, v_{i,1}, \ldots, w_{i,k-2})$. Next, we add $2\lceil \log q \rceil$ vertices $a_1, \ldots, a_{\lceil \log q \rceil}$ and $b_1, \ldots,$
$b_{\lceil \log q \rceil}$.

At this point, the construction differs a bit depending on the parity of k.

Assume k is odd. For every $i = 1, \ldots, q$, the middle vertices of the paths P_i and Q_i, i.e., the vertices $v_{i,(k-3)/2}, v_{i,(k-1)/2}, w_{i,(k-3)/2}, w_{i,(k-1)/2}$ are called *portals*. For every $t = 1, \ldots, \lceil \log q \rceil$ we put edges between $\{a_t, b_t\}$ and all portals.

Assume k is even. Then there are two kinds of portals. For every $i = 1, \ldots, q$, the middle vertex of the paths P_i and Q_i, i.e., the vertices $v_{i,(k-2)/2}, w_{i,(k-2)/2}$ are called 1-*portals*. For every $i = 1, \ldots, q$, the neighbors of the 1-portals on P_i and Q_i, i.e., the vertices $v_{i,(k-4)/2}, v_{i,k/2}, w_{i,(k-4)/2}, w_{i,k/2}$ are called 0-*portals*. For every $t = 1, \ldots, \lceil \log q \rceil$ and for every i, we put edges between $\{a_t, b_t\}$ and all $\theta_t(i)$-portals of C_i, where $\theta_t(i)$ is the t-th bit of i.

Finally, form a clique from all vertices a_r and b_r for $r = 1, \ldots, \lceil \log q \rceil$. This completes the construction. Note that we have added $2(k-1)q + 2\lceil \log q \rceil = O(kq) = O(kp^2\Delta_1(G_S) \log |V|)$ vertices and at most $q(2(k-1) + |V| + 8\lceil \log q \rceil) + 2\lceil \log q \rceil^2 = O(q|V|) = O(|V|p^2\Delta_1(G_S) \log |V|)$ edges.

Assume that (G, S) is a YES-instance of Subset Rainbow k-Coloring, and let c be the corresponding coloring. We will show that there is a rainbow k-coloring c' of $E(G')$. Define $c'(e) = c(e)$ for $e \in E$. Next, for every biclique B_i, $i = 1, \ldots, q$ we define colors of the corresponding edges as follows.

- The edges of the cycle $(v_{i,0}, v_{i,1}, \ldots, v_{i,k-2}, w_{i,k-3}, \ldots, w_{i,1})$ are colored with colors $2, 3, \ldots, k-1, 2, \ldots, k-1$ respectively.
- for every $u \in U_i$, we put $c'(uv_{i,0}) = 1$,
- for every $v \in V_i$, we put $c'(vv_{i,k-2}) = k$.

Assume k is odd. For every $t = 1, \ldots, \lceil \log q \rceil$ and for every $i = 1, \ldots, q$ consider the set $A_{t,i}$ (resp. $B_{t,i}$) of four edges between a_t (resp. b_t) and the portals of C_i. If $\theta_t(i) = 0$ the edges of both $A_{t,i}$ and $B_{t,i}$ are colored with $\frac{k+1}{2}$. If $\theta_t(i) = 1$ the edges of $A_{t,i}$ are colored with 1 and the edges of $B_{t,i}$ are colored with k.

Assume k is even. For every $t = 1, \ldots, \lceil \log q \rceil$ and for every $i = 1, \ldots, q$ consider the set $A_{t,i}$ (resp. $B_{t,i}$) of four or two edges between a_t (resp. b_t) and the $\theta_t(i)$-portals of C_i. If $\theta_t(i) = 0$ the edges of both $A_{t,i}$ and $B_{t,i}$ are colored with $\frac{k}{2}$. If $\theta_t(i) = 1$ the edges of $A_{t,i}$ are colored with 1 and the edges of $B_{t,i}$ are colored with k.

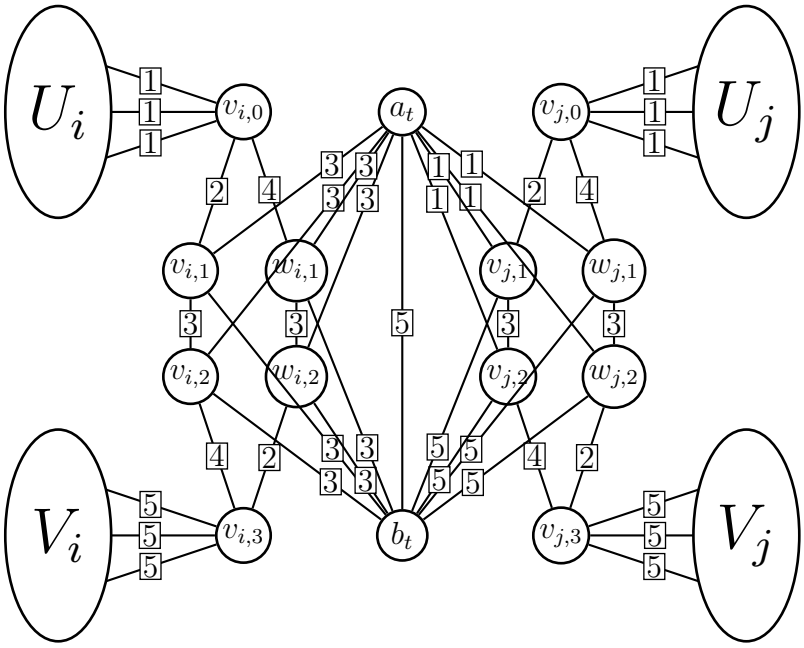

Figure 5.5: Illustration of the $k = 5$ case. Rectangular labels denote colors. Here, t is the number of any bit on which i and j differ; $\theta_t(i) = 0$ and $\theta_t(j) = 1$.

Finally, the clique formed of vertices a_r and b_r is colored in color k. (See Fig 5.5 for an illustration of the $k = 5$ case.) Now we need to verify whether every pair u, v of vertices of (G', c') is connected by a rainbow path. Let us consider cases depending on the types of vertices in the pair.

- If $u, v \in V$ and $\{u, v\} \in S$ then u and v are connected by a rainbow path in (G, c), and we can use the same path.
- If $u, v \in V$ and $\{u, v\} \notin S$ then let B_i be the biclique that contains uv and assume w.l.o.g. $u \in U_i$ and $v \in V_i$. Then the path $(u, v_{i,0}, \ldots, v_{i,k-2}, v)$ is colored by $(1, 2, \ldots, k-1, k)$ and hence is rainbow.
- Assume $u \in V$ and $v \in C_j$ for some $j = 1, \ldots, q$. Pick any biclique B_i that contains u. Assume that $u \in U_i$ (the case $u \in V_i$ is symmetric). If $i = j$ we use the path which begins with $uv_{i,0}$ (colored by 1) and then continues to v using the shortest path on C_j. Since C_j is colored using only colors $2, \ldots, k-1$, this path is rainbow. Hence we can focus on the case $i \neq j$. Let t be any bit on which i and j differ. Assume k is odd. If $v \in \{v_{j,0}, \ldots, v_{j,(k-3)/2}\}$, say $v = v_{j,\ell}$, we use the path $(u, w_{i,0}, \ldots, w_{i,(k-3)/2}, b_t, v_{j,(k-3)/2}, \ldots, v_{j,\ell})$. Depending on whether $\theta_t(i)$ is 0 or 1, the successive edges of this path have colors $(1, k-1, \ldots, \frac{k+3}{2}, \frac{k+1}{2}, k, \frac{k-1}{2}, \ldots, \ell+2)$ or $(1, k-1, \ldots, \frac{k+3}{2}, k, \frac{k+1}{2}, \frac{k-1}{2}, \ldots, \ell+2)$. The remaining three cases $v \in \{v_{j,(k-1)/2}, \ldots, v_{j,k-2}\}$, $v \in \{w_{j,0}, \ldots, w_{j,(k-3)/2}\}$ and $v \in \{w_{j,(k-1)/2}, \ldots, w_{j,k-2}\}$ are analogous. When k is even, $\theta_t(i) = 0$ and $v \in \{v_{j,0}, \ldots, v_{j,(k-2)/2}\}$, say $v = v_{j,\ell}$, we use the path $(u, w_{i,0}, \ldots, w_{i,(k-4)/2}, b_t, v_{j,(k-2)/2}, \ldots, v_{j,\ell})$. Again, there are three

more analogous cases when $\theta_t(i) = 0$ and four ones when $\theta_t(i) = 1$. When $u \in V_i$ the paths are analogous, but we use a_t instead of b_t, to avoid repeating the color k.

- If $u \in V$ and $v = a_r$ or $v = b_r$ for some $r = 1, \ldots, \lceil \log q \rceil$, the requested path is a subpath of one of the rainbow paths described in the previous case.

- Assume $u \in C_i$ and $v \in C_j$ for some $i, j = 1, \ldots, q$. When $i = j$ we can reach v from u by a rainbow path going through the shortest path in C_i. Assume $i \neq j$. Let t be the first bit on which i and j differ. Let us describe the case of odd k only (the case of even k is similar, but slightly different, because of non-symmetric neighborhoods of vertices a_r and b_r). Denote $\text{clone}_j(v_{i,r}) = v_{j,r}$ and $\text{clone}_j(w_{i,r}) = w_{j,r}$. Recall that for every path $v_{i,0}, \ldots, v_{i,k-2}$ there are two portals, similarly for every path $w_{i,0}, \ldots, w_{i,k-2}$. Two portals x_1 and x_2 on the same path are called *twins* and we denote $\text{twin}(x_1) = x_2$ and $\text{twin}(x_2) = x_1$. If $v_{i,r}$ is a portal then $w_{i,r}$ is also a portal and we denote $\text{opposite}(v_{i,r}) = w_{i,r}$ and $\text{opposite}(w_{i,r}) = v_{i,r}$. First assume the shortest path P from u to $\text{clone}_i(v)$ in C_i goes through a portal. Let x be the first portal visited by P from u. Then the rainbow path from u to v is formed by going in P from u to x, then going to a_t, and then either through $\text{twin}(\text{clone}_j(x))$ to v using the shortest path on C_j, when $v \neq \text{clone}_j(x)$, or directly to $\text{clone}_j(x)$, when $v = \text{clone}_j(x)$. The colors are the same as on path P, plus color 1 (and plus color $\frac{k+1}{2}$ when $v = \text{clone}_j(x)$), so the path is rainbow. Now assume that P does not go through a portal. Then the rainbow path from u to v is formed by going in P from u to the nearest portal x, then going through a_t to $\text{clone}_j(\text{opposite}(x))$, and then to v using the shortest path on C_j. This path uses colors of $[k] \setminus c(E(P)) \cup \{1\}$, each exactly once.

- If $u \in C_i$ for some i and $v = a_r$ for some $r = 1, \ldots, \lceil \log q \rceil$, the requested path is a subpath of one of the rainbow paths described in the previous case; when $v = b_r$ we use the path to a_r and extend it by the edge $a_r b_r$.

- If $\{u, v\} \subseteq \{a_r \ : \ r = 1, \ldots, \lceil \log q \rceil\} \cup \{b_r \ : \ r = 1, \ldots, \lceil \log q \rceil\}$, then u and v are adjacent, hence connected by a rainbow path of length 1.

Now assume that G' is a YES-instance of Rainbow k-Coloring, and let c' be the corresponding coloring. We claim that the coloring $c'|_{E(G)}$ of $E(G)$ satisfies all the pairs in S. It follows from the observation that for every $\{u, v\} \in S$ every path between u and v that leaves $E(G)$ either has length at least $k + 1$, or contains two edges $av_{i,0}$ and $v_{i,0}b$ for some $i = 1, \ldots, q$ and $a, b \in U_i$, or contains two edges $av_{i,3}$ and $v_{i,3}b$ for some $i = 1, \ldots, q$ and $a, b \in V_i$. □

5.4 Putting everything together

By pipelining lemmas 16, 46, 47 and 48 we get the following corollary.

Corollary 55. *Fix $k \geq 2$. Given a* 3-CNF-SAT *formula φ with m clauses one can construct in polynomial time an equivalent instance $(G = (V, E), S)$ of* Subset Rainbow k-Coloring *such that $|V| = O(m^{2/3})$, $|E| = O(m)$, $\Delta((V, S)) = O(m^{1/3})$, and the graph $G_S = (V, S)$ is $O(1)$-colorable.*

Note that in Corollary 55 we have $|S| = |V|\Delta((V, S)) = O(m)$. It follows that the Sparsification Lemma (Lemma 14) and Corollary 55 imply Theorem 8.

Pipelining Corollary 55 and Lemma 53 (for $k = 2$) or Lemma 54 (for $k \geq 3$) gives the following corollary.

Corollary 56. *Fix $k \geq 2$. Given a* 3-CNF-SAT *formula φ with $O(m)$ clauses one can construct an equivalent instance G of* Rainbow k-Coloring *with $O(m^{2/3})$ vertices and $O(m \log m)$ edges. The construction algorithm can run in (i) expected polynomial time, or (ii) deterministic $2^{O(m^{2/3})}$ time.*

Again, the above and the Sparsification Lemma immediately imply Theorem 7. (Note that we do not use the randomized reduction algorithm — we state it just in case it is useful in some other applications.)

Chapter 6

Multicoloring and low-degree monomial testing

In this chapter we prove that there is no algorithm solving $(a{:}b)$-COLORING in time $f(b) \cdot 2^{o(\log b)\cdot n}$ for any computable function $f(b)$, unless ETH fails. Additionally we prove that there is no algorithm solving (r,k)-MONOMIAL TESTING in time $2^{o(k\cdot\frac{\log r}{r})}\cdot |C|^{O(1)}$, also unless ETH fails.

Organization of the chapter. In Section 6.1 we recall definitions and well-known facts. We also discuss d-detecting families, the main combinatorial tool used in our reduction. In Section 6.2 we prove the lower bound for the list version of the problem, i.e., Theorem 11, and then in Section 6.3 we present the few steps needed for the standard version, thereby proving Theorem 9. Section 6.4 is devoted to deriving lower bounds for low-degree monomial testing.

Additional notation for this chapter. All graphs we consider in this chapter are simple and undirected. By $\uplus$ we denote the disjoint union, i.e., by $A \uplus B$ we mean $A \cup B$ with the indication that A and B are disjoint.

6.1 Preliminaries

List and nonuniform list $(a{:}b)$-coloring. For integers a, b and a graph G with a function $L\colon V(G) \to 2^{[a]}$ (assigning a list of colors to every vertex), an *L-$(a{:}b)$-coloring* of G is an assignment of exactly b colors from $L(v)$ to each vertex $v \in V(G)$, such that adjacent vertices get disjoint color sets. The LIST $(a{:}b)$-COLORING problem asks, given (G, L), whether an L-$(a{:}b)$-coloring of G exists.

As an intermediary step of our reduction, we use the following generalization of list colorings where the number of demanded colors varies with every vertex. For integers a, b, a graph G with a function $L\colon V(G) \to 2^{[a]}$ and a *demand function* $\beta\colon V(G) \to \{1, \ldots, b\}$, an *$L$-$(a{:}\beta)$-coloring* of G is an assignment of exactly $\beta(v)$ colors from $L(v)$ to each vertex $v \in V(G)$, such that adjacent vertices get disjoint color sets. NONUNIFORM LIST $(a{:}b)$-COLORING is then the problem in which given (G, L, β) we ask if an L-$(a{:}\beta)$-coloring of G exists.

d-detecting families. The crucial ingredient in the proof of Theorem 11 is the usage of *d-detecting matrices* introduced by Lindström [99]. We choose to work with their combinatorial formulation, hence we shall talk about *d-detecting families*. Suppose we are given some universe U and there is an unknown function $f\colon U \to \{0, 1, \ldots, d-1\}$, for some fixed positive integer d. One may think of U as consisting of coins of unknown weights that are integers between 0 and $d-1$. We would like to learn f (the weight of every coin) by asking a small number of queries of the following form: for a subset $X \subseteq U$, what is $\sum_{e\in X} f(e)$ (the total weight of coins in X)? A set of queries sufficient for determining all the values of an arbitrary f is called a *d-detecting family*. Of course f can be learned by asking $|U|$ questions about single coins, but it turns out that significantly fewer questions are needed: there is a d-detecting family of size $O(|U|/\log|U|)$, for every fixed d [99]. The logarithmic factor in the denominator will be crucial for deriving our lower bound. Let us state this notion in a formal way.

Definition 57. *A* d-detecting family *for a finite set U is a family $\mathcal{F} \subseteq 2^U$ of subsets of U such that for every two functions $f, g : U \to \{0, \ldots, d-1\}$, $f \neq g$, there is a set S in the family such that $\sum_{x\in S} f(x) \neq \sum_{x\in S} g(x)$.*

A deterministic construction of sublinear, d-detecting families was given by Lindström [99], together with a proof that even the constant factor 2 in the family size cannot be improved.

Theorem 58 ([99])**.** *For every constant $d \in \mathbb{N}$ and finite set U, there is a d-detecting family $\mathcal{F}$ on U of size $\frac{2|U|}{\log_d |U|} \cdot (1 + o(1))$. Furthermore, $\mathcal{F}$ can be constructed in $|U|^{O(1)}$ time.*

Other constructions, generalizations, and discussion of similar results can be found in Grebinski and Kucherov [65], and in Bshouty [16]. Note that the expression $\sum_{x\in S} f(x)$ is just the product of f as a vector in $[d]^{|U|}$ with the characteristic vector of S. Hence, instead of subset families, Lindström speaks of *detecting vectors*, while later works see them as *detecting matrices*, that is, $(0, 1)$-matrices with these vectors as rows (which define an injection on $[d]^{|U|}$ despite having few rows). Similar definitions appear in the study of query complexity, e.g., as in the popular Mastermind game [32].

While known polynomial deterministic constructions of detecting families involve some number theory or Fourier analysis, their existence can be argued with an elementary probabilistic argument. Intuitively, a random subset $S \subseteq U$ will distinguish two distinct functions $f, g : U \to \{0, \ldots, d-1\}$ (meaning $\sum_{x\in S} f(x) \neq \sum_{x\in S} g(x)$) with probability at least $\frac{1}{2}$. This is because any x where f and g disagree is taken or not taken into S with probability $\frac{1}{2}$, while sums over S cannot agree in both cases simultaneously, as they differ by $f(x)$ and $g(x)$ respectively. There are $d^n \cdot d^n$ function pairs to be distinguished. In any subset of pairs, at least half are distinguished by a random set in expectation, thus at least one such set exists. Repeatedly finding such a set for undistinguished pairs, we get $|\log_{\frac{1}{2}}(d^n \cdot d^n)| = O(n \log d)$ sets that distinguish all functions. More strongly though, when two functions differ on more

values, the probability of distinguishing them increases significantly. Hence we need fewer random sets to distinguish all pairs of distant functions. On the other hand, there are few function pairs that are close, so we need few random sets to distinguish them all as well. This allows to show that in fact $O(\frac{n}{\log_d n})$ random sets are enough to form a d-detecting family with positive probability [65].

Let us now sketch how d-detecting families are used in the proof of Theorem 11. Given an instance φ of 3-CNF-SAT with n variables and $O(n)$ clauses, and a number $b \leq n/\log n$, we will construct an instance G of List $(a{:}b)$-coloring for some a. This instance will have a positive answer if and only if φ is satisfiable, and the constructed graph G will have $O(n/\log b)$ vertices. It can be easily seen that this will yield the promised lower bound.

Partition the clause set C of φ into groups $C_1, C_2, \ldots, C_p$, each of size roughly b; thus $p = O(n/b)$. Similarly, partition the variable set V of φ into groups $V_1, \ldots, V_q$, each of size roughly $\log_2 b$; thus $q = O(n/\log b)$. In the output instance we create one vertex per each variable group—hence we have $O(n/\log b)$ such vertices—and one block of vertices per each clause group, whose size will be determined in a moment. Our construction ensures that the set of colors assigned to a vertex created for a variable group misses one color from some subset of b colors. The choice of the missing color corresponds to one of $2^{\log_2 b} = b$ possible boolean assignments to the variables of the group.

Take any vertex u from a block of vertices created for some clause group C_j. We make it adjacent to vertices constructed for precisely those variable groups V_i, for which there is some variable in V_i that occurs in some clause of C_j. This way, u can only take a subset of the above missing colors corresponding to the chosen assignment on variables relevant to C_j. By carefully selecting the list of u, and some additional technical gadgeteering, we can express a constraint of the following form: the total number of satisfied literals in some subset of clauses of C_j is exactly some number. Thus, we could verify that every clause of C_j is satisfied by creating a block of $|C_j|$ vertices, each checking one clause. However, the whole graph output by the reduction would then have $O(n)$ vertices, and we would not obtain any non-trivial lower bound. Instead, we create one vertex per each question in a d-detecting family on the universe $U = C_j$, which has size $O(|C_j|/\log|C_j|) = O(|C_j|/\log b)$. Then, the total number of vertices in the constructed graph will be $O(n/\log b)$, as intended.

6.2 Hardness of List $(a{:}b)$-coloring

In this section we show our main technical contribution: an ETH-based lower bound for List $(a{:}b)$-coloring. We begin with the key part: reducing an n-variable instance 3-CNF-SAT to an instance of Nonuniform List $(a{:}b)$-coloring with only $O(\frac{n}{\log b})$ vertices. Next, it is rather easy to reduce Nonuniform List $(a{:}b)$-coloring to List $(a{:}b)$-coloring.

6.2.1 The nonuniform case

We prove the following theorem through the remaining part of this section.

Theorem 59. *For any instance ϕ of* (3,4)-CNF-SAT *with n variables and any integer $2 \leq b \leq n/\log_2 n$, there is an equivalent instance (G, β, L) of* Nonuniform List $(a{:}2b)$-coloring *such that $a = O(b^2 \log b)$, $|V(G)| = O(\frac{n}{\log b})$ and G is 3-colorable. Moreover, the instance (G, β, L) and the 3-coloring of G can be constructed in $n^{O(1)}$ time.*

Consider an instance ϕ of 3-CNF-SAT where each variable appears in at most four clauses. Let V be the set of its variables and C be the set of its clauses. Note that $\frac{1}{3}|V| \leq |C| \leq \frac{4}{3}|V|$. Let $a = 12b^2 \cdot \lfloor \log_2 b \rfloor$. We shall construct, for some integers $n_V = O(|V|/\log b)$ and $n_C = O(|C|/b)$:

- a partition $V = V_1 \uplus \ldots \uplus V_{n_V}$ of variables into groups of size at most $\lfloor \log_2 b \rfloor$,
- a partition $C = C_1 \uplus \ldots \uplus C_{n_C}$ of clauses into groups of size at most b,
- a function $\sigma \colon \{1, \ldots, n_V\} \to [12 \cdot b \cdot \lfloor \log_2 b \rfloor]$,

such that the following condition holds:

> For any $j = 1, \ldots, n_C$, the variables occurring in clauses of C_j are all different and they all belong to pairwise different variable groups. Moreover, the indices of these groups are mapped to pairwise different values by σ. (✠)

In other words, any two literals of clauses in C_j have different variables, and if they belong to V_i and $V_{i'}$ respectively, then $\sigma(i) \neq \sigma(i')$.

Lemma 60. *Partitions $V = V_1 \uplus \ldots \uplus V_{n_V}$, $C = C_1 \uplus \ldots \uplus C_{n_C}$ and a function σ satisfying (✠) can be found in time $O(n)$.*

Proof. We first group variables, in a way such that the following holds: (P1) the variables occurring in any clause are different and belong to different variable groups. To this end, consider the graph G_1 with variables as vertices and edges between any two variables that occur in a common clause (i.e. the primal graph of ϕ). Since no clause contains repeated variables, G_1 has no loops. Since every variable of ϕ occurs in at most four clauses, and since those clauses contain at most two other variables, the maximum degrees of G_1 is at most 8. Hence G_1 can be greedily colored with 9 colors. Then, we refine the partition given by colors to make every group have size at most $\lfloor \log_2 b \rfloor$, producing in total at most $n_V := \lceil |V|/\lfloor \log_2 b \rfloor \rceil + 9$ groups $V_1, \ldots, V_{n_V}$. (P1) holds, because any two variables occurring in a common clause are adjacent in G_1, and thus get different colors, and thus are assigned to different groups.

Next, we group clauses in a way such that: (P2) the variables occurring in clauses of a group C_j are all different and belong to different variable groups. For this, consider the graph G_2 with clauses as vertices, and with an edge between clauses if they contain two different variables from the same variable group. By (P1), G_2

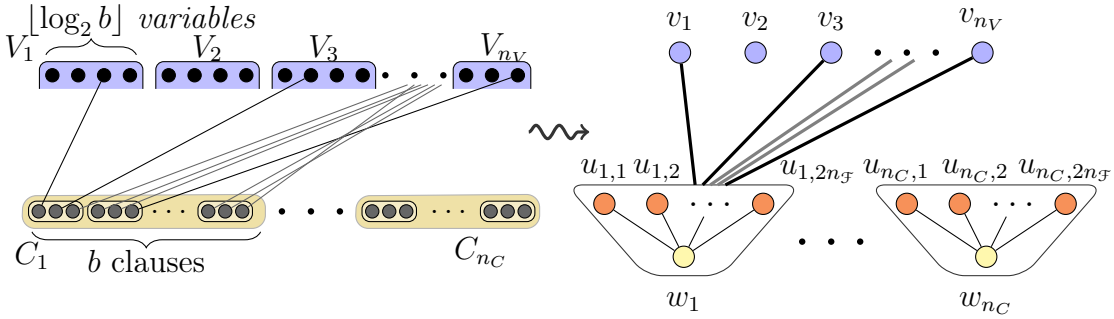

Figure 6.1: *(left)* The groups of variables and clauses of the formula; literals in C_1 are joined with their variables. Since no variable of V_2 occurs in C_1, we have $2 \notin I_1$ – this may allow us to make $\sigma(2)$ the same number as $\sigma(3)$, say, reducing the total number a of colors needed. *(right)* The constructed graph; thick lines represent edges to all vertices corresponding to C_1.

has no loops. Since every clause contains exactly 3 variables, each variable is in a group with at most $\lfloor \log_2 b \rfloor - 1$ others, and every such variable occurs in at most 4 clauses, the maximum degree of G_2 is at most $12(\lfloor \log_2 b \rfloor - 1)$. We can therefore color G_2 greedily with $12\lfloor \log_2 b \rfloor$ colors. Similarly as before, we partition clauses into $n_C := \lceil |C|/b \rceil + 12\lfloor \log_2 b \rfloor$ monochromatic groups $C_1, \ldots, C_{n_C}$ of size at most b each. Then (P2) holds by construction of the coloring.

Finally, consider a graph G_3 with variable groups as vertices, and with an edge between two variable groups if they contain two different variables occurring in clauses from a common clause group. More precisely, V_i and $V_{i'}$ are adjacent if there are two different variables $x \in V_i$ and $x' \in V_{i'}$, and a clause group C_j with clauses c and c' (possibly $c = c'$), such that x occurs in c and x' occurs in c'. By (P2), G_3 has no loops. Since a variable has at most $\lfloor \log_2 b \rfloor - 1$ other variables in its group, each of these variables occur in at most 4 clauses, each of these clauses has at most $b - 1$ other clauses in its group, and each of these contains exactly 3 variables, the maximum degree of G_3 is at most $4 \cdot (\lfloor \log_2 b \rfloor - 1) \cdot (b-1) \cdot 3$. We can therefore color it greedily into $12b\lfloor \log_2 b \rfloor$ colors. Let σ be the resulting coloring. By (P2) and the construction of this coloring, (✠) holds.

The colorings can be found in linear time using standard techniques. Note that we have $n_V = \lceil |V|/\lfloor \log_2 b \rfloor \rceil + 9 = O(|V|/\log b)$. Moreover, since $b \leq n/\log_2 n$, we get $\log_2 b \leq \log_2 n \leq \frac{n}{b} = \Theta(|C|/b)$ and hence $n_C = \lceil |C|/b \rceil + 12\lfloor \log_2 b \rfloor = O(|C|/b)$. □

For every $1 \leq i \leq n_V$, the set V_i of variables admits $2^{|V_i|} \leq b$ different assignments. We will therefore say that each assignment on V_i is *given* by an integer $x \in [b]$, for example by interpreting the first $|V_i|$ bits of the binary representation of x as truth values for variables in V_i. Note that when $|V_i| < \log_2 b$, different integers from $[b]$ may give the same assignment on V_i.

For $1 \leq j \leq n_C$, let $I_j \subseteq \{1, \ldots, n_V\}$ be the set of indices of variable groups that contain some variable occurring in the clauses of C_j. Since every clause contains exactly three literals, property (✠) means that $|I_j| = 3|C_j|$ and that σ is injective over each I_j. See Fig. 6.1.

For $1 \leq j \leq n_C$, let $\{C_{j,1}, \ldots, C_{j,n_\mathcal{F}}\}$ be a 4-detecting family of subsets of C_j, for some $n_\mathcal{F} = O(\frac{b}{\log b})$ (we can assume $n_\mathcal{F}$ does not depend on j by adding arbitrary sets

when $|C_j| < b$). For every $1 \leq k \leq n_{\mathcal{F}}$, let $C_{j,n_{\mathcal{F}}+k} = C_j \setminus C_{j,k}$.

We are now ready to build the graph G, the demand function $\beta : V(G) \to \{1, \ldots, 2b\}$, and the list assignment L as follows.

(1) For $1 \leq i \leq n_V$, create a vertex v_i with $\beta(v_i) = b - 1$ and $L(v_i) = \{b \cdot \sigma(i) + x \mid x \in [b]\}$.

(2) For $1 \leq j \leq n_C$ and $1 \leq k \leq 2n_{\mathcal{F}}$, create a vertex $u_{j,k}$ adjacent to each v_i for $i \in I_j$.
Let $\beta(u_{j,k}) = |C_{j,k}|$ and

$$L(u_{j,k}) = \{b \cdot \sigma(i) + x \mid 1 \leq i \leq n_V, x \in [2^{|V_i|}] \text{ such that } x \text{ gives an assignment of } V_i \text{ that satisfies some clause of } C_{j,k}\}.$$

(3) For $1 \leq j \leq n_C$, create a vertex w_j, adjacent to each v_i for $i \in I_j$ and to each $u_{j,k}$ ($1 \leq k \leq 2n_{\mathcal{F}}$). Let $\beta(w_j) = 2|C_j|$ and $L(w_j) = \bigcup_{i \in I_j} \{b \cdot \sigma(i) + x \mid x \in [b]\}$.

Before giving a detailed proof of the correctness, let us describe the reduction in intuitive terms. Note that vertices of type v_i get all but one color from their list; this missing color, say $b \cdot \sigma(i) + x_i$, for some $x_i \in [b]$, defines an assignment on V_i. For every $j = 1, \ldots, n_C$ the goal of the gadget consisting of w_j and vertices $u_{j,k}$ is to express the constraint that every clause in C_j has a literal satisfied by this assignment. Since $w_j, u_{j,k}$ are adjacent to all vertices in $\{v_i \mid i \in I_j\}$, they may only use the missing colors (of the form $b \cdot \sigma(i) + x_i$, where $i \in I_j$). Since $|I_j| = 3|C_j|$, there are $3|C_j|$ such colors and $2|C_j|$ of them go to w_j. This leaves exactly $|C_j|$ colors for vertices of type $u_{j,k}$, corresponding to a choice of $|C_j|$ satisfied literals from the $3|C_j|$ literals in clauses of C_j. The lists and demands for vertices $u_{j,k}$ guarantee that exactly $|C_{j,k}|$ chosen satisfied literals occur in clauses of $C_{j,k}$. The properties of 4-detecting families will ensure that every clause has exactly one chosen, satisfied literal, and hence at least one satisfied literal. We proceed with formal proofs.

Lemma 61. *If ϕ is satisfiable then G is L-$(a{:}\beta)$-colorable.*

Proof. Consider a satisfying assignment η for ϕ. For $1 \leq i \leq n_V$, let $x_i \in [2^{|V_i|}]$ be an integer giving the same assignment on V_i as η. For every clause c of ϕ, choose one literal satisfied by η in it, and let i_c be index of the group V_{i_c} containing the literal's variable. Let $\alpha : V(G) \to \binom{[a]}{\leq 2b}$ be the L-$(a{:}\beta)$-coloring of G defined as follows, for $1 \leq i \leq n_V$, $1 \leq j \leq n_C$, $1 \leq k \leq 2n_{\mathcal{F}}$:

- $\alpha(v_i) = L(v_i) \setminus \{b \cdot \sigma(i) + x_i\}$
- $\alpha(u_{j,k}) = \{b \cdot \sigma(i_c) + x_{i_c} \mid c \in C_{j,k}\}$
- $\alpha(w_j) = \{b \cdot \sigma(i) + x_i \mid i \in I_j \setminus \{i_c \mid c \in C_j\}\}$.

Let us first check that every vertex v gets colors from its list $L(v)$ only. This is immediate for vertices v_i and w_j, while for $u_{j,k}$ it follows from the fact that x_{i_c} gives a partial assignment to V_i that satisfies some clause of $C_{j,k}$.

Now let us check that for every vertex v, the coloring α assigns exactly $\beta(v)$ colors to v. For $\alpha(v_i)$ this follows from the fact that $|L(v_i)| = b$ and $0 \leq x_i < 2^{|V_i|} \leq b$. Since by property (✠), σ is injective on I_j, and thus on $\{i_c \mid c \in C_{j,k}\} \subseteq I_j$, we have $|\alpha(u_{j,k})| = |C_{j,k}| = b(u_{j,k})$. Similarly, since σ is injective on I_j and $|I_j \setminus \{i_c \mid c \in C_j\}| = 3|C_j| - |C_j| = 2|C_j|$, we get $|\alpha(w_j)| = 2|C_j| = \beta(w_j)$.

It remains to argue that the sets assigned to any two adjacent vertices are disjoint. There are three types of edges in the graph, namely $v_i u_{j,k}$, $v_i w_j$, and $w_j u_{j,k}$. The disjointness of $\alpha(w_j)$ and $\alpha(u_{j,k})$ is immediate from the definition of α, since $C_{j,k} \subseteq C_j$. Fix $j = 1, \ldots, n_C$. Since σ is injective on I_j, for any two different $i, i' \in I_j$, we have $b \cdot \sigma(i) + x_i \notin L(v_{i'})$. Hence,

$$\bigcup_{i \in I_j} \alpha(v_i) = \{b \cdot \sigma(i) + x \mid i \in I_j \text{ and } x \in [b]\} \setminus \{b \cdot \sigma(i) + x_i \mid i \in I_j\}.$$

Since $\alpha(u_{j,k}), \alpha(w_j) \subseteq \{b \cdot \sigma(i) + x_i \mid i \in I_j\}$, it follows that edges of types $v_i u_{j,k}$ and $v_i w_j$ received disjoint sets of colors on their endpoints, concluding the proof. □

Lemma 62. *If G is L-$(a{:}\beta)$-colorable then ϕ is satisfiable.*

Proof. Assume that G is L-$(a{:}\beta)$-colorable, and let α be the corresponding coloring.

For $1 \leq i \leq n_V$, we have $|L(v_i)| = b$ and $|\alpha(v_i)| = b - 1$, so v_i misses exactly one color from its list. Let $b \cdot \sigma(i) + x_i$, for some $x_i \in [b]$, be the missing color. We want to argue that the assignment x for ϕ given by x_i on each V_i satisfies ϕ.

Consider any clause group C_j, for $1 \leq j \leq n_C$. Every vertex in $\{w_j\} \cup \{u_{j,k} \mid 1 \leq k \leq 2n_{\mathcal{F}}\}$ contains $\{v_i \mid i \in I_j\}$ in its neighborhood. Therefore, the sets $\alpha(u_{j,k})$ and $\alpha(w_j)$ are disjoint from $\bigcup_{i \in I_j} \alpha(v_i)$. Since $L(u_{j,k}), L(w_j) \subseteq \{b \cdot \sigma(i) + x' \mid i \in I_j, x' \in [b]\}$, we get that $\alpha(u_{j,k})$ and $\alpha(w_j)$ are contained in the set of missing colors $\{b \cdot \sigma(i) + x_i \mid i \in I_j\}$ (corresponding to the chosen assignment). By property (✠), this set has exactly $|I_j| = 3|C_j|$ different colors. Of these, exactly $2|C_j|$ are contained in $\alpha(w_j)$. Let the remaining $|C_j|$ colors be $\{b \cdot \sigma(i) + x_i \mid i \in J_j\}$, for some subset $J_j \subseteq I_j$ of $|C_j|$ indices.

Since $\alpha(u_{j,k})$ is disjoint from $\alpha(w_j)$, we have $\alpha(u_{j,k}) \subseteq \{b \cdot \sigma(i) + x_i \mid i \in J_j\}$ for all k. By definition of I_j, for every $i \in J_j \subseteq I_j$ there is a variable in V_i that appears in some clause of C_j. By property (✠), it can only occur in one such clause, so let l_i be the literal in the clause of C_j where it appears. For every color $b \cdot \sigma(i) + x_i \in \alpha(u_{j,k})$, by definition of the lists for $u_{j,k}$ we know that x_i gives a partial assignment to V_i that satisfies some clause of $C_{j,k}$. This means x_i makes the literal l_i true and l_i occurs in a clause of $C_{j,k}$. Therefore, for each k, at least $|\alpha(u_{j,k})| = |C_{j,k}|$ literals from the set $\{l_i \mid i \in J_j\}$ occur in clauses of $C_{j,k}$ and are made true by the assignment x.

Let $f : C_j \to \{0, 1, 2, 3\}$ be the function assigning to each clause $c \in C_j$ the number of literals of c in $\{l_i \mid i \in J_j\}$. By the above, $\sum_{c \in C_{j,k}} f(c) \geq |C_{j,k}|$ for $1 \leq k \leq 2n_{\mathcal{F}}$. Since each literal in $\{l_i \mid i \in J_j\}$ belongs to some clause of C_j, we have $\sum_{c \in C_j} f(c) = |J_j| = |C_j|$. Then,

$$\sum_{c \in C_{j,k}} f(c) = \sum_{c \in C_j} f(c) - \sum_{c \in C_{j,n_{\mathcal{F}}+k}} f(c) \leq |C_j| - |C_{j,n_{\mathcal{F}}+k}| = |C_{j,k}|.$$

Hence $\sum_{c \in C_{j,k}} f(c) = |C_{j,k}|$ for $1 \leq k \leq 2n_{\mathcal{F}}$. Let $g : C_j \to \{0,1,2,3\}$ be the constant function $g \equiv 1$. Note that

$$\sum_{c \in C_{j,k}} g(c) = |C_{j,k}| = \sum_{c \in C_{j,k}} f(c).$$

Since $\{C_{j,1}, \ldots, C_{j,n_{\mathcal{F}}}\}$ is a 4-detecting family, this implies that $f \equiv 1$. Thus, for every clause c of C_j we have $f(c) = 1$, meaning that there is a literal from the set $\{l_i \mid i \in J_j\}$ in this clause. All these literals are made positive by the assignment η, therefore all clauses of C_j are satisfied. Since $j = 1, \ldots, n_C$ was arbitrary, this concludes the proof that η is a satisfying assignment for ϕ. □

The construction can clearly be made in polynomial time and the total number of vertices is $n_V + n_C \cdot O(\frac{b}{\log b}) + n_C = O(\frac{n}{\log b})$. Moreover, we get a proper 3-coloring of G, by coloring vertices of the type v_i by color 1, vertices of the type $u_{j,k}$ by color 2, and vertices of the type w_j by color 3. By Lemmas 61 and 62, this concludes the proof of Theorem 59.

6.2.2 The uniform case

In this section we reduce the nonuniform case to the uniform one, and state the resulting lower bound on the complexity of List $(a{:}b)$-coloring.

Lemma 63. *For any instance $I = (G, \beta, L)$ of* Nonuniform List $(a{:}b)$-coloring *where the graph G is t-colorable, there is an equivalent instance (G, L') of* List $((a+tb){:}b)$-coloring. *Moreover, given a t-coloring of G the instance (G, L') can be constructed in time polynomial in $|I| + b$.*

Proof. Let $c\colon V(G) \to [t]$ be a t-coloring of G. For every vertex v, define a set of filling colors $F(v) = \{a + c(v)b + i \,:\, i = 0, \ldots, b - |\beta(v)| - 1\}$ and put $L'(v) = L(v) \cup F(v)$.

Let $\alpha\colon V(G) \to 2^{[a]}$ be an L-$(a{:}\beta)$-coloring of G. We define a coloring $\alpha'\colon V(G) \to 2^{[a+tb]}$ by setting $\alpha'(v) = \alpha(v) \cup F(v)$ for every vertex $v \in V(G)$. Observe that $\alpha'(v) \subseteq L'(v)$ and $|\alpha'(v)| = |\alpha(v)| + (b - |\beta(v)|) = b$. Since α was a proper L-$(a{:}\beta)$-coloring, adjacent vertices can only share the filling colors. However, the lists of adjacent vertices have disjoint subsets of filling colors, since these vertices are colored differently by c. It follows that α' is an L'-$(a{:}b)$-coloring of G.

Conversely, let $\alpha'\colon V(G) \to 2^{[a+tb]}$ be an L'-$(a{:}b)$-coloring of G. For every vertex v, we have $|\alpha'(v) \cap [a]| = b - |\alpha'(v) \cap F(v)| \geq b - (b - |\beta(v)|) = |\beta(v)|$. Define $\alpha(v)$ to be any cardinality $\beta(v)$ subset of $\alpha'(v) \cap [a]$. It is immediate that α is an L-$(a{:}\beta)$-coloring of G. □

We are now ready to prove one of main results of this chapter.

Theorem 11. *If there is an algorithm for* List $(a{:}b)$-coloring *that runs in time $2^{o(\log b)\cdot n}$, then ETH fails. This holds even if the algorithm is only required to work on instances where $a = \Theta(b^2 \log b)$ and $b = \Theta(b(n))$ for an arbitrarily chosen polynomial-time computable function $b(n)$ such that $b(n) \in \omega(1)$ and $b(n) = O(n/\log n)$.*

Proof. Let $b(n)$ be a function as in the statement. We can assume w.l.o.g. that $2 \leq b(n) \leq n/\log_2 n$ (otherwise, replace $b(n)$ with $b'(n) = 2 + \lfloor b(n)/c \rfloor$ in the reasoning below, for c a big enough constant; clearly $b'(n) = \Theta(b(n))$). Fix a function $f(b) = o(\log b)$ and assume there is an algorithm $\mathcal{A}$ for List $(a{:}b)$-coloring that runs in time $2^{f(b)\cdot n}$, whenever $b = \Theta(b(n))$. Consider an instance of (3,4)-CNF-SAT with n variables. Let $b = b(n)$. By Theorem 59 in $n^{O(1)}$ time we get an equivalent instance (G, β, L) of Nonuniform List $(a{:}(2b))$-coloring such that $a = \Theta(b^2 \log b)$, $|V(G)| = O(\frac{n}{\log b})$, and a 3-coloring of G. Next, by Lemma 63 in $n^{O(1)}$ time we get an equivalent instance (G, L') of List $((a + 6b){:}(2b))$-coloring. Finally, we solve the instance (G, L') using algorithm $\mathcal{A}$. Since $b(n) = \omega(1)$, we have $f(b(n)) = o(\log(b(n)))$, and $\mathcal{A}$ runs in time $2^{o(\log b(n))\cdot|V(G)|}$. Thus, we solved (3,4)-CNF-SAT in time $2^{o(\log b(n))\cdot|V(G)|} = 2^{o(\log b(n))\cdot\frac{n}{\log b(n)}} = 2^{o(n)}$. By Corollary 17, this contradicts ETH. □

6.3 From List $(a{:}b)$-coloring to $(a{:}b)$-coloring

Finally, we reduce List $(a{:}b)$-coloring to $(a{:}b)$-coloring. This is done by increasing the number of colors by b, adding a Kneser graph $KG_{a+b,b}$ (which can be colored essentially only by assigning each b-set of colors to its corresponding vertex), and replacing the lists by edges to appropriate vertices of the Kneser graph.

We will need the following well-known property of Kneser graphs (see e.g., Theorem 7.9.1 in the textbook [63]).

Theorem 64. *If $p > 2q$ then every homomorphism from $KG_{p,q}$ to $KG_{p,q}$ is an automorphism.*

We proceed with the reduction.

Lemma 65. *Given an instance of* List $(a{:}b)$-coloring *with n vertices, an equivalent instance of $(a + b : b)$-*coloring *with $n + \binom{a+b}{b}$ vertices can be computed in $\max\{n, \binom{a+b}{b}\}^{O(1)}$-time.*

Proof. Let (G, L) be an instance of List $(a{:}b)$-coloring where G is a graph and $L\colon V(G) \to 2^{[a]}$ describes the lists of allowed colors. Define a graph K with $V(K) = \binom{[a+b]}{b}$ and

$$E(K) = \{\, XY \;:\; X, Y \in V(K) \text{ and } X \cap Y = \emptyset \,\}.$$

That is, K is isomorphic to the Kneser graph $KG_{a+b,b}$. Then let $V' = V(G) \uplus V(K)$ and

$$E' = E(G) \uplus E(K) \uplus \{\, vX \;:\; v \in V(G) \text{ and } X \in V(K) \text{ and } L(v) \cap X = \emptyset \,\}.$$

The graph $G' = (V', E')$ has $n + \binom{a+b}{b}$ vertices, and the construction can be done in time polynomial in $n + \binom{a+b}{b}$. Let G' be our output instance of $((a+b){:}b)$-coloring. We will show that it is equivalent to the instance (G, L) of List $(a{:}b)$-coloring.

Let us assume that $\alpha : V(G) \to \binom{[a]}{b}$ is an L-$(a{:}b)$-coloring of G. Consider $\alpha' : V(G') \to \binom{[a+b]}{b}$ such that

$$\alpha'(v) = \begin{cases} \alpha(v) & \text{for } v \in V(G) \\ v & \text{for } v \in V(K) = \binom{[a+b]}{b}. \end{cases}$$

We claim that α' is an $((a+b){:}b)$-coloring of G'. Indeed, for every edge $uv \in E(G)$ we have $\alpha'(u) \cap \alpha'(v) = \alpha(u) \cap \alpha(v) = \emptyset$ because α is an L-$(a{:}b)$-coloring of G. For every edge $XY \in E(K)$ we have $\alpha'(X) \cap \alpha'(Y) = X \cap Y = \emptyset$. For every edge $vX \in E(V(G), V(K))$ we have $\alpha'(v) \cap \alpha'(X) = \alpha(v) \cap X \subseteq L(v) \cap X = \emptyset$.

Now, let us assume that $\alpha' : V(G') \to \binom{[a+b]}{b}$ is an $((a+b){:}b)$-coloring of G'. Recall that α' is a homomorphism of G' to $KG_{a+b,b}$. Denote $\phi = \alpha'|_{V(K)}$. By Theorem 64, ϕ is an automorphism of K. Define $\alpha'' = \phi^{-1} \circ \alpha'$. Then α'' is an $((a+b){:}b)$-coloring of G' with the property that $\alpha''(X) = X$ for every $X \in V(K)$. We claim that $\alpha''|_{V(G)}$ is an L-$(a{:}b)$-coloring of G. Since α'' is a $((a+b){:}b)$-coloring of G', it suffices to show that $\alpha''(v) \subseteq L(v)$ for every vertex $v \in V(G)$. Pick a color $\gamma \notin L(v)$. Let X_γ be the b-element set consisting of γ and arbitrary $b-1$ elements from $[a+b] \setminus ([a] \cup \{\gamma\})$. Then $L(v) \cap X_\gamma = \emptyset$ and hence $vX_\gamma \in E(G')$. It follows that $X_\gamma \cap \alpha''(v) = \alpha''(X_\gamma) \cap \alpha''(v) = \emptyset$, and in particular $\gamma \notin \alpha''(v)$. Thus, $\alpha''(v) \subseteq L(v)$ as required. □

We now prove our main result.

Theorem 9. *If there is an algorithm for $(a{:}b)$-*COLORING *that runs in time $f(b) \cdot 2^{o(\log b)\cdot n}$, for some computable function $f(b)$, then ETH fails. This holds even if the algorithm is only required to work on instances where $a = \Theta(b^2 \log b)$.*

Proof. Fix a computable function $f(b)$, a function $g(b) = o(\log b)$ and assume there is an algorithm $\mathcal{A}$ for $(a{:}b)$-COLORING that runs in time $f(b) \cdot 2^{g(b)\cdot n}$ for a given n-vertex graph, whenever $a = \Theta(b^2 \log b)$. Without loss of generality we can replace $f(b)$ by any non-decreasing function $f'(n)$ such that $f'(n) \geq f(n)$ and $f'(n) > n$. Intuitively, we now define an unbounded function $b(N)$ which should be at least 2, at most the inverse of f, and small enough so that $2^{O(b \log b)} \leq \frac{N}{\log b}$. The following function is $\omega(1)$ and a standard argument shows how to compute it in $N^{O(1)}$ time (see Lemmas 3.2 and 3.4 in [18]).

$$b(N) = \min\left(\max\{b : f(b) \leq N\}\ ,\ \max\{b : b \log b \leq \log N / \log\log N\}\right) + 2.$$

Consider an instance of (3,4)-CNF-SAT with N variables. Let $b = b(N)$. By Theorem 59 in $N^{O(1)}$ time we get an equivalent instance (G, β, L) of NONUNIFORM LIST $(a{:}(2b))$-COLORING such that $a = \Theta(b^2 \log b)$, $|V(G)| = O(\frac{N}{\log b})$, and a 3-coloring of G. Next, by Lemma 63 in $N^{O(1)}$ time we get an equivalent instance (G, L') of LIST $((a+6b){:}(2b))$-COLORING. Then, by Lemma 65, in time $\max\{N, \binom{a+8b}{2b}\}^{O(1)} = N^{O(1)}$ we get an equivalent instance G' of $((a+8b){:}(2b))$-COLORING such that $|V(G')| = |V(G)| + \binom{a+8b}{2b}$. Observe that since $a = \Theta(b^2 \log b)$ and $b \log b \leq \log N / \log\log N$,

$$\binom{a+8b}{2b} \leq (a+8b)^{2b} = 2^{O(b \log b)} = 2^{O(\log N / \log\log N)} = N^{o(1)} = o(N / \log b(N))$$

Hence $|V(G')| = O(\frac{N}{\log b})$. Finally, we solve the instance G' using algorithm $\mathcal{A}$. Since $b(N) = \omega(1)$, we have $g(b(N)) = o(\log(b(N)))$. Therefore, $\mathcal{A}$ runs in time

$$f(b) \cdot 2^{o(\log b(N)) \cdot |V(G')|} \leq N \cdot 2^{o(\log b(N)) \cdot O(N/\log b(N))} = 2^{o(N)}$$

solving the instance ϕ of (3,4)-CNF-SAT in time $2^{o(N)}$. By Corollary 17, this contradicts ETH.

□

Corollary 10. *If there is an algorithm for* Graph Homomorphism *that runs in time $f(h) \cdot 2^{o(\log\log h)\cdot n}$, for some computable $f(h)$, then ETH fails. This holds even if the algorithm is only required to work on instances where H is a Kneser graph $KG_{a,b}$ with $a = \Theta(b^2 \log b)$.*

Proof. Fix a computable function $f(h)$ and assume there is an algorithm $\mathcal{A}$ for Graph Homomorphism that runs in time $f(h) \cdot 2^{o(\log\log h)\cdot n}$ for a given n-vertex graph, whenever H is a Kneser graph $K_{a,b}$ with $a = \Theta(b^2 \log b)$. Consider an instance of $(a{:}b)$-coloring with n vertices and $a = \Theta(b^2 \log b)$. This is an instance of Graph Homomorphism with $h = \binom{a}{b} \leq a^b = 2^{O(b \log b)}$, hence $\mathcal{A}$ solves it in

$$f(h) \cdot 2^{o(\log\log h)\cdot n} = f(2^{O(b\log b)}) \cdot 2^{o(\log(b\log b))\cdot n} \leq f'(b) \cdot 2^{o(\log b)\cdot n}$$

for some computable function $f'(b) \geq f(2^{\Theta(b\log b)})$, which contradicts Theorem 9. □

6.4 Low-degree testing

In this section we derive lower bounds for (r,k)-Monomial Testing. In this problem, we are given an arithmetic circuit C over some field $\mathbb{F}$ (with input, constant, addition, and multiplication gates). One gate is designated to be the output gate, and it computes some polynomial P of the variables $x_1, x_2, \ldots, x_n$ that appear in the input gates. We assume that P is a homogeneous polynomial of degree k, i.e., all its monomials have total degree k. The task is to verify whether P contains an r-monomial, i.e., a monomial in which every variable has its individual degree bounded by r, for a given $r \leq k$. Abasi et al. [2] gave a very fast randomized algorithm for (r,k)-Monomial Testing.

Theorem 66 (Abasi et al. [2])**.** *Fix integers r, k with $2 \leq r \leq k$. Let $p \leq 2r^2 + 2r$ be a prime, and let $g \in \mathrm{GF}(p)[x_1, \ldots, x_n]$ be a homogeneous polynomial of degree k, computable by a circuit C. There is a randomized algorithm running in time $O(r^{2k/r}|C|(rn)^{O(1)})$ which:*

- *with probability at least $1/2$ answers YES when g contains an r-monomial,*
- *always answers NO when g contains no r-monomial.*

This result was later derandomized by Gabizon et al. [60] under the assumption that the circuit is *non-cancelling*, that is, it contains only input, addition, and multiplication gates. Many concrete problems like r-Simple k-Path can be reduced to (r,k)-Monomial Testing by encoding the set of candidate objects as monomials of some large polynomial, so that "good" objects correspond to monomials with low individual degrees. As we will see in a moment, this is also the case for List $(a{:}b)$-coloring.

Let $(G=(V,E),L)$ be an instance of the List $(a{:}b)$-coloring problem and let $\mathcal{I}$ be the family of all independent sets of G. We denote $n=|V|$. Let $C_a(G,L)$ denote the set of all functions $c : V \to 2^{[a]}$ such that for every edge $uv \in E$ the sets $c(u)$ and $c(v)$ are disjoint, and for every vertex v we have $c(v) \subseteq L(v)$. Consider the following polynomial in $n(a+1)$ variables $\{x_v\}_{v\in V}$ and $\{y_{v,j}\}_{v\in V, j\in[a]}$, over GF(2).

$$p_G = \sum_{\substack{c\in C_a(G,L)\\ \sum_v |c(v)|=bn}} \prod_{v\in V} x_v^{|c(v)|} \prod_{j\in c(v)} y_{v,j}. \tag{6.1}$$

Note that every summand in expression (6.1) has a different set of variables, therefore it corresponds to a monomial (with coefficient 1). Then the following proposition is immediate.

Proposition 67. *There is a list $(a{:}b)$-coloring of graph G iff p_G contains a b-monomial.*

Now we show that p_G can be evaluated relatively fast.

Lemma 68. *The polynomial p_G can be evaluated using a circuit of size $2^n \cdot \max\{a,n\}^{O(1)}$.*

Proof. Consider the following polynomial:

$$q_G = \prod_{j=1}^{a} \sum_{I\in\mathcal{I}} \prod_{v\in I} x_v y_{v,j}. \tag{6.2}$$

Observe that p_G is obtained from q_G by removing all monomials of degree different than $2bn$. Eq. (6.2) shows that q_G can be evaluated by a circuit C_q of size $|\mathcal{I}| \cdot \max\{a,n\}^{O(1)} = 2^n \cdot \max\{a,n\}^{O(1)}$. We obtain from C_q a circuit C_p for p_G by splitting gates according to degrees, in a bottom-up fashion, as follows.

Every input gate u of C_q is replaced with a gate u_1 in C_p. Every addition gate u with inputs x and y in C_q is replaced in C_p by $2an$ addition gates $u_1,\ldots,u_{2an}$, where u_i has inputs x_i and y_i (whenever x_i and y_i exist). Every multiplication gate u with inputs x and y in C_q is replaced in C_p by $2an$ addition gates $u_1,\ldots,u_{2an}$. Moreover, for every pair of integers $1 \le r,s \le 2an$ we create a multiplication gate $u_{r,s}$ with inputs x_r and y_s (whenever they exist) and make it an input of the addition gate u_{r+s}. It is easy to see that for every gate u of C_q, for every i, the gate u_i of C_p evaluates the same polynomial as u, but restricted to monomials in which the total degree is equal to i. When o is the output gate of C_q, then o_{2bn} is the output gate of C_p. Clearly, $|C_p| \le (2an+1)^2|C_q|$. □

Since p_G is a homogeneous polynomial of degree $k = 2bn$, by putting $r = b$ we can combine Proposition 67, Theorem 66 and Lemma 68 to get yet another polynomial-space algorithm for List $(a{:}b)$-Coloring, running in time $O(b^{O(n)}n^{O(1)})$. Similarly, if the running time in Theorem 66 was improved to $2^{o(\log r/r)\cdot k}\cdot|C|\cdot\max\{r,n\}^{O(1)}$, then we would get an algorithm for List $(a{:}b)$-Coloring in time $O(2^{o(\log b)n}n^{O(1)})$, which contradicts ETH by Theorem 11. However, a careful examination shows that this chain of reductions would only yield instances of (r,k)-Monomial Testing with $r = O(\sqrt{k/\log k})$. Hence, this does not exclude the existence of a fast algorithm that works only for large r. Below we show a more direct reduction, which excludes fast algorithms for a wider spectrum of pairs (r,k).

We will use Carry-Less Subset Sum as a convenient starting point for the further reduction.

Lemma 22. *Unless ETH fails, the* Carry-Less Subset Sum *problem cannot be solved in* $2^{o(n)}$ *time.*

Proof. Let φ be an instance of 3-CNF-SAT with N variables and M clauses. By a standard NP-hardness reduction for Subset Sum (see e.g. the textbook of Cormen et al. [35]) in polynomial time one can build an equivalent instance of Carry-Less Subset Sum, with $O(N+M)$ numbers, each having of $O(N+M)$ decimal digits, and with the sum of j-th digit in all the numbers not exceeding 7. In case the number of numbers is different from the length of their decimal representations, we can make them equal by padding the instance by zero numbers or with zeros in the decimal representations. Thus, by Theorem 15, an $2^{o(n)}$ algorithm for Carry-Less Subset Sum would contradict ETH. □

We proceed to reducing Carry-Less Subset Sum to (r,k)-Monomial Testing. Let us choose a parameter $t \in \{1,\ldots,n\}$. Assume w.l.o.g. that t divides n (otherwise, add zeros at the end of every input number). Let $q = n/t$. For an n-digit decimal number x, for every $j = 1,\ldots t$, let $x^{[j]}$ denote the q-digit number given by the j-th block of q digits in x, i.e., $x^{[j]} = (x^{(jq-1)}\ldots x^{((j-1)q)})_{10}$.

Let $r = 10^q - 1$. Define the following polynomial over GF(2):

$$q_S = \prod_{i=1}^{n}\left(y_i + z_i\cdot\prod_{j=1}^{t}x_j^{a_i^{[j]}}\right)\cdot\prod_{j=1}^{t}x_j^{r-s^{[j]}} = \sum_{S\subseteq\{1,\ldots,n\}}\prod_{j=1}^{t}x_j^{\sum_{i\in S}a_i^{[j]}+r-s^{[j]}}\prod_{i\notin S}y_i\prod_{i\in S}z_i.$$

Proposition 69. $(s,a_1,\ldots,a_n)$ *is a YES-instance of* Carry-Less Subset Sum *iff* q_S *contains the monomial* $\prod_{j=1}^{t}x_j^r\prod_{i\notin S}y_i\prod_{i\in S}z_i$, *for some* $S\subseteq\{1,\ldots,n\}$.

Proof. Consider the following polynomial over GF(2):

$$r_S = \sum_{S\subseteq\{1,\ldots,n\}}\prod_{j=1}^{t}x_j^{\sum_{i\in S}a_i^{[j]}+r-s^{[j]}}\prod_{i\notin S}y_i\prod_{i\in S}z_i.$$

The summands in the expression above have unique sets of y_i variables, so each of them corresponds to a monomial (of coefficient 1). It is clear that these monomials

where for every j the degree of x_j is exactly r are in one-to-one correspondence with solutions of the instance $(s, a_1, \ldots, a_n)$. The claim follows by observing that polynomials r_S and q_S coincide. □

Let p_S denote the polynomial obtained from q_S by filtering out all the monomials of degree different than $k = tr + n$.

Proposition 70. *$(s, a_1, \ldots, a_n)$ is a YES-instance of* Carry-Less Subset Sum *iff p_S contains an r-monomial.*

Proof. If $(s, a_1, \ldots, a_n)$ is a YES-instance and let then by Proposition 69 polynomial q_S contains the monomial $\prod_{j=1}^{t} x_j^r \prod_{i \notin S} y_i \prod_{i \in S} z_i$, which is an r-monomial. This monomial has degree $tr + n$, so it is contained in p_S as well.

Conversely, assume p_S contains an r-monomial m. Every monomial of q_s (and hence also of p_S) contains exactly one of the variables y_i and z_i, with degree 1, for every $i = 1, \ldots, n$. It means that the total degree of x_j-type variables in m is tr. Hence, since m is an r-monomial, each of x_j's has degree exactly r. In other words, m is of the form $\prod_{j=1}^{t} x_j^r \prod_{i \notin S} y_i \prod_{i \in S} z_i$, for some $S \subseteq \{1, \ldots, n\}$. Then $(s, a_1, \ldots, a_n)$ is a YES-instance of Carry-Less Subset Sum by Proposition 69. □

Proposition 71. *p_S can be evaluated by a circuit of size $O(nt^2r + n^2t)$, which can be constructed in time polynomial in $n + t + r$.*

Proof. Polynomial q_S can be evaluated by a circuit of size $O(nt)$. The circuit for p_S is built using the construction from Lemma 68. Thus, its size is $O(nt(tr + n)) = O(nt^2r + n^2t)$. □

With this reduction, we obtain our main lower bound for (r, k)-Monomial Testing. We state it in the most general, but technical form, and derive an exemplary corollary below.

Theorem 72. *If there is an algorithm solving (r, k)-*Monomial Testing *in time $2^{o(k \log r/r)}|C|^{O(1)}$, then ETH fails. The statement remains true even if the algorithm works only for instances where $r = 2^{\Theta(n/t(n))}$ and $k = t(n)2^{\Theta(n/t(n))}$, for an arbitrarily chosen function $t : \mathbb{N} \to \mathbb{N}$ computable in $2^{o(n)}$ time, such that $t(n) = \omega(1)$ and $t(n) \leq n$ for every n.*

Proof. By Lemma 22, it suffices to give an algorithm for Carry-Less Subset Sum that works in time $2^{o(n)}$, where n is the number of input numbers. Let $t = t(n)$ and $q = n/t$, $r = 10^q - 1$, $k = tr + n$ as before. Note that $r = 2^{\Theta(n/t(n))}$. Also, since $10^{n/t(n)} = \Omega(n/t(n))$, $k = \Theta(t(n)10^{n/t(n)} + n) = \Theta(t(n)10^{n/t(n)}) = t(n)2^{\Theta(n/t(n))}$.

By Proposition 70, solving Carry-Less Subset Sum is equivalent to detecting an r-monomial in p_S, which is a homogeneous polynomial of degree $k = tr + n$. Let C be the circuit for p_S; by Proposition 71 we have $|C| = O(nt^2r + n^2t)$. If this can be done in time $2^{o(k \log r/r)}|C|^{O(1)}$, we get an algorithm for Carry-Less Subset Sum running in time

$$2^{o(k \log r/r)}|C|^{O(1)} = 2^{o((tr+n)q/r)}(ntr)^{O(1)} = 2^{o(n+nq/10^q)}(ntr)^{O(1)} = 2^{o(n)}(ntr)^{O(1)}.$$

Recall that $t \leq n$ and $r = 10^{n/t} - 1 = 2^{o(n)}$, since $t = t(n) = \omega(1)$. Hence $(ntr)^{O(1)} = 2^{o(n)} n^{O(1)}$. The claim follows. □

Theorem 73. *Let $\sigma \in [0, 1)$. Then, unless ETH fails, there is no algorithm for (r, k)-*Monomial Testing *that solves instances with $r = \Theta(k^\sigma)$ in time $2^{o(k \cdot \frac{\log r}{r})} \cdot |C|^{O(1)}$.*

Proof. We prove that an algorithm for (r, k)-Monomial Testing with properties as in the statement can be used to derive an algorithm for the same problem with properties as in the statement of Theorem 72, which implies that ETH fails. Take t to be a positive integer not larger than n such that

$$\frac{1}{2} \leq \frac{10^{n/t} - 1}{(t \cdot (10^{n/t} - 1) + n)^\sigma} \leq 2; \tag{6.3}$$

it can be easily verified that since $\sigma < 1$, for large enough n such an integer $t \leq n$ always exists. Moreover, we have that $t = t(n) \in \omega(1)$ and $t(n)$ can be computed in polynomial time by brute-force. Hence, $t(n)$ satisfies the properties stated in Theorem 72.

Let $t = t(n)$ and $q = n/t$. Define $r = 10^q - 1$ and $k = tr + n$, then (6.3) is equivalent to

$$1/2 \leq r/k^\sigma \leq 2.$$

Hence $r = \Theta(k^\sigma)$. Consequently, the assumed algorithm solves (r, k)-Monomial Testing in time $2^{o(k \log r/r)} |C|^{O(1)}$, however in the proof of Theorem 72 we have shown that the existence of an algorithm that achieves such a running time for this particular choice of parameters implies that ETH fails. □

In particular, no algorithm solves (r, k)-Monomial Testing in time $2^{o(\frac{\log r}{r}) \cdot k} \cdot |C|^{O(1)}$ for all input values r, unless ETH fails.

Chapter 7

Minimax Approval Voting

In this chapter we prove that if there is no algorithm solving Minimax Approval Voting in time $O^*(2^{o(d \log d)})$ unless ETH fails.

Organization of the chapter. We present two independent reductions proving that there is no algorithm solving Minimax Approval Voting in time $O^*(2^{o(d \log d)})$ assuming ETH: a direct reduction form 3-CNF-SAT in Section 7.1 and then in Section 7.2 a simpler version where the starting problem is $k \times k$-Clique.

Additional notation for this chapter. For a set of words $S \subseteq \{0,1\}^m$ and a word $x \in \{0,1\}^m$ we denote $\mathcal{H}(x,S) = \max_{s \in S} \mathcal{H}(x,s)$. For a string $s \in \{0,1\}^m$, the number of 1's in s is denoted as $n_1(s)$ and it is also called the Hamming weight of s; similarly $n_0(s) = m - n_1(s)$ denotes the number of zeros. Moreover, the set of all strings of length m with k ones is denoted by $S_{k,m}$, i.e., $S_{k,m} = \{s \in \{0,1\}^m : n_1(s) = k\}$. $s[j]$ means the j-th letter of a string s. For a subset of positions $P \subseteq [m]$ we define a subsequence $s|_P$ by removing the letters at positions $[m] \setminus P$ from s.

For a string $s \in \{0,1\}^m$, any string $s' \in S_{k,m}$ at distance $|n_1(s) - k|$ from s is called a *k-completion* of s. Note that it is easy to find such a k-completion s': when $n_1(s) \geq k$ we obtain s' by replacing arbitrary $n_1(s) - k$ ones in s by zeros; similarly when $n_1(s) < k$ we obtain s' by replacing arbitrary $k - n_1(s)$ zeros in s by ones.

7.1 Reduction from (3,4)-CNF-SAT

Lemma 74. *Given a* (3,4)-CNF-SAT *formula φ with n variables one can construct in polynomial time an equivalent instance (S,k,d) of the Minimax Approval Voting problem such that S has $O(n)$ words of length $m = O(n^2/\log^2 n)$, $k = O(n/\log\ n)$ and $d = O(n/\log n)$.*

Proof. Let m denote the number of clauses in φ. Observe that $m \leq \frac{4}{3}n$. Let Var and Cl denote the sets of variables and clauses of φ. Put $\text{Cl} = \{c_1, ..., c_m\}$. Choose a close $c_{i_0} = (x_{r_1}, x_{r_2}, x_{r_3},) \in \text{Cl}$. Now choose $p \in \{1,2,3\}$. There exists exactly one $y \in \text{Var}$ such that $x_{r_p} = y$ or $x_{r_p} = \neg y$. Define $x'_{r_p} = y$.

The instance. There exists a O(1)-coloring of all variables such that there is no pair of variables of the same color in the same clause. Next, each of the O(1) color

classes is partitioned into disjoint groups, each of size at most $A = \lceil \log(n/\log\ n) \rceil$. Denote by k the total number of groups and by $G_1, ..., G_k$ the groups. So $k = O(n/\log(n/\log\ n)) = O(n/\log\ n)$. If a group has $\tau < A$ elements we add $A - \tau$ new variables. So now every group has exactly A elements and $\mathrm{Var} \subseteq \cup_{i\in\{1,...,k\}} G_i$. From the above it follows that for every group of variables G_i there exist exactly 2^A assignments. For $i \in \{1, ..., k\}$ denote by $\Phi_i = \{\phi_i^1, ..., \phi_i^{2^A}\}$ the set of all possible assignments of G_i.

Let us describe the words. Every word L consists of k blocks $L_1, ..., L_k$, each of size $(3/2) \cdot 2^A$. Denote

$$L = (L_1, L_2, ..., L_k) = (l_1^1, l_1^2, ..., l_1^{(3/2)\cdot 2^A}, l_2^1, l_2^2, ..., l_2^{(3/2)\cdot 2^A}, ..., l_k^1, l_k^2, ..., l_k^{(3/2)\cdot 2^A}).$$

For any word v by $l_i^j(v)$ we will denote the suitable letters of the word v. So the size of every word is equal $m = k \cdot O(2^A) = O(n^2/\log^2 n)$. Put $d = (3/2) \cdot 2^A + k - 1 = O(n/\log n)$. Now define the set $\{s^1, ..., s^N\}$ where $N = k + m$. Choose $i_0 \in \{1, ..., k\}$ and define the word s^{i_0}. For $i \in \{1, ..., k\}$ and $j \in \{1, ..., (3/2) \cdot 2^A\}$ let

$$l_i^j(s^{i_0}) = \begin{cases} 1 & \text{if } i = i_0, \\ 0 & \text{otherwise.} \end{cases}$$

Now choose $i_0 \in \{1, ..., m\}$. So we have the clause $c_{i_0} = (x_{r_1}, x_{r_2}, x_{r_3},) \in \mathrm{Cl}$. Let $x'_{r_p} \in G_{i_p}$ for $p \in \{1, 2, 3\}$. The numbers i_1, i_2, i_3 are different. Define the word s^{k+i_0}. For $i \in \{1, ..., k\}$ and $j \in \{1, ..., (3/2) \cdot 2^A\}$ let

$$l_i^j(s^{k+i_0}) = \begin{cases} 1 & \text{if } \exists_{p\in\{1,2,3\}}\ i = i_p,\ j \le 2^A \text{ and } \phi_i^j(x_{r_p}) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Notice that we constructed our instance in polynomial time.

Some remarks. It is easy to see that for every word s^i for $i \in \{1, ..., N\}$ we have $n_1(s^i) = (3/2) \cdot 2^A$. Denote by $v \in \Sigma^m$ a word such that $n_1(v) = k$. Because $d = (3/2) \cdot 2^A + k - 1$ from this it follows that for every word s^i (where $i \in \{1, ..., N\}$) we have $\mathcal{H}(s^i, v) \le d$ if and only if the words s^i and v have at least one common letter 1.

From an assignment to a word. Let $\xi : \mathrm{Var} \to \{0, 1\}$ be a satisfying assignment of φ. We claim that there is a string $v \in \Sigma^m$ with $\max_{x\in S} \mathcal{H}(x, v) \le d$ and $n_1(v) = k$. Define $\overline{\xi} : \cup_{i\in\{1,...,k\}} G_i \to \{0, 1\}$. For $g \in \cup_{i\in\{1,...,k\}} G_i$ put

$$\overline{\xi}(g) = \begin{cases} \xi(g) & \text{if } g \in \mathrm{Var}, \\ 0 & \text{otherwise.} \end{cases}$$

For every $i \in \{1, ..., k\}$ denote $\phi_i^{j_i} = \overline{\xi}\ |_{G_i}$. Now we will define v as follows. For $i \in \{1, ..., k\}$ and $j \in \{1, ..., (3/2) \cdot 2^A\}$ let

$$l_i^j(v) = \begin{cases} 1 & \text{if } j = j_i, \\ 0 & \text{otherwise.} \end{cases}$$

Notice that we have $n_1(L_i) = 1$ for $i \in \{1, ..., k\}$ and $n_1(v) = n_1(L) = k$. From the remarks above it follows that $\max_{x \in \{s^1, ..., s^k\}} \mathcal{H}(x, v) \leq d$.

Now we claim that $\max_{x \in \{s^{k+1}, ..., s^{k+m}\}} \mathcal{H}(x, v) \leq d$. Choose $i_0 \in \{1, ..., m\}$. We will show that $\mathcal{H}(s^{k+i_0}, v) \leq d$. So we have the clause $c_{i_0} = (x_{r_1}, x_{r_2}, x_{r_3},) \in \text{Cl}$. Denote $x'_{r_p} \in G_{i_p}$ for $p \in \{1, 2, 3\}$. Because $\xi : \text{Var} \to \{0, 1\}$ is a satisfying assignment of φ there exists $p \in \{1, 2, 3\}$ such that $\xi(x_{r_p}) = 1$. But $x'_{r_p} \in \text{Var}$, so $\overline{\xi}(x_{r_p}) = 1$. Assume $p = 1$. So $\phi_{i_1}^{j_{i_1}}(x_{r_1}) = \overline{\xi} \mid_{G_{i_1}} (x_{r_1}) = 1$. From this by the definition of the word s^{k+i_0} we have $l_{i_1}^{j_{i_1}}(s^{k+i_0}) = 1$. But $l_{i_1}^{j_{i_1}}(v) = 1$. From the remarks above it follows that $\mathcal{H}(s^{k+i_0}, v) \leq d$. The proof is similar if $p = 2$ or $p = 3$.

From a word to an assignment. Let $v \in \Sigma^m$ be a string with $\max_{x \in S} \mathcal{H}(x, v)$ $\leq d$ and $n_1(v) = k$. We will define $\xi : \text{Var} \to \{0, 1\}$, a satisfying assignment of φ. From the remarks above it follows that for every word s^i (where $i \in \{1, ..., k\}$) the words s^i and v have at least one common letter 1. So for every $i \in \{1, ..., k\}$) we have

$$\sum_{j=1}^{(3/2)\cdot 2^A} l_i^j(v) \geq 1.$$

But

$$\sum_{i=1}^{k} \sum_{j=1}^{(3/2)\cdot 2^A} l_i^j(v) = k.$$

Consequently for every $i \in \{1, ..., k\}$ we have

$$\sum_{j=1}^{(3/2)\cdot 2^A} l_i^j(v) = 1,$$

that means that for every $i \in \{1, ..., k\}$ there exists exactly one number $j_i \in \{1, ..., (3/2)\cdot 2^A\}$ such that $l_i^{j_i}(v) = 1$. Now define $\overline{\xi} : \cup_{i \in \{1, ..., k\}} G_i \to \{0, 1\}$ such that for every $i \in \{1, ..., k\}$

$$\overline{\xi} \mid_{G_i} = \begin{cases} \phi_i^{j_i} & \text{if } j_i \leq 2^A, \\ \phi_i^1 & \text{otherwise.} \end{cases}$$

Because the sets G_i are disjoint the above definition is correct. Define $\xi : \text{Var} \to \{0, 1\}$ by $\xi = \overline{\xi}|_{\text{Var}}$. Now we will show that ξ is the satisfying assignment of φ. Choose a clause $c_{i_0} = (x_{r_1}, x_{r_2}, x_{r_3},) \in \text{Cl}$ and denote $x'_{r_p} \in G_{i_p}$ for $p \in \{1, 2, 3\}$. We claim that ξ satisfies the clause c_{i_0}. Because $\mathcal{H}(s^{k+i_0}, v) \leq d$ from the remarks above it follows that the words s^{k+i_0} and v have at least one common letter 1. So there exist $i \in \{1, ..., k\}$ and $j \in \{1, ..., (3/2) \cdot 2^A\}$ such that $l_i^j(s^{k+i_0}) = l_i^j(v) = 1$. From this it follows that $j = j_i$. From the definition of the word s^{k+i_0} there exists $p \in \{1, 2, 3\}$ such that $i = i_p$, $j \leq 2^A$ and $\phi_i^j(x_{r_p}) = 1$. Because $x_{r_p} \in G_{i_p}$ it is easy to see that $\xi(x_{r_p}) = \overline{\xi}(x_{r_p}) = \phi_i^j(x_{r_p}) = 1$. So ξ satisfies the clause c_{i_0}.

□

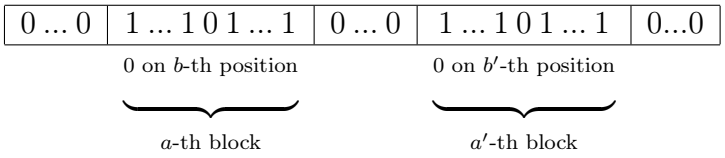

Figure 7.1: Nonedge string.

7.2 Reduction from $k \times k$-Clique

In this subsection we show a lower bound for Minimax Approval Voting parameterized by d. To this end, we use a reduction from $k \times k$-Clique.

Lemma 75. *Given an instance $I = (G, k)$ of $k \times k$-Clique with $k \geq 2$, one can construct an instance $I' = (S, k, d)$ of Minimax Approval Voting, such that I' is a yes-instance iff I is a yes-instance, $d = 3k - 3$ and the set S contains $O(k\binom{2k-2}{k-2})$ strings of length $k^2 + 2k - 2$ each. The construction takes time polynomial in the size of the output.*

Proof. Each string in the set S will be of size $m = k^2 + 2k - 2$. Let us split the set of positions $[m]$ into $k+1$ blocks, where the first k blocks contain exactly k positions each, and the last $(k+1)$-th block contains the remaining $2k-2$ positions. Our construction will enforce that if a solution exists, it will have the following structure: there will be a single 1 in each of the first k blocks and only zeros in the last block. Intuitively the position of the 1 in the first block encodes the clique vertex of the first row of G, the position of the 1 in the second block encodes the clique vertex of the second row of G, etc.

We construct the set S as follows.

- **(nonedge strings)** For each pair of nonadjacent vertices $v, v' \in V(G)$ of G belonging to different rows, i.e., $v = (a, b)$, $v' = (a', b')$, $a \neq a'$, we add to S a string $s_{vv'}$, where all the blocks except a-th and a'-th are filled with zeros, while the blocks a, a' are filled with ones, except the b-th position in block a and the b'-th position in block a' which are zeros (see Fig. 7.1). Formally, $s_{vv'}$ contains ones at positions $\{(a-1)k + j : j \in [k], j \neq b\} \cup \{(a'-1)k + j : j \in [k], j \neq b'\}$. Note that the Hamming weight of $s_{vv'}$ equals $2k - 2$.

- **(row strings)** For each row $i \in [k]$ we create exactly $\binom{2k-2}{k-2}$ strings, i.e., for $i \in [k]$ and for each set X of exactly $k-2$ positions in the $(k+1)$-th block we add to S a string $s_{i,X}$ having ones at all positions of the i-th block and at X, all the remaining positions are filled with zeros (see Fig. 7.2). Note that similarly as for the nonedge strings the Hamming weight of each row string equals $2k-2$, and to achieve this property we use the $(k+1)$-th block.

To finish the description of the created instance $I' = (S, k, d)$ we need to define the target distance d, which we set as $d = 3k - 3$. Observe that as the Hamming weight of each string $s' \in S$ equals $2k - 2$, for $s \in \{0, 1\}^m$ with exactly k ones we

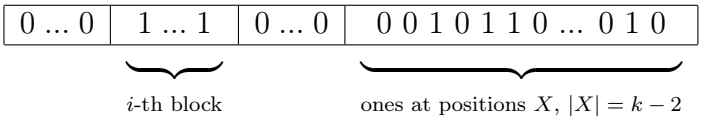

Figure 7.2: Row string.

have $\mathcal{H}(s, s') \leq d$ if and only if the positions of ones in s and s' have a non-empty intersection.

Let us assume that there is a clique K in G of size k containing exactly one vertex from each row. For $i \in [k]$ let $j_i \in [k]$ be the column number of the vertex of K from row i. Define s as a string containing ones exactly at positions $\{(i-1)k+j_i : i \in [k]\}$, i.e., the $(k+1)$-th block contains only zeros and for $i \in [k]$ the i-th block contains a single 1 at position j_i. Obviously s contains exactly k ones, hence it suffices to show that s has at least one common one with each of the strings in S. This is clear for the row strings, as each row string contains a block full of ones. For a nonedge string $s_{vv'}$, where $v = (a, b)$ and $v' = (a', b')$ note that K does not contain v and v' at the same time. Consequently s has a common one with $s_{vv'}$ in at least one of the blocks a, a'.

In the other direction, assume that s is a string of length m with exactly k ones such that the Hamming distance between s and each of the strings in S is at most d, which by construction implies that s has a common one with each of the strings in S. First, we are going to prove that s contains a 1 in each of the first k blocks (and consequently has only zeros in block $k+1$). For the sake of contradiction assume that this is not the case. Consider a block $i \in [k]$ containing only zeros. Let X be any set of $k-2$ positions in block $k+1$ holding only zeros in s (such a set exists as block $k+1$ has $2k-2$ positions). But the row string $s_{i,X}$ has $2k-2$ ones at positions where s has zeros, and consequently $\mathcal{H}(s, s_{i,X}) = k + (2k-2) = 3k-2 > d = 3k-3$, a contradiction.

As we know that s contains exactly one one in each of the first k blocks let $j_i \in [k]$ be such a position of block $i \in [k]$. Create $X \subseteq V(G)$ by taking the vertex from column j_i for each row $i \in [k]$. Clearly X is of size k and it contains exactly one vertex from each row, hence it remains to prove that X is a clique in G. Assume the contrary and let $v, v' \in X$ be two distinct nonadjacent vertices of X, where $v = (i, j_i)$ and $v' = (i', j_{i'})$. Observe that the nonedge string $s_{vv'}$ contains zeros at the j_i-th position of the i-th block and at the $j_{i'}$-th position of the i'-th block. Since for $i'' \in [k]$, $i'' \neq i$, $i'' \neq i'$ block i'' of $s_{vv'}$ contains only zeros, we infer that the sets of positions of ones of s and $s_{vv'}$ are disjoint leading to $\mathcal{H}(s, s_{vv'}) = k + (2k-2) = 3k-2 > d$, a contradiction.

As we have proved that I is a yes-instance of $k \times k$-Clique iff I' is a yes-instance of Minimax Approval Voting, the lemma follows. □

We are ready to prove the main result of this section.

Theorem 76. *There is no* $2^{o(d \log d)} \max\{n, m\}^{O(1)}$*-time algorithm for* Minimax Approval Voting *unless ETH fails.*

Proof. Using Lemma 75, the input instance G of $k \times k$-Clique is transformed into an equivalent instance $I' = (S, k, d)$ of Minimax Approval Voting, where $n = |S| = O(k\binom{2k-2}{k-2}) = 2^{O(k)}$, each string of S has length $m = O(k^2)$ and $d = \Theta(k)$. Using a $2^{o(d \log d)} \cdot \max\{n, m\}^{O(1)}$-time algorithm for Minimax Approval Voting we can solve $k \times k$-Clique in time $2^{o(k \log k)} 2^{O(k)} = 2^{o(k \log k)}$, which contradicts ETH by Theorem 20. □

Chapter 8

4-OPT Detection for Traveling Salesman Problem

In this chapter we prove that for any $\epsilon > 0$ an algorithm solving 4-OPT Optimization in time $O(n^{3-\epsilon})$ would imply an algorithm solving All Pairs Shortest Paths in time $O(n^{3-\delta})$ for some $\delta > 0$.

More precisely, we work with the decision version, called 4-OPT Detection, where the input is the same as in 4-OPT Optimization and the goal is to determine if there is a 4-move which improves the weight of the given Hamiltonian cycle. To this end, we reduce the Negative Edge-Weighted Triangle problem, where the input is an undirected, complete graph G, and a weight function $w : E(G) \to \mathbb{Z}$. The goal is to determine whether G contains a triangle whose total edge-weight is negative.

Lemma 77. *Every instance $I = (G, w)$ of* Negative Edge-Weighted Triangle *can be reduced in $O(|V(G)|^2)$ time into an instance $I' = (G', w', C)$ of* 4-OPT Detection *such that G contains a triangle of negative weight iff I' admits an improving* 4*-move. Moreover, $|V(G')| = O(|V(G)|)$, and the maximum absolute weight in w' is larger by a constant factor than the maximum absolute weight in w.*

Proof. Let $V(G) = \{v_1, \ldots, v_n\}$. Then let $V_{\mathsf{up}} = \{a_1, b_1, \ldots, a_n, b_n\}$, $V_{\mathsf{down}} = \{a'_1, b'_1, \ldots, a'_n, b'_n\}$ and $V(G') = V_{\mathsf{up}} \dot\cup V_{\mathsf{down}}$. Let W be the maximum absolute value of a weight in w. Then let $M_1 = 5W + 1$ and $M_2 = 21M_1 + 1$ and let

$$w'(u,v) = \begin{cases} 0 & \text{if } (u,v) \text{ is of the form } (a_i, b'_i) \\ w(v_i, v_j) & \text{if } (u,v) \text{ is of the form } (a_i, b_j) \text{ for } i < j \text{ or } (a'_i, b_j) \text{ for } j < i \\ M_1 & \text{if } (u,v) \text{ is of the form } (a_i, b_i) \\ -3M_1 & \text{if } (u,v) \text{ is of the form } (a'_i, b'_i) \\ -M_2 & \text{if } (u,v) \text{ is of the form } (b_i, a_{i+1}) \text{ or } (b'_i, a'_{i+1}) \text{ or } (a_1, a'_1) \text{ or } (b_n, b'_n) \\ M_2 & \text{in other case.} \end{cases}$$

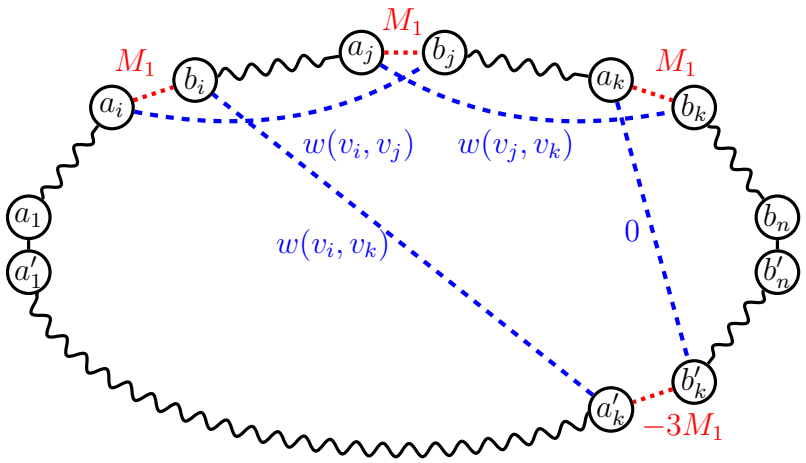

Figure 8.1: A simplified view of the instance (G', w', C) together with an example of a 4-move. The added edges are marked as blue (dashed) and the removed edges are marked as red (dotted).

Note that the cases are not overlapping. (Note also that although some weights are negative, we can get an equivalent instance with nonnegative weights by adding M_2 to all the weights.) Let $C = a_1, b_1, \ldots, a_n, b_n, b'_n, a'_n, \ldots, b'_1, a'_1$. The construction is illustrated in Fig. 8.1

If there is a negative triangle v_i, v_j, v_k for some $i < j < k$ in G then we can improve C by removing edges $(a_i, b_i), (a_j, b_j), (a_k, b_k)$ and (a'_k, b'_k) and inserting edges $(a_i, b_j), (a_j, b_k), (a_k, b'_k)$ and (a'_k, b_i). The total weight of the removed edges is $M_1 + M_1 + M_1 + (-3M_1) = 0$ and the total weight of the inserted edges is $w(v_i, v_j) + w(v_j, v_k) + 0 + w(v_k, v_i) < 0$ hence indeed the cycle is improved.

Let us assume that C can be improved by removing 4 edges and inserting 4 edges. Note that all the edges of weight $-M_2$ belong to C and all the edges of weight M_2 do not belong to C. All the other edges have absolute values of their weights bounded by $3M_1$. Therefore even a single edge of the weight $-M_2$ cannot be removed and even a single edge of the weight M_2 cannot be inserted because a loss of M_2 cannot be compensated by any other 7 edges (inserted or removed), as they can result in a gain of at most $7 \cdot 3M_1 < M_2$. Hence in the following we treat edges of weights $\pm M_2$ as fixed, i.e., they cannot be inserted or removed from the cycle. Note that the edges of C that can be removed are only the edges of the form (a_i, b_i) (of weights M_1) and (a'_i, b'_i) (of weights $-3M_1$).

All the edges of weight $-3M_1$ already belong to C and all the remaining edges of the graph that can be inserted or removed from the cycle are the edges of the weight M_1 belonging to C and the edges of absolute values of their weights bounded by W. Therefore we cannot remove more than one edge of the weight $-3M_1$ from C because a loss of $6M_1$ cannot be compensated by any 2 removed and 4 inserted edges (we could potentially gain only $2M_1 + 4W < 3M_1$). Hence we can remove at most one edge of the weight $-3M_1$ from C. For the same reason if we do remove one edge of

the weight $-3M_1$ (i.e., of the form (a'_i, b'_i)) from C we need to remove also three edges of the weights M_1 (i.e., of the form (a_j, b_j)) in order to compensate the loss of $3M_1$ (otherwise we could compensate up to $2M_1 + 5W < 3M_1$).

Note that the only edges that can be added (i.e., the edges with the weights less than M_2 that do not belong to C) are the edges of the form (a_i, b_j) for $i < j$, (a'_i, b_j) for $j < i$ and (a_i, b'_i). Therefore if the removed edges from $G[V_{\mathsf{up}}]$ are $(a_{i_1}, b_{i_1}), \ldots, (a_{i_\ell}, b_{i_\ell})$ for some $i_1 < \ldots < i_\ell$ (and no other edges belonging to $G[V_{\mathsf{up}}]$) then in order to close the cycle we need to insert some edge incident to b_{i_1} but since for any $i_0 < i_1$ there is no removed edge (a_{i_0}, b_{i_0}) it cannot be an edge of the form (a_{i_0}, b_{i_1}). Hence it has to be an edge of the form (a'_j, b_{i_1}) for some $j > i_1$. But then also the edge (a'_j, b'_j) has to be removed. Therefore if we remove at least one edge of the form (a_i, b_i) then we need to remove also an edge of the form (a'_j, b'_j) (and as we know this implies also that at least three edges of the form (a_i, b_i) have to be removed). So if any edge is removed, then exactly three edges of the form (a_i, b_i) and exactly one edge of the form (a'_j, b'_j) have to be removed. Note that this implies also that the total weight of the removed edges has to be equal to zero.

Clearly the move has to remove at least one edge in order to improve the weight of the cycle. Let us assume that the removed edges are (a_i, b_i), (a_j, b_j) and (a_k, b_k) for some $i < j < k$ and (a'_ℓ, b'_ℓ) for some ℓ. For the reason mentioned in the previous paragraph in order to obtain a Hamiltonian cycle one of the inserted edges has to be the edge (a'_ℓ, b_i). Also the vertex b_j has to be connected with something but the vertex a'_ℓ is already taken and hence it has to be connected with the vertex a_i. Similarly the vertex b_k has to be connected with a_j because a'_ℓ and a_i are already taken. Thus a_k has to be connected with b'_ℓ and this means that $k = \ell$. The total weight change of the move is negative and therefore the total weight of the added edges has to be negative (since the total weight of the removed edges is equal to zero). Thus we have $w(v_i, v_j) + w(v_j, v_k) + w(v_k, v_i) = w'(a_i, b_j) + w'(a_j, b_k) + w'(a'_k, b_i) + w'(a_k, b'_k) < 0$. So v_i, v_j, v_k is a negative triangle in (G, w). □

Theorem 13. *If there is $\epsilon > 0$ such that* 4-OPT Detection *admits an algorithm in time $O(n^{3-\epsilon} \cdot (\log M)^{O(1)})$, then there is $\delta > 0$ such that both* Negative Edge-Weighted Triangle *and* All Pairs Shortest Paths *admit an algorithm in time $O(n^{3-\delta} \cdot (\log M)^{O(1)})$, where in all cases we refer to n-vertex input graphs with integer weights from $\{-M, \ldots, M\}$.*

Proof. The first part of the claim follows from Lemma 77, while the second part follows from the reduction of All Pairs Shortest Paths to Negative Edge-Weighted Triangle by Vassilevska-Williams and Williams (Theorem 1.1 in [132]). □

Chapter 9

Further Work

An obvious idea for further work is improving some lower bounds from this thesis which are not tight enough. Two most natural examples are listed below.

- Is there a $2^{n \cdot o(\log \ell)} \cdot r^{O(1)}$-time algorithm for Channel Assignment with the weight function bounded by ℓ?
- Is there a $2^{o(n^2)}$-time algorithm for Rainbow k-Coloring?

For some problems considered in this thesis one can formulate complexity questions which were not addressed here. Let us mention three of them.

- Is there an $n^{O(1/\log \epsilon)}$-time PTAS for Minimax Approval Voting or Closest String?
- Is there a $n^{o(k)}$-time algorithm for k-OPT Detection?
- Is there a reduction from 4-OPT Detection to All Pairs Shortest Paths showing that if there exists an algorithm solving All Pairs Shortest Paths in time $O(n^{3-\delta} \cdot (\log M)^{O(1)})$ for some $\delta > 0$ then there exists an algorithm solving 4-OPT Detection in time $O(n^{3-\epsilon} \cdot (\log M)^{O(1)})$ for some $\epsilon > 0$? This question is in fact a question whether 4-OPT Detection belongs to the class of *subcubic equivalence* to the All Pairs Shortest Paths problem (note that in Chapter 8 we presented the reduction in the other direction). This class is a class of problems including All Pairs Shortest Paths such that either each problem of this class admits an algorithm working in time $O(n^{3-\epsilon})$ for some $\epsilon > 0$ depending on the problem or none of these problems admits such a subquadratic algorithm.

Finally there are some fundamental open problems in the area of fine-grained complexity which were not mentioned in this thesis. (For the definitions of the problems below, we refer the reader to Appendix A.)

- Is there a $2^{o(n^2)}$-time algorithm for Edge Coloring?
- Is there a $2^{O(n)} \cdot r^{O(1)}$-time algorithm for Integer Linear Programming?

	lower bound under ETH	upper bound
Channel Assignment with ℓ-bounded weight function	$2^{n\cdot\Omega(\log\log\ell)}\cdot r^{O(1)}$	$2^{n\cdot O(\log\ell)}\cdot r^{O(1)}$
Rainbow k-Coloring	$2^{\Omega(n^{3/2})}$	$2^{O(n^2)}$
PTAS for Minimax Approval Voting	$n^{\Omega(1/\epsilon)}$	$n^{O(\log\epsilon/\epsilon^2)}$
PTAS for Closest String	$n^{\Omega(1/\epsilon)}$	$n^{O(1/\epsilon^2)}$
Edge Coloring	$2^{\Omega(n)}$	$2^{O(n^2)}$
Integer Linear Programming	$2^{\Omega(n)}\cdot r^{O(1)}$	$2^{O(n\log n)}\cdot r^{O(1)}$
Generalized T-Coloring and Generalized List T-coloring with ℓ-bounded weight function	$2^{n\cdot\Omega(\log\ell/\log^2\log\ell)}\cdot r^{O(1)}$	$2^{n\cdot O(\log\ell)}\cdot r^{O(1)}$
k-OPT Detection	$n^{\Omega(k/\log k)}$	$n^{O(k)}$
Subgraph Isomorphism with $k := \|E(G)\|$	$f(k)n^{\Omega(k/\log k)}$	$n^{O(k)}$

Table 9.1: Differences between known lower bounds under ETH and known algorithms for various problems.

- Is there an $f(k)n^{o(k)}$-time algorithm for Subgraph Isomorphism where n is the number of the vertices in the host graph, k is the number of edges of the pattern graph and f is an arbitrary function (see [111])?
- Given a family of graphs $\mathcal{H}$ can we infer the time complexity of Graph Homomorphism or Subgraph Isomorphism problems restricted only to host graphs that belongs to $\mathcal{H}$? In particular, is there a criterion that settles whether for a given family $\mathcal{H}$ there exists an algorithm that in time c^n for some $c > 0$ solves Graph Homomorphism restricted only to host graphs that belongs to $\mathcal{H}$? For example if $\mathcal{H}$ is the family of all cliques then there exists such a c^n-time algorithm since this problem is exactly the Graph Coloring problem that can be solved in time $O^*(2^n)$ [12]. But if $\mathcal{H}$ is the family of all Kneser graphs then Corollary 10 states that there is no such c^n algorithm for any $c > 0$. Can we say something more about this c^n dichotomy? Perhaps we can even chop the spectrum of different time complexities that belong to $O(h^n)$ into segments and provide a set of criteria that tell us which segment of the complexities the problem of Graph Homomorphism with host graphs from the given family $\mathcal{H}$ belongs to. Very recently a slightly different dichotomy for *counting* subgraphs was found by Curticapean, Dell and Marx [37]. Namely, depending whether the vertex-cover number is bounded in the given family of pattern graphs $\mathcal{G}$ the problem of counting subgraphs restricted only to pattern graphs G that belongs to $\mathcal{G}$ is either polynomial-time solvable or #W[1]-complete when parametrized by the pattern size $|V(G)|$.

For the known bounds see Table 9.1.

Bibliography

[1] School on parameterized algorithms and complexity - open problems. In *http://fptschool.mimuw.edu.pl/opl.pdf*, page 8, 2014. 11

[2] H. Abasi, N. H. Bshouty, A. Gabizon, and E. Haramaty. On r-simple k-path. In *MFCS 2015*, volume 8635 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2014. 17, 91

[3] A. Abboud, F. Grandoni, and V. V. Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In P. Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1681–1697. SIAM, 2015. 7

[4] A. Abboud, V. V. Williams, and H. Yu. Matching triangles and basing hardness on an extremely popular conjecture. In R. A. Servedio and R. Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 41–50. ACM, 2015. 7

[5] O. Amini, F. V. Fomin, and S. Saurabh. Counting subgraphs via homomorphisms. *SIAM J. Discrete Math.*, 26(2):695–717, 2012. 11

[6] P. Ananth, M. Nasre, and K. K. Sarpatwar. Rainbow connectivity: Hardness and tractability. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011)*, pages 241–251, 2011. 12

[7] A. Andoni, P. Indyk, and M. Patrascu. On the Optimality of the Dimensionality Reduction Method. In *47th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2006*, pages 449–458, 2006. 19

[8] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5):753–782, 1998. 20

[9] A. Backurs and P. Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In R. A. Servedio and R. Rubinfeld,

editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 51–58. ACM, 2015. 7

[10] R. Beigel and D. Eppstein. 3-coloring in time $o(1.3289^n)$. *J. Algorithms*, 54(2):168–204, 2005. 6, 11

[11] A. Björklund. Determinant sums for undirected Hamiltonicity. *SIAM J. Comput.*, 43(1):280–299, 2014. 11

[12] A. Björklund, T. Husfeldt, and M. Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009. 9, 11, 15, 16, 108

[13] M. Bonamy, L. Kowalik, M. Pilipczuk, A. Socała, and M. Wrochna. Tight lower bounds for the complexity of multicoloring. *CoRR (To appear in the proceedings of The 25th Annual European Symposium on Algorithms, ESA 2017)*, abs/1607.03432, 2016. 23

[14] N. Bourgeois, B. Escoffier, V. T. Paschos, and J. M. M. van Rooij. Fast algorithms for max independent set. *Algorithmica*, 62(1-2):382–415, 2012. 11

[15] S. J. Brams, D. M. Kilgour, and M. R. Sanver. A Minimax Procedure for Electing Committees. *Public Choice*, 132(3-4):401–420, 2007. 18

[16] N. H. Bshouty. Optimal algorithms for the coin weighing problem with a spring scale. In *COLT 2009*, 2009. 82

[17] J. Byrka and K. Sornat. PTAS for Minimax Approval Voting. In *Proceedings of 10th International Conference Web and Internet Economics, WINE 2014*, pages 203–217, 2014. 19

[18] L. Cai, J. Chen, R. G. Downey, and M. R. Fellows. On the structure of parameterized problems in NP. *Inf. Comput.*, 123(1):38–49, 1995. 90

[19] I. Caragiannis, D. Kalaitzis, and E. Markakis. Approximation Algorithms and Mechanism Design for Minimax Approval Voting. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI*, 2010. 19

[20] M. L. Carmosino, J. Gao, R. Impagliazzo, I. Mihajlin, R. Paturi, and S. Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In M. Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 261–270. ACM, 2016. 7

[21] Y. Caro, A. Lev, Y. Roditty, Z. Tuza, and R. Yuster. On rainbow connection. *Electron. J. Combin*, 15(1):R57, 2008. 12

[22] S. Chakraborty, E. Fischer, A. Matsliah, and R. Yuster. Hardness and algorithms for rainbow connection. *Journal of Combinatorial Optimization*, 21(3):330–347, 2009. 12, 13, 61, 62

[23] J. R. Chamberlin and P. N. Courant. Representative Deliberations and Representative Decisions: Proportional Representation and the Borda Rule. *American Political Science Review*, 77:718–733, 9 1983. 18

[24] B. Chandra, H. J. Karloff, and C. A. Tovey. New results on the old k-opt algorithm for the traveling salesman problem. *SIAM J. Comput.*, 28(6):1998–2029, 1999. 21

[25] L. S. Chandran and D. Rajendraprasad. Rainbow Colouring of Split and Threshold Graphs. *Computing and Combinatorics*, pages 181–192, 2012. 12

[26] L. S. Chandran and D. Rajendraprasad. Inapproximability of rainbow colouring. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013)*, pages 153–162, 2013. 12

[27] L. S. Chandran, D. Rajendraprasad, and M. Tesař. Rainbow colouring of split graphs. *Discrete Applied Mathematics*, 2015. To appear. 12

[28] G. Chartrand, G. L. Johns, K. A. McKeon, and P. Zhang. Rainbow connection in graphs. *Mathematica Bohemica*, 133(1), 2008. 12

[29] G. Chartrand and P. Zhang. *Chromatic graph theory*. CRC press, 2008. 12

[30] M. G. Christ, L. M. Favrholdt, and K. S. Larsen. Online multi-coloring with advice. In *WAOA 2014*, volume 8952 of *Lecture Notes in Computer Science*, pages 83–94. Springer, 2014. 15

[31] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, DTIC Document, 1976. 20

[32] V. Chvátal. Mastermind. *Combinatorica*, 3(3):325–329, 1983. 82

[33] V. Chvátal, M. Garey, and D. Johnson. Two results concerning multicoloring. In *Algorithmic Aspects of Combinatorics*, volume 2 of *Annals of Discrete Math.*, pages 151–154. Elsevier, 1978. 15

[34] V. Conitzer. Making Decisions Based on the Preferences of Multiple Agents. *Commun. ACM*, 53(3):84–94, 2010. 17

[35] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2009. 27, 93

[36] G. A. Croes. A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812, 1958. 20

[37] R. Curticapean, H. Dell, and D. Marx. Homomorphisms are a good basis for counting small subgraphs. In *STOC*, pages 210–223. ACM, 2017. 11, 108

[38] M. Cygan, F. V. Fomin, A. Golovnev, A. S. Kulikov, I. Mihajlin, J. Pachocki, and A. Socała. Tight bounds for Graph Homomorphism and Subgraph Isomorphism. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1643–1649, 2016. 12, 22, 27, 60

[39] M. Cygan, F. V. Fomin, A. Golovnev, A. S. Kulikov, I. Mihajlin, J. W. Pachocki, and A. Socała. Tight lower bounds on graph embedding problems. *CoRR*, abs/1602.05016, 2016. 26

[40] M. Cygan, F. V. Fomin, A. Golovnev, A. S. Kulikov, I. Mihajlin, J. W. Pachocki, and A. Socała. Tight lower bounds on graph embedding problems. *Journal of the ACM (JACM)*, (to appear), 2017. 12, 22

[41] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015. 11, 25, 26

[42] M. Cygan and Ł. Kowalik. Channel assignment via fast zeta transform. *Inf. Process. Lett.*, 111(15):727–730, 2011. 9

[43] M. Cygan, L. Kowalik, and A. Socała. Improving TSP tours using dynamic programming over tree decomposition. *CoRR (To appear in the proceedings of the 25th Annual European Symposium on Algorithms, ESA 2017)*, abs/1703.05559, 2017. 21, 22, 23

[44] M. Cygan, L. Kowalik, A. Socała, and K. Sornat. Approximation and parameterized complexity of minimax approval voting. In S. P. Singh and S. Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 459–465. AAAI Press, 2017. 20, 23

[45] M. Cygan, J. Pachocki, and A. Socała. The hardness of subgraph isomorphism. *CoRR*, abs/1504.02876, 2015. 22

[46] M. Cygan and M. Pilipczuk. Bandwidth and distortion revisited. *Discrete Applied Mathematics*, 160(4-5):494–504, 2012. 11

[47] M. Cygan, M. Pilipczuk, and M. Pilipczuk. Known algorithms for EDGE CLIQUE COVER are probably optimal. In S. Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1044–1053. SIAM, 2013. 9

[48] M. de Berg, K. Buchin, B. M. P. Jansen, and G. J. Woeginger. Fine-grained complexity analysis of two classic TSP variants. In *ICALP*, volume 55 of *LIPIcs*, pages 5:1–5:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. 21, 22

[49] R. Diestel. *Graph Theory*. Springer-Verlag Heidelberg, 2010. 25

[50] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank Aggregation Methods for the Web. In *Proceedings of the Tenth International World Wide Web Conference, WWW 2001*, pages 613–622, 2001. 18

[51] E. Eiben, R. Ganian, and J. Lauri. On the complexity of rainbow coloring problems. In *Proceedings of the Twenty-Sixth International Workshop on Combinatorial Algorithms, IWOCA 2015, Verona, Italy, October 5-7*, pages 209–220, 2015. 12

[52] E. Elkind, P. Faliszewski, P. Skowron, and A. Slinko. Properties of Multiwinner Voting Rules. *Social Choice and Welfare*, 48(3):599–632, 2017. 18

[53] P. C. Fishburn. Axioms for Approval Voting: Direct Proof. *Journal of Economic Theory*, 19(1):180–185, 1978. 18

[54] D. C. Fisher. Fractional colorings with large denominators. *J. Graph Theory*, 20(4):403–409, 1995. 15

[55] F. Fomin, K. Iwama, and D. Kratsch. Moderately Exponential Time Algorithms (Dagstuhl Seminar 08431). In *Dagstuhl Reports, http://drops.dagstuhl.de/opus/volltexte/2008/1798/pdf/08431.SWM.Paper.1798.pdf*, page 1, 2008. 11

[56] F. V. Fomin, A. Golovnev, A. S. Kulikov, and I. Mihajlin. Lower bounds for the graph homomorphism problem. *CoRR*, abs/1502.05447, 2015. 11, 49

[57] F. V. Fomin, A. Golovnev, A. S. Kulikov, and I. Mihajlin. Tight bounds for subgraph isomorphism and graph homomorphism. *CoRR*, abs/1507.03738, 2015. 12, 27, 60

[58] F. V. Fomin, P. Heggernes, and D. Kratsch. Exact algorithms for graph homomorphisms. *Theory Comput. Syst.*, 41(2):381–393, 2007. 16

[59] F. V. Fomin and D. Kratsch. *Exact exponential algorithms*. Springer Science & Business Media, 2010. 11

[60] A. Gabizon, D. Lokshtanov, and M. Pilipczuk. Fast algorithms for parameterized problems with relaxed disjointness constraints. In *ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, pages 545–556. Springer, 2015. 17, 92

[61] A. Gajentaan and M. H. Overmars. On a class of o(n2) problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995. 7

[62] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 26

[63] C. Godsil and G. F. Royle. *Algebraic Graph Theory*. Springer, 2001. 89

[64] J. Gramm, R. Niedermeier, and P. Rossmanith. Fixed-Parameter Algorithms for Closest String and Related Problems. *Algorithmica*, 37(1):25–42, 2003. 19

[65] V. Grebinski and G. Kucherov. Optimal reconstruction of graphs under the additive model. *Algorithmica*, 28(1):104–124, 2000. 82, 83

[66] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981. Corrigendum available at: http://dx.doi.org/10.1007/BF02579139. 16

[67] J. Guo, S. Hartung, R. Niedermeier, and O. Suchý. The parameterized complexity of local search for tsp, more refined. *Algorithmica*, 67(1):89–110, 2013. 21, 22

[68] M. M. Halldórsson and G. Kortsarz. Multicoloring: Problems and techniques. In *MFCS 2013*, volume 3153 of *Lecture Notes in Computer Science*, pages 25–41. Springer, 2004. 15

[69] M. M. Halldórsson, G. Kortsarz, A. Proskurowski, R. Salman, H. Shachnai, and J. A. Telle. Multicoloring trees. *Inf. Comput.*, 180(2):113–129, 2003. 15

[70] F. Havet. Channel assignment and multicolouring of the induced subgraphs of the triangular lattice. *Discrete Math.*, 233(1-3):219–231, 2001. 15

[71] M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962. 6, 20

[72] P. Hell and J. Nešetřil. On the complexity of H-coloring. *J. Comb. Theory, Ser. B*, 48(1):92–110, 1990. 15

[73] K. Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106 – 130, 2000. 21

[74] Q. Hua, Y. Wang, D. Yu, and F. C. M. Lau. Dynamic programming based algorithms for set multicover and multiset multicover problems. *Theor. Comput. Sci.*, 411(26-28):2467–2474, 2010. 15

[75] T. Husfeldt, R. Paturi, G. B. Sorkin, and R. Williams. Exponential Algorithms: Algorithms and Complexity Beyond Polynomial Time (Dagstuhl Seminar 13331). *Dagstuhl Reports*, 3(8):40–72, 2013. 9, 11

[76] R. Impagliazzo and R. Paturi. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. 5, 6, 19, 25

[77] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. 5, 6, 25

[78] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988. 21

[79] S. Jukna. On set intersection representations of graphs. *Journal of Graph Theory*, 61(1):55–75, 2009. 63, 74, 75

[80] R. M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Operations Research Letters*, 1(2):49–51, 1982. 20

[81] M. Kchikech and O. Togni. Approximation algorithms for multicoloring planar graphs and powers of square and triangular meshes. *Discrete Math. Theor. Comput. Sci.*, 8(1):159–172, 2006. 15

[82] D. M. Kilgour. Approval Balloting for Multi-winner Elections. In J.-F. Laslier and R. M. Sanver, editors, *Handbook on Approval Voting*, pages 105–124. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. 18

[83] L. Kowalik, J. Lauri, and A. Socała. On the fine-grained complexity of rainbow coloring. In P. Sankowski and C. D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 58:1–58:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. 13, 23

[84] Ł. Kowalik and A. Socała. Assigning channels via the meet-in-the-middle approach. In R. Ravi and I. L. Gørtz, editors, *SWAT*, volume 8503 of *Lecture Notes in Computer Science*, pages 282–293. Springer, 2014. 9

[85] D. Král. An exact algorithm for the channel assignment problem. *Discrete Applied Mathematics*, 145(2):326–331, 2005. 9

[86] M. W. Krentel. On finding and verifying locally optimal solutions. *SIAM J. Comput.*, 19(4):742–749, 1990. 21

[87] F. Kuhn. Local multicoloring algorithms: Computing a nearly-optimal TDMA schedule in constant time. In *STACS 2009*, volume 3 of *LIPIcs*, pages 613–624. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009. 15

[88] M. Künnemann and B. Manthey. Towards understanding the smoothed approximation ratio of the 2-opt heuristic. In *ICALP (1)*, volume 9134 of *Lecture Notes in Computer Science*, pages 859–871. Springer, 2015. 21

[89] J. Laslier and M. Sanver. *Handbook on Approval Voting*. Studies in Choice and Welfare. Springer Berlin Heidelberg, 2010. 18

[90] E. L. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Letters*, 5(3):66–67, 1976. 6

[91] V. B. Le and Z. Tuza. Finding optimal rainbow connection is hard. Technical Report CS-03-09, Universität Rostock, 2009. 12

[92] R. LeGrand. Analysis of the Minimax Procedure. Technical Report WUCSE-2004-67, Department of Computer Science and Engineering, Washington University, St. Louis, Missouri, 2004. 18, 19

[93] R. LeGrand, E. Markakis, and A. Mehta. Some Results on Approximating the Minimax Solution in Approval Voting. In *6th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2007*, pages 1193–1195, 2007. 18, 19

[94] M. Li, B. Ma, and L. Wang. On the Closest String and Substring Problems. *Journal of the ACM*, 49(2):157–171, 2002. 19

[95] X. Li, Y. Shi, and Y. Sun. Rainbow Connections of Graphs: A Survey. *Graphs and Combinatorics*, 29(1):1–38, 2012. 12

[96] S. Lin. Computer solutions of the traveling salesman problem. *The Bell System Technical Journal*, 44(10):2245–2269, 1965. 20

[97] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973. 20

[98] W. Lin. Multicoloring and Mycielski construction. *Discrete Math.*, 308(16):3565 – 3573, 2008. 15

[99] B. Lindström. On a combinatorial problem in number theory. *Canad. Math. Bull*, 8(4):477–490, 1965. 82

[100] H. Liu and J. Guo. Parameterized Complexity of Winner Determination in Minimax Committee Elections. In *Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2016*, pages 341–349, 2016. 19

[101] D. Lokshtanov, D. Marx, and S. Saurabh. Known algorithms on graphs on bounded treewidth are probably optimal. In D. Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 777–789. SIAM, 2011. 9

[102] D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011. 6

[103] D. Lokshtanov, D. Marx, and S. Saurabh. Slightly superexponential parameterized problems. In D. Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 760–776. SIAM, 2011. 9, 19, 27, 53

[104] D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of EATCS*, 3(105), 2013. 26

[105] L. Lovász. Kneser's conjecture, chromatic number, and homotopy. *J. Comb. Theory, Ser. A*, 25(3):319–324, 1978. 15

[106] T. Lu and C. Boutilier. Budgeted Social Choice: From Consensus to Personalized Decision Making. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011*, pages 280–286, 2011. 18

[107] B. Ma and X. Sun. More Efficient Algorithms for Closest String and Substring Problems. *SIAM Journal of Computing*, 39(4):1432–1443, 2009. 19

[108] B. Manthey and R. Veenstra. Smoothed analysis of the 2-opt heuristic for the TSP: polynomial bounds for gaussian noise. In *ISAAC*, volume 8283 of *Lecture Notes in Computer Science*, pages 579–589. Springer, 2013. 21

[109] D. Marx. The complexity of tree multicolorings. In *MFSC 2002*, volume 2420 of *Lecture Notes in Computer Science*, pages 532–542. Springer, 2002. 15

[110] D. Marx. Searching the k-change neighborhood for TSP is w[1]-hard. *Oper. Res. Lett.*, 36(1):31–36, 2008. 21

[111] D. Marx. Can you beat treewidth? *Theory of Computing*, 6(1):85–112, 2010. 108

[112] D. Marx and M. Pilipczuk. Everything you always wanted to know about the parameterized complexity of subgraph isomorphism (but were afraid to ask). In *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, pages 542–553, 2014. 11

[113] C. McDiarmid and B. A. Reed. Channel assignment and weighted coloring. *Networks*, 36(2):114–117, 2000. 15

[114] C. J. H. McDiarmid. On the span in channel assignment problems: bounds, computing and counting. *Discrete Mathematics*, 266(1-3):387–397, 2003. 8, 9

[115] N. Misra. personal communication, 2016. 19

[116] N. Misra, A. Nabeel, and H. Singh. On the Parameterized Complexity of Minimax Approval Voting. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015*, pages 97–105, 2015. 19, 20

[117] J. W. Moon and L. Moser. On cliques in graphs. *Israel J. Math.*, 3(1):23–28, 1965. 15

[118] J. Nederlof. Inclusion exclusion for hard problems. Master's thesis, Department of Information and Computer Science, Utrecht University, 2008. Available at http://www.win.tue.nl/~jnederlo/MScThesis.pdf. 15

[119] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991. 20

[120] M. Patrascu. Towards polynomial lower bounds for dynamic problems. In L. J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 603–610. ACM, 2010. 7

[121] A. Sebö and J. Vygen. Shorter tours by nicer ears: 7/5-approximation for the graph-tsp, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs. *Combinatorica*, 34(5):597–629, 2014. 20

[122] M. Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 2005. 26

[123] P. K. Skowron, P. Faliszewski, and J. Lang. Finding a Collective Set of Items: From Proportional Multirepresentation to Group Recommendation. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI 2015*, pages 2131–2137, 2015. 18

[124] A. Socała. Tight lower bound for the channel assignment problem. In *SODA*, 2015. 22

[125] A. Socała. Tight lower bound for the channel assignment problem. *ACM Trans. Algorithms*, 12(4):48:1–48:19, 2016. 22

[126] K. S. Sudeep and S. Vishwanathan. A technique for multicoloring triangle-free hexagonal graphs. *Discrete Math.*, 300(1-3):256–259, 2005. 15

[127] C. A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984. 26

[128] P. Traxler. The time complexity of constraint satisfaction. In M. Grohe and R. Niedermeier, editors, *IWPEC*, volume 5018 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 2008. 9

[129] K. Uchizawa, T. Aoki, T. Ito, A. Suzuki, and X. Zhou. On the Rainbow Connectivity of Graphs: Complexity and FPT Algorithms. *Algorithmica*, 67(2):161–179, 2013. 13

[130] M. Wahlström. New plain-exponential time classes for graph homomorphism. *Theory Comput. Syst.*, 49(2):273–282, 2011. 16

[131] R. Williams. A new algorithm for optimal 2-constraint satisfaction and its lications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. 7

[132] V. V. Williams and R. Williams. Subcubic equivalences between path, matrix and triangle problems. In *FOCS*, pages 645–654. IEEE Computer Society, 2010. 7, 105

Appendix A

Problem Definitions

(3,4)-CNF-SAT
Input: A k-CNF-SAT formula φ where each clause of the input formula contains exactly 3 different variables, and each variable occurs in at most 4 clauses.
Question: Is φ satisfiable?

3-Coloring
Input: A graph G.
Question: Is it possible to color the vertices of G with three colors in such a way that there are no adjacent vertices of the same color?

(a:b)-coloring
Input: $G = (V, E),\ a, b \in \mathbb{N}$
Question: Is G (a:b)-colorable?

All Pairs Shortest Paths
Input: A graph G and a function $w : E(G) \to \mathbb{Z}$ called a *weight* function.
Output: A matrix A with rows and columns indexed by the vertices of G such that for every $u, v \in V(G)$ the cell $A[u, v] \in \mathbb{Z} \cup \{+\infty\}$ contains a minimum weight of a path from u to v in G (where the weight of the path is the sum of the weights of its edges).

Carry-Less Subset Sum
Input: $n + 1$ numbers $s, a_1, \ldots, a_n$, each represented as n decimal digits. For any number x, the j-th decimal digit of x is denoted by $x^{(j)}$. It is assumed that $\sum_{i=1}^{n} a_i^{(j)} < 10$, for every $j = 1, \ldots, n$.
Question: Does there exist a sequence of indices $1 \leq i_1 < \ldots < i_k \leq n$ such that $\sum_{q=1}^{k} a_{i_q} = s$?

Channel Assignment
Input: $w : V^2 \to \mathbb{N}$, $s \in \mathbb{N}$
Question: Is there a proper assignment of span at most s?

CLOSEST STRING
Input: A multiset $S = \{s_1, \ldots, s_n\}$ of 0-1 strings of length m (also called votes), an integer d.
Question: Does there exist a string $s \in \{0,1\}^m$ such that for every $i = 1, \ldots, n$ we have $\mathcal{H}(s, s_i) \leq d$?

EDGE COLORING
Input: A graph G and a number $k \in \mathbb{N}$.
Question: Is it possible to color the edges of G with k colors in such a way that there are no adjacent edges of the same color?

EQUAL WEIGHT MATCHINGS
Input: Two complete weighted bipartite graphs $G_1 = (V_1 \cup W_1, E, w_1)$ and $G_2 = (V_2 \cup W_2, E, w_2)$ such that $|V_1| = |W_1|$ and $|V_2| = |W_2|$. The weight functions w_1, w_2 have nonnegative integer values.
Question: Are there two perfect matchings M_1 in G_1 and M_2 in G_2 such that $w_1(M_1) = w_2(M_2)$?

FAMILY INTERSECTION
Input: A function $f : [a] \times [b] \to \mathbb{N}$ and a function $g : [c] \times [d] \to \mathbb{N}$.
Question: Is $X_f \cap X_g$ nonempty?

GENERALIZED LIST T-COLORING
Input: A graph $G = (V, E)$, a function $\Lambda : V \to 2^{\mathbb{N}}$, a function $t : E \to 2^{\mathbb{N}}$ and a number $s \in \mathbb{N}$.
Question: Is there an assignment $c : V \to [s]$ such that for every vertex $v \in V$ we have $c(v) \in \Lambda(v)$ and for every edge $uv \in E$ we have $|c(u) - c(v)| \notin t(uv)$.

GENERALIZED T-COLORING
Input: A graph $G = (V, E)$, a function $t : E \to 2^{\mathbb{N}}$ and a number $s \in \mathbb{N}$.
Question: Is there an assignment $c : V \to [s]$ such that for every edge $uv \in E$ we have $|c(u) - c(v)| \notin t(uv)$.

GRAPH COLORING
Input: A graph G and a number $k \in \mathbb{N}$.
Question: Is it possible to color the vertices of G with k colors in such a way that there are no adjacent vertices of the same color?

GRAPH HOMOMORPHISM
Input: undirected graphs G, H.
Question: Is there a homomorphism from G to H, i.e., does there exist a function $h : V(G) \to V(H)$, such that for each edge $uv \in E(G)$ we have $h(u)h(v) \in E(H)$.

Integer Linear Programming
Input: An $n \times m$ integer matrix A and a vector $b \in \mathbb{Z}^m$.
Question: Does the linear program $A\mathbf{x} \leq b$ have an integral feasible solution?

k-CNF-SAT
Input: A formula φ in Conjunctive Normal Form with each clause of size at most k.
Question: Is φ satisfiable?

$k \times k$-Clique
Input: A graph G over the vertex set $V = [k] \times [k]$, i.e., V forms a grid (as a vertex set; the edge set of G is a part of the input and it can be arbitrary) with k rows and k columns.
Question: Is there in G a clique containing exactly one vertex in each row?

k-OPT Detection
Input: A TSP tour H in an edge weighted complete graph G.
Question: Does there exist an improving k-move?

Minimax Approval Voting
Input: A multiset $S = \{s_1, \ldots, s_n\}$ of 0-1 strings of length m (also called votes), two integers k and d.
Question: Does there exist a string $s \in \{0,1\}^m$ with exactly k ones such that for every $i = 1, \ldots, n$ we have $\mathcal{H}(s, s_i) \leq d$?

Negative Edge-Weighted Triangle
Input: A graph G and a function $w : E(G) \to \mathbb{Z}$ called a *weight* function.
Question: Is there a triangle (i.e. a cycle of length 3) in G such that its weight (i.e. the sum of the weights of its edges) is negative?

Rainbow k-Coloring
Input: $G = (V, E),\ k$
Question: Does G have the rainbow connection number at most k?

(r, k)-Monomial Testing
Input: An arithmetic circuit that evaluates a homogeneous polynomial $P(x_1, x_2, \ldots, x_n)$ over some field $\mathbb{F}$, a parameter r.
Question: Does P have some monomial in which every variable has individual degree not larger than r

SUBGRAPH ISOMORPHISM
Input: Undirected graphs G, H.
Question: Is G a subgraph of H, i.e., does there exist an injective function $g : V(G) \to V(H)$, such that for each edge $uv \in E(G)$ we have $g(u)g(v) \in E(H)$?

SUBSET SUM
Input: $n + 1$ numbers $s, a_1, \ldots, a_n$.
Question: Does there exist a sequence of indices $1 \leq i_1 < \ldots < i_k \leq n$ such that $\sum_{q=1}^{k} a_{i_q} = s$?

Index