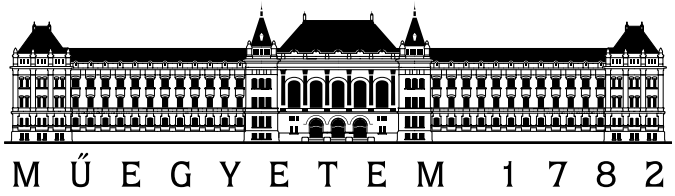
Graph Coloring with Local and Global Constraints

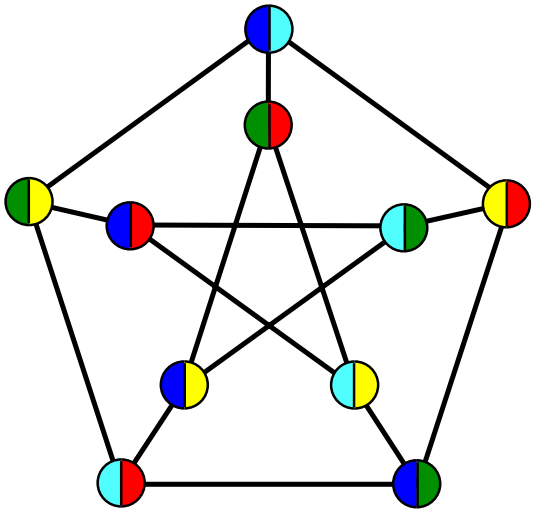


Graph Coloring with Local and Global Constraints

by

Dániel Marx

Under the supervision of
Dr. Katalin Friedl



Department of Computer Science and Information Theory
Budapest University of Technology and Economics

Budapest
2004

Alulírott Marx Dániel kijelentem, hogy ezt a doktori értekezést magam készítettem és abban csak a megadott forrásokat használtam fel. Minden olyan részt, amelyet szó szerint, vagy azonos tartalomban, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Budapest, 2004. október 12.

Marx Dániel

Az értekezés bírálatai és a védésről készült jegyzőkönyv megtekinthető a Budapesti Műszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai Karának Dékáni Hivatalában.

Contents

Acknowledgments

I'm grateful to my supervisor Katalin Friedl for her kind guidance throughout the years. She gave tons of advice on my manuscripts, teaching me how to write papers. Judit Csima and Lajos Rónyai provided valuable comments on some of my papers.

The most important years of my mathematical education were my high school years in Szent István Gimnázium, Budapest, where László Lackó was my teacher. He taught me what mathematics is, and his classes had a lasting effect on my career.

I would like to thank my colleagues at the Department of Computer Science and Information Theory of the Budapest University of Technology and Economics, and especially András Recski for providing a pleasant working environment. The financial support of the OTKA grants 30122, 44733, 42559, and 42706 of the Hungarian National Science Fund are gratefully acknowledged.

Last but not least I would like to thank the love and support of my family, without which this project would not have been possible.

CHAPTER 1

Introduction

Color possesses me. I don't have to pursue it.
It will possess me always, I know it.
That is the meaning of this happy hour:
Color and I are one. I am a painter.
Paul Klee (1879–1940)

The birth of graph theory is usually attributed to Leonhard Euler's solution of the Königsberg Bridge problem. The citizens of Königsberg (now Kaliningrad, Russia) asked the famous mathematician whether it is possible to visit all the bridges of the city in such a way that we go through every bridge exactly once. Euler observed that if such a walk exists, then there can be at most two land masses (the islands and the two banks of the river Pregel) that have odd number of bridges. There were more than two such islands in Königsberg (see Figure 1.1), hence he concluded that it is not possible to have a walk that visits each bridge exactly once. Possibly this negative answer made the citizens disappointed, but the argument opened a new chapter in mathematics. In order to answer the question, Euler reasoned about objects (land masses in this case) and connections between objects (bridges). This is precisely the notion of graph, hence graph theory was born.

Graph theory opened a treasure trove of deep questions and results. There seems to be an unstoppable flow of interesting questions about graphs. Some of these questions were investigated because they appear to be very fundamental and natural (in the mathematical sense), or follow naturally from earlier results. Moreover, this new paradigm of "objects" and "connections" turned out to be a very powerful tool in modeling a wide range of real-life problems. For example, the German physicist Kirchhoff analyzed electrical circuits using graphs. Roads, railways, and other transportation networks can be described as graphs. More recently, computer networks and the internet offer a particularly good example where graph-theoretic concepts and methods can be used successfully to solve real-life engineering problems. But graphs are useful not only in situations involving network-like physical structures, they can be used to model more abstract problems. For example, graphs hold the key to the solution in such diverse application areas as optimizing register allocations in compilers or reassembling DNA fragments.

Graph coloring is one of the earliest areas of graph theory. It was motivated by the famous Four Color Conjecture. Map makers in the nineteenth century observed that apparently every planar map can be colored using four colors in such a way that countries sharing a boundary have different colors. In the language of graph theory, the conjecture says that every planar graph can be colored with 4 colors such

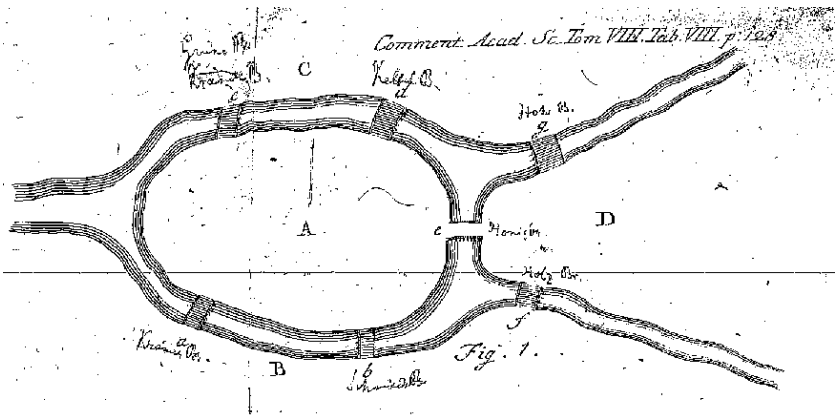


Figure 1.1: The original illustration of the Königsberg Bridge problem from [Eul36].

that neighboring vertices receive different colors. It was proved that 5 colors are always sufficient, but despite heavy efforts, the conjecture remained open for more than a century. The conjecture was settled only in 1977 (which happens to be the year when the author was born) by Appel and Haken, with a proof that requires the use of computers.

Meanwhile, coloring became a well-studied area of graph theory. As with other parts of graph theory (and with mathematics in general), the new directions were motivated both by pure theoretical interest and by possible practical applications. It turned out that besides coloring maps, there are several other situations that can be modeled by graph coloring and its variants. There are numerous examples in scheduling theory, where the assignment of resources can be reduced to a problem of assigning colors in a graph. Here we briefly recall six classical examples to give the flavor of these applications.

1. **Scheduling committees.** Assume that an organization has a number of committees, each committee requires one full day for a meeting. We have to allocate a day for each committee. However, it has to be taken into account that a person can be member in several committees, thus there are committees whose meetings cannot be scheduled to the same day. The *conflict graph* of the problem is a graph where vertices are the committees, and two vertices are connected by an edge if the corresponding two committees share at least one member. The meetings of the committees can be scheduled for k days if and only if the conflict graph can be colored with k colors. As we will see in the following, defining a conflict graph and reducing the problem to finding an appropriate coloring of the conflict graph is a widely used technique in the modeling of scheduling problems.

2. **Aircraft assignment.** We have a limited number of aircrafts, and these aircrafts have to be used to perform certain flights. Of course, an aircraft cannot be assigned to two flights at the same time. The problem can be formulated as a coloring of the conflict graph: the vertices of the conflict graph are the flights, and two flights are connected if they overlap in time. The colors correspond to the aircraft. Now the flights can be performed with k aircrafts if and only if the conflict graph is k-colorable. In the solution of the problem, we can make use of the fact that in this case the conflict graph will be an interval graph.

3. **WDM all-optical networks.** Modern networking technology is based on the large bandwidth available in the optical fiber. *Wavelength Division Multiplexing (WDM)* further increases the available bandwidth by allowing the use of multiple independent channels on the different wavelengths of the same fiber. A typical system can have 96 channels with 10Gb/sec each. However, the full

capacity of the optical fiber can be exploited only if the network devices are capable of routing these channels optically: processing 96×10Gb/sec of data electronically requires too much computing power. An all-optical network is built from optical switches that can select a specific wavelength of an incoming fiber and can transmit it without any changes on one of the outgoing fibers. Thus by appropriately configuring the optical switches, we can establish *lightpaths* between distant nodes of the network: the two ends of a lightpath are in direct optical contact through some number of fibers and switches. The switches cannot change the wavelength of a data stream, therefore the same wavelength is used on the whole length of the lightpath. When we want to configure an all-optical network, the first step is to determine how many lightpaths are required between the different nodes. Next we set a route for each lightpath. Finally, we have to assign a wavelength to each lightpath in such a way that lightpaths going through the same fiber have different wavelength. This last step can be formulated as a coloring problem by defining a conflict graph. The conflict graph has one vertex for each lightpath and two vertices are connected if the corresponding lightpaths share a fiber. The colors correspond to the wavelengths, thus by properly coloring the vertices of the conflict graph we can obtain an assignment of wavelengths such that two lightpaths that go through the same fiber do not use the same color. This coloring problem is discussed for different network topologies in several papers (see e.g., [EJ01, EJK$^+$99, BGP$^+$00, CMLF00]).

4. **Optimizing register allocations.** Processors can work fastest if they are working on data stored in the *registers.* A register is capable of storing a single value, and it can be accessed very quickly, much faster than ordinary memory. However, there is a limited number of registers, typically on the order of 10. When the compiler turns the source code of a program into machine code, it has to decide where to store the variables. Preferably, we would like to store all the variables in the registers. Since the number of registers is limited, this can be done only if we reuse the registers: two variables can be stored in the same register if they are not "live" at the same time. The register allocation problem can be modeled by graph coloring: the conflict graph has one vertex for each variable and two vertices are connected if the corresponding variables cannot be stored in the same register. The colors correspond to the registers: the variables can be stored in k registers if and only if the vertices of the conflict graph can be colored with k colors. An extensive treatment of the problem and further references can be found in [Bri92].

5. **Timetable design.** To model the timetable design problem, consider a school having a set of teachers and a set of classes. Every week, a teacher has to teach certain classes for a given number of hours. The design of the weekly timetable can be turned into a graph coloring problem as follows. Consider the bipartite graph where one bipartition class corresponds to the teachers, the other to the classes. A teacher is connected to a class if the teacher has to meet this class every week (if she has to teach the class d hours a week, then add d parallel edges). Let k be the number of available (one hour long) time slots in the week, the colors correspond to these time slots. A complete timetable for all the teachers and students exists if and only if the *edges* of the bipartite graph can be colored with k colors. The requirement that two edges with the same color cannot be incident to the same vertex corresponds to the requirement that a teacher cannot teach two classes in the same time slot, and a class cannot be taught by two teachers at the same time.

6. **Biprocessor task scheduling.** Assume we have a set of jobs, each job requires the simultaneous work of two preassigned processors for a unit amount of time. For example, the jobs can be file transfers between computers. A processor can work only on one job at the same time. We want to schedule the jobs such that every job is finished in at most k units of time. We create a graph (possible with multiple parallel edges) where each vertex corresponds to a processor and each edge corresponds to a biprocessor task. Clearly, the jobs can be finished in k units of time if and only if the edges of the graph can be colored with k colors.

In the last two examples we have to color the edges of a graph. These examples motivate the study of the edge coloring versions of the coloring problems. In fact, a large fraction of the results presented in this work concerns edge colorings. Edge coloring can be considered as a special case of vertex coloring: edge coloring of G is nothing else than the vertex coloring of the *line graph* of G. The line graph $L(G)$ of G contains one vertex v_e for each edge e of G, vertices v_e and v_f are connected if e and f have a common vertex. However, when we want to solve an edge coloring problem, it is almost always advantageous not to consider it as a general vertex coloring problem, but to make use of the special properties of edge colorings.

In most cases, however, the real-life problem does not appear in such a pure form as in the examples above, there are additional constraints that have to be satisfied. For example, a flight can be performed only by certain aircrafts, or a teacher is not available on certain days. In this work we will consider variants of the basic graph coloring problem that allow us to take into account such constraints.

The concept of *list coloring* was introduced by Vizing [Viz76] and independently by Erdős, Rubin, and Taylor [ERT80]. In a graph each vertex v has a list $L(v)$ of admissible colors, and the coloring has to satisfy the requirement that the color of vertex v has to be taken from its list $L(v)$. The combinatorial properties of list colorings have been intensively studied, several nice conjectures and results appear in the literature. Moreover, the lists allow us to model the type of requirements mentioned in the previous paragraph. If a flight can be served only by certain aircrafts, then the list of the corresponding vertex in the conflict graph contains only the colors representing these aircrafts. If a teacher can teach only during certain time slots, then only these colors appear in the list of the edges incident to the teacher's vertex. The downside of this approach is that this more general problem can be more difficult algorithmically than traditional vertex coloring. For example, list coloring is NP-hard even for bipartite graphs [HT93] and for the edges of bipartite graphs (see e.g., [Col84]).

In the *precoloring extension* problem some vertices of a graph have preassigned colors, and this precoloring has to be extended to the whole graph using the given number of colors. The problem is not equivalent to vertex coloring: it is possible that a graph is k-colorable, but there is an unfortunate precoloring that cannot be finished using k-colors. Precoloring extension can be viewed as a special case of list coloring: the list of a precolored vertex contains only a single color, while the list of a not precolored vertex contains all the available colors. Thus we can expect that in certain situations, this special case of list coloring is easier to solve than the general problem. For example, list coloring is NP-hard for split graphs [Tuz97] and for the complements of bipartite graphs [Jan97], while the precoloring extension problem is polynomial-time solvable for these classes [HT93]. However, for some other classes, such as bipartite graphs [HT93] and interval graphs [BHT92], not only list coloring is NP-hard, but the special case of precoloring extension is hard as well. For such graphs, it is worthwhile to study some restricted form of precoloring extension, in the hope of finding a polynomial time solvable case. One possible restriction is to give a bound on the number of precolored vertices, or on the number of times a color can appear in the precoloring.

List coloring and precoloring extension are examples of local constraints. We are looking for a coloring that satisfies some additional requirements, and these requirements are local in the sense that they restrict the color of individual vertices. There are other coloring problems where the constraints involve a small neighborhood of the vertices. In the *H-coloring* problem it is not sufficient that adjacent vertices have different colors: there are some prescribed pairs of colors that cannot be neighbors. For example, we might require that colors 2 and 3 cannot be neighbors, or we might require that if a vertex has color 1, then its neighbors can have only color 4 or 5, etc. One application of H-coloring is assigning frequencies to base stations in mobile networks. The base stations are the vertices and the two base stations are connected by an edge if they are "near" to each other. The colors correspond to the available frequencies. Due to interference, stations near to each other cannot receive the same frequency, which means we have a coloring problem. Moreover, to further reduce the possibility of interference problems, it is also required that the frequencies of near stations differ at least by a given number. This problem is exactly H-coloring: there are forbidden pairs of colors. *Distance constrained coloring* generalizes this setting (see e.g., [FKP01]): we

have two parameters p, q, now the requirement is that the colors of neighboring vertices differ by at least p, and the colors of vertices at distance 2 differ by at least q. This is also a local requirement: whether a color is allowed on a vertex or not depends only on the coloring of a small neighborhood of the vertex.

Coloring problems with *global constraints* are very different from the local problems. A local constraint can be violated by a small part of the coloring. For example, if a vertex receives a color not on its list in the list coloring problem, or a forbidden pair of colors appears on an edge in the H-coloring problem, then the coloring is invalid. If the constraints are global, then there are no such bad configurations: anything is acceptable, as long as the constraints on the whole graph are not violated.

Let us assume that the colors are the positive integers. In traditional vertex coloring, we are assigning positive integers to vertices and the goal is to minimize the maximum number assigned. In the *minimum sum coloring* problem the goal is to minimize the *sum* of the assigned numbers. This is a global constraint: it does not say anything about the smaller parts of the coloring, only the total sum is important.

To see an application of minimum sum coloring, we will revisit our first example above, the committee scheduling problem. If the conflict graph can be colored with k colors, then the committee meetings can be finished in k days. Solving this problem is useful if there is a deadline, and the only important thing is to finish all the meetings in at most k days. However, it is possible that, instead of minimizing the number of days, we have some other goal. For example, it seems natural to ask for a schedule that ensures that the committee meetings are finished quickly *on average* (and not that every committee is ready before a given deadline). Minimizing the average time is exactly the same as minimizing the sum of the coloring of the conflict graph. Thus minimum sum coloring can be used in scheduling problems where we want to minimize the average completion times of the jobs.

Minimizing the sum of the coloring and minimizing the number of colors can be very different. There are graphs that can be colored with few colors, but many colors are required to minimize the sum. The *chromatic strength* of a graph is the minimum number of colors that is required for a minimum sum coloring. The strength can be much larger than the chromatic number: trees are 2-colorable, but there are trees with arbitrarily large strength [KS89]. In some other respects, chromatic strengths behaves very much like the chromatic number: for example, there are analogs of Brooks's Theorem and Vizing's Theorem [MMS97, HMT00] for the chromatic strength.

If we look at the application examples sketched above, then it is natural to consider variants of the problems where we have to assign more than one color to each vertex. Maybe a committee meeting requires more than one day, or the base stations require several frequencies. The most basic setup is when we have to assign the same number m of colors to each vertex such that neighboring vertices have to receive disjoint sets of colors. As m goes to infinity, this gives us the notion of *fractional coloring*. From the point of view of applications, it is more useful to allow different color requirements on the different vertices. In the *multicoloring* (or *weighted coloring*) problem each vertex has a *demand* $x(v)$, which is the number of colors it requires. The multicoloring version of list coloring is defined similarly.

Multicoloring problems always have two versions. In a *preemptive* multicoloring problem any color set can be assigned to a vertex. On the other hand, in a *non-preemptive* problem it is required that the color set contains a consecutive interval of colors. For example, in the committee scheduling problem, if a committee needs more than one days, then it is natural to ask for a schedule where the committee meets on consecutive days. Thus the problem can be modeled by non-preemptive multicoloring. Preemptive problems arise when we have jobs that can be interrupted arbitrarily (such jobs are most commonly found in computing environments), while non-preemptive problems model jobs that cannot be interrupted once they are started (this is often the case with machine scheduling in factories). The preemptive and non-preemptive versions of the same coloring problem can be very different; the two versions might require different techniques and often the complexity is different.

The multicoloring version of minimum sum coloring has to be defined carefully. The natural generalization of minimum sum coloring is to minimize the total sum of all the colors assigned. This leads to the *Optimum Cost Chromatic Partition* problem [Jan00]. However, applications in scheduling theory suggest a different objective function. Recall that minimum sum coloring was introduced to model problems where

the average completion time has to be minimized. The vertices are the jobs, two jobs are connected if they cannot be performed at the same time for some reason. In the multicoloring version, each vertex has a demand, which is the number of time slots the jobs require. A multicoloring corresponds to a scheduling of the jobs: by assigning colors to the vertices we select the time slots when the job will be active. The highest color assigned to the vertex corresponds to the time slot when the job will be finished. Thus if we want to minimize the average completion times of the jobs, then our goal is to minimize the sum of the highest colors. In the *minimum sum multicoloring* problem the *finish time* of a vertex is defined to be the highest color assigned to it, and the goal is to minimize the sum of the finish times. This problem received considerable attention lately [BNHK$^+$00, HKP$^+$03, HK02, Kov04]. It turns out that the problem is hard in most cases, but it is better approximable than the traditional coloring problems.

Clique coloring is another example of a coloring problem with global constraints. Unlike the previous problems that demand a coloring with some additional requirements, clique coloring is less restrictive than ordinary vertex coloring. We relax the requirement that neighbors have to receive different colors, the only thing we require is that every inclusion-wise maximal (non-extendable) clique contains at least 2 colors. In this problem it is hard even to check the correctness of a coloring: it is not sufficient to check the coloring locally, the requirement has to be verified for all the (possibly exponentially many) maximal cliques. Since clique coloring is a relaxation of vertex coloring, it is possible that a graph is k-clique-colorable, but its chromatic number is much larger than k. For example, it is conjectured that every perfect graph is 3-clique-colorable (see [BGG$^+$04]).

Results

Unless noted otherwise, all the Theorems, Lemmas, etc. in this work are new results. The introduction of each chapter lists where the new results were published.

List coloring is a more general problem than ordinary vertex coloring, hence it is NP-hard in those cases where vertex coloring is NP-hard. Therefore polynomial-time algorithms can be expected only in those cases where vertex coloring is polynomial-time solvable. For trees and for graphs with bounded treewidth list coloring can be solved in polynomial time [JS97] (see Appendix A.1 for background on treewidth). But for some other classes of graphs, such as bipartite graphs [HT93], complements of bipartite graphs [Jan97], and interval graphs [BHT92], graph coloring is easy, but list coloring is NP-hard.

In Chapter 2, we study the edge coloring version of list coloring. Unlike ordinary edge coloring, list edge coloring is NP-hard for bipartite graphs (this follows from e.g., [Col84, Fia03, EP01]). We improve on this result by showing that list edge coloring remains NP-hard for planar regular bipartite graphs (Theorem 2.1.2). In further chapters, this will be the base for other complexity results. If the problem is restricted to trees, then it becomes polynomial-time solvable. Furthermore, Marcotte and Seymour [MS90] have shown that for trees the more general list edge multicoloring problem can be solved in polynomial time. However, the complexity of the problem remained open even for such simple graphs as cycles. In Section 2.3 we present a polynomial-time algorithm for a class of graphs that includes trees and odd cycles (Corollary 2.3.6). This algorithm can be extended to a randomized algorithm (with guaranteed success probability) solving the problem on graphs with few cycles (Theorem 2.3.10). On the other hand, we show in Section 2.2 that the vertex coloring version of list multicoloring is NP-complete for binary trees (Theorem 2.2.2). The results for list coloring are summarized on Table 1.1.

Chapter 3 focuses on the precoloring extension (PrExt) problem. Biró, Hujter, and Tuza [BHT92, HT93, HT96] started a systematic survey of precoloring extensions. They identified several graph classes where PrExt is polynomial time solvable and several graph classes where PrExt is NP-hard. In particular, it is shown in [BHT92] that for interval graphs PrExt is NP-hard, but the special case 1-PrExt, where every color is used at most once in the precoloring, can be solved in polynomial time. An open question in [HT96] asks whether PrExt remains NP-hard for unit interval graphs. We give a positive answer to this question in Section 3.3 (Theorem 3.3.1). To prove this result, in Section 3.2 we first show the NP-

Table 1.1: Results for the list (multi)coloring problems

Graph	Vertex coloring		Edge coloring	
	list coloring	list multicoloring	list coloring	list multicoloring
Trees	Polynomial [JS97]	NP-hard (Theorem 2.2.2)	Polynomial	Polynomial (Corollary 2.3.6)
Partial k-trees	Polynomial [JS97]	NP-hard (Theorem 2.2.2)	NP-hard (Theorem 2.1.4)	NP-hard (Theorem 2.1.4)
Cycles	Polynomial [JS97]	(Randomized) Poly. (Corollary 2.3.11)	Polynomial [JS97]	(Randomized) Poly. (Corollary 2.3.11)
Bipartite graphs	NP-hard [JS97]	NP-hard [JS97]	NP-hard [EIS76]	NP-hard [EIS76]
Planar regular bipartite graphs	NP-hard [KT94]	NP-hard [KT94]	NP-hard (Theorem 2.1.2)	NP-hard (Theorem 2.1.2)

hardness of an Eulerian disjoin path problem (Theorem 3.2.7), which answers an open question of Vygen [Vyg94]. In Section 3.1 we resolve another open question from [HT96] by showing that the polynomial-time 1-PrExt algorithm for interval graphs can be generalized to chordal graphs (Theorem 3.1.5).

Section 3.4 briefly discusses the edge coloring version of precoloring extension. The problem was shown to be NP-hard for 3-regular bipartite graphs by Fiala [Fia03]. We answer an open question from [Fia03] by showing that the problem remains NP-hard for planar 3-regular bipartite graphs (Theorem 3.4.1). Moreover, in Theorem 3.4.2 and 3.4.3 we show that edge precoloring extension is NP-hard for outerplanar graphs and for series-parallel graphs (hence for graphs with bounded treewidth), but polynomial-time solvable for bounded degree outerplanar graphs.

Chapter 4 studies the complexity of minimum sum coloring. Giaro and Kubale [GK00] show that the edge coloring version is NP-hard for bipartite graphs. In Section 4.2 we improve this result by showing that it remains NP-hard for planar bipartite graphs (Theorem 4.2.1). Moreover, we give the first inapproximability result for the edge coloring version by showing that the minimum sum edge coloring is APX-hard for bipartite graphs (Theorem 4.2.3). This result implies that it is unlikely that minimum sum coloring has a polynomial-time approximation scheme (PTAS) for bipartite graphs.

Minimum sum edge coloring is polynomial-time solvable for trees [GK00, Sal03, ZN04] by a method based on dynamic programming. If dynamic programming can be used for trees, then it is expected that the method generalizes for bounded treewidth graphs. In Section 4.3 we show that this is not the case for minimum sum edge coloring, as the problem is NP-hard already for partial 2-trees (Theorem 4.3.6).

The complexity of chromatic strength (the minimum number of colors required in a minimum sum coloring) was investigated in [Sal03]. It is shown in [Sal03] that for every $k \geq 3$, it is NP-hard to decide whether the chromatic strength is at most k. The complexity of case $k = 2$ remained an open question. In Section 4.4 we completely characterization of the complexity of chromatic strength. First we show that it is coNP-complete to decide whether the chromatic strength is at most 2 (Theorem 4.4.3). Furthermore, we show that for $k \geq 3$, the problem is not only NP-hard, but complete for the less-known complexity class Θ_2^p. Similar results are obtained for the complexity of chromatic edge strength (Corollary 4.4.12).

Minimum sum multicoloring, the subject of Chapter 5, was first studied in [BNHK$^+$99]. This problem can be difficult even for very simple graphs. For example, Kovács [Kov04] presents a highly nontrivial algorithm for paths. The algorithm in [Kov04] is pseudopolynomial, which means that the running time is a polynomial of n (the size of the graphs) and p (the maximum demand). In Section 5.1 we show that for paths there is always an optimum solution where every vertex receives a color set that is the union of $O(\log p)$ intervals (Theorem 5.1.14). The importance of this result comes from the fact that most

Table 1.2: Results for the minimum sum edge (multi)coloring problems

Graph	Minimum sum edge coloring		Minimum sum edge multicoloring	
	Algorithm	Hardness	Algorithm	Hardness
Paths	Polynomial [GK00, Sal03]		Pseudopolynomial [Kov04]	
Trees	Polynomial [GK00, Sal03]		PTAS (Theorem 5.4.15)	NP-hard (Theorem 5.2.5)
Bipartite graphs	1.796-approx. [HKS03]	APX-hard (Theorem 4.2.4)	2-approx. [BNHK$^+$00]	APX-hard (Theorem 4.2.4)
Planar graphs	PTAS [Mar04f]	NP-hard (Theorem 4.2.1)	PTAS [Mar04f]	NP-hard (Theorem 4.2.1)
Partial k-trees	PTAS [Mar04f]	NP-hard (Theorem 4.3.6)	PTAS [Mar04f]	NP-hard (Theorem 4.3.6)

approximation algorithms for minimum sum multicoloring try to find a solution where every color set is the union of only a few intervals. Therefore it can be quite useful to know that in certain cases such simple color sets are actually sufficient to find an optimum solution.

Halldórsson et al. [HKP$^+$03] presented a polynomial time approximation scheme (PTAS) for minimum sum multicoloring in trees, and they asked as an open question whether minimum sum multicoloring is NP-hard for trees. In Section 5.2 we show that the problem is indeed NP-hard, even for binary trees with polynomially bounded demand (Theorem 5.2.5). In Section 5.3 we investigate the complexity of minimum sum edge coloring for trees, it turns out that the problem is NP-hard for trees even if every demand is 1 or 2 (Theorem 5.3.1). However, somewhat surprisingly, the problem becomes polynomial-time solvable if every demand is 2, or more generally, if every demand is the same. This follows from a scaling property proved in Section 5.4.2, which implies that if every demand is the same, then the problem can be reduced to the case where every demand is 1 (Theorem 5.4.6). In Section 5.4, a polynomial-time approximation scheme is given for minimum sum edge coloring of trees: we show that for every $\epsilon > 0$ there is a polynomial-time algorithm that finds a coloring with sum at most $1 + \epsilon$ times the minimum (Theorem 5.4.15). In [Mar04f] this result is generalized to planar graphs and partial k-trees. We omit here these stronger results, in Chapter 5 we describe the approximation algorithm only for trees.

Table 1.2 summarizes the results for minimum sum edge coloring and multicoloring. We can see that the positive and the negative results nicely complement each other. Minimum sum edge coloring is polynomial-time solvable for trees, but this result cannot be generalized to planar graphs or to partial k-trees. Moreover, the algorithm for trees cannot be generalized to the multicoloring version either. However, minimum sum multicoloring for trees admits a PTAS, which can be generalized to partial k-trees and planar graphs, but cannot be generalized to bipartite graphs.

In Chapter 6 we investigate the complexity of clique coloring. It is coNP-complete to check whether a coloring is a proper 2-clique-coloring [BGG$^+$04], and it is NP-hard to check whether a perfect graph is 2-clique-colorable [KT02]. However, in Theorem 6.2.1 we show that clique coloring is even harder than that: it is not only NP-hard, but it is Σ_2^p-complete to decide whether a graph is 2-clique-colorable (see Section 6.1 and Appendix A.3 for the definition of Σ_2^p-completeness). We also consider two concepts related to clique coloring: clique choosability and hereditary clique coloring. These concepts give rise to problems with complexity even higher in the polynomial hierarchy: we show that the corresponding problems are Π_3^p-complete (Theorem 6.3.1 and Theorem 6.4.3).

In Chapter 7 we discusses some open questions related to the above results. Appendix A gives some background on treewidth, approximation, and oracles.

CHAPTER 2

List coloring

People can have the Model T in any color—so long as it's black.
Henry Ford (1863–1947)

The concept of list coloring was introduced independently by Vizing [Viz76] and by Erdős, Rubin, and Taylor [ERT80]. Given a graph $G(V, E)$, a *list assignment* L is a function that assigns to each vertex $v \in V$ a set of admissible colors $L(v)$. The list assignment is called a *k-assignment* if $|L(v)| = k$ for every $v \in V$. Graph G is *L-colorable* if there is a coloring ψ of the vertices such that $\psi(v) \in L(v)$, and $\psi(u) \neq \psi(v)$ whenever u and v are neighbors.

Much of the research done on list coloring concerns the notion of choosability. A graph G is *k-choosable* if it has an L-coloring for every k-assignment L. The *list chromatic number* of a graph is k if the graph is k-choosable but not $(k-1)$-choosable. It is obvious that the list chromatic number cannot be smaller than the chromatic number. On the other hand, there can be an arbitrarily large gap between the two parameters: for example, there are bipartite graphs with arbitrarily large list chromatic number [ERT80]. There are several deep results and conjectures in the literature on the combinatorial properties of the list chromatic number. However, here we approach list coloring from the algorithmic and complexity theoretic point of view. Choosability will be considered only in Section 6, in the context of clique colorings.

List coloring, being a generalization of vertex coloring, is NP-complete on every class of graphs where vertex coloring is NP-complete. Furthermore, there are cases where vertex coloring is easy, but list coloring is hard. For example, list coloring is NP-hard for bipartite graphs [HT93], complements of bipartite graphs [Jan97], and interval graphs [BHT92], while there are efficient coloring algorithms for these classes of perfect graphs. There are very few cases where list coloring is polynomial time solvable: it can be solved in linear time for trees, and more generally, for partial k-trees [JS97]. Moreover, if every list contains at most 2 colors, then the problem can be solved in linear time by a reduction to 2SAT.

In this chapter we consider the edge coloring version of list coloring. Ordinary edge coloring is NP-hard [Hol81, LG83], but can be solved in polynomial time for bipartite graphs. In fact, Kőnig's Line Coloring Theorem states that the edges of a bipartite graph can be colored with k colors if and only if the maximum degree is at most k. On the other hand, as observed in [Kub93], it follows from an old result of Even, Itai, and Shamir [EIS76] on the timetable problem that list edge coloring is NP-hard for bipartite graphs with maximum degree 3. In Section 2.1.1 we strengthen this result by showing that the problem remains NP-hard for planar, 3-regular bipartite graphs.

In [FZN03, FIZN03, JMT99, Wu00] the list edge coloring problem is considered for series-parallel graphs, sufficient conditions are given for some special cases. In Section 2.1.2 we investigate the computational complexity of the problem. An easy argument shows that list edge coloring can be solved in polynomial time for bounded degree series-parallel graphs. However, we prove that the problem becomes NP-hard for series-parallel graphs if the maximum degree can be arbitrary. The same results hold for outerplanar graphs as well.

Using a method that combines dynamic programming and matching, list edge coloring can be solved in polynomial time for trees [Tuz97]. However, unlike in the vertex coloring case, this approach cannot be generalized to partial k-trees: outerplanar and series-parallel graphs have treewidth at most 2, thus the results mentioned in the previous paragraph show that list edge coloring is NP-complete for partial 2-trees (see Appendix A.1 for the definition of treewidth and partial k-trees). This is somewhat surprising, since there are very few problems that are polynomial time solvable for trees but NP-hard for partial 2-trees. Usually it is expected that if a dynamic programming approach works for trees, then it can be generalized to partial k-trees. Another recent example where the problem is easy for trees but NP-hard for partial 2-trees is the edge disjoint paths problem [NVZ01].

In the *list multicoloring* problem each vertex v has a demand $x(v)$, and we have to assign v a subset $\Psi(v)$ of $L(v)$ that has size $x(v)$. Of course, the color sets assigned to neighboring vertices have to be disjoint. How does the complexity of the problem changes if we move from list coloring to list multicoloring? Is it possible to generalize the polynomial-time solvable cases of list coloring to the multicoloring problem? In Section 2.2 we show that list multicoloring is NP-hard for binary trees, thus the dynamic programming method for trees and for partial k-trees cannot be generalized to the multicoloring case. On the other hand, Marcotte and Seymour [MS90] proved a good characterization theorem for list edge multicoloring of trees. The proof can be turned into a polynomial-time algorithm for the list edge multicoloring of trees. In Section 2.3, we give a polynomial-time algorithm for a slightly more general class of graphs, which includes odd cycles, for example. With some further work, the algorithm can be extended to a randomized polynomial-time algorithm that works for graphs that have only a constant number of cycles. Randomized algorithm here means that the algorithm uses random numbers, and depending on the random numbers, there is a small constant probability that the result is wrong. However, this probability can be made arbitrarily small by repeating the algorithm multiple times.

Section 2.1 will appear as part of [Mar04i]. The results in Section 2.2 are taken from [Mar02]. Section 2.3 contains the results of [Mar03a] and in [Mar04e].

2.1 List edge coloring planar graphs

In this section we prove that list edge coloring is NP-complete restricted to various classes of planar graphs. Formally, we study the following problem:

List edge coloring

Input: A graph $G(V, E)$, a set of colors C and a *color list* L: $E \to 2^C$ for each edge.

Question: Is there an edge coloring ψ: $E \to C$ such that

- $\psi(e) \in L(e)$ for every $e \in E$, and
- $\psi(e_1) \neq \psi(e_2)$ if e_1 and e_2 are incident to the same vertex in G?

Section 2.1.1 shows that the problem is NP-complete for planar 3-regular bipartite graphs. Section 2.1.2 shows that the problem is NP-complete for outerplanar and series-parallel graphs.

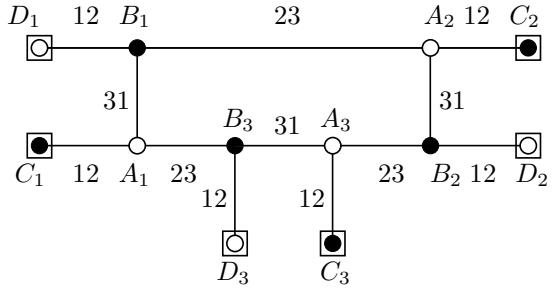


Figure 2.1: The variable-setting gadget. The two numbers on each edge show the list of available colors.

2.1.1 Planar bipartite graphs

The NP-completeness of list edge coloring bipartite graphs (with degrees at most 3) follows from [EIS76, Fia03, EP01]. We strengthen this result in two ways: here the bipartite graph is restricted to be planar and 3-regular. The main difficulty in the proof is that we need gadgets fulfilling both requirements.

The proof is by reduction from the 1-in-3 Satisfiablity problem, which is the following: given a formula in conjunctive normal form, every clause contains exactly 3 literals, decide if a variable assignment exists such that *exactly one* literal is true in every clause. The 1-in-3 Satisfiability problem remains NP-complete even with the following restrictions:

Theorem 2.1.1 ([MR01]). *1-in-3 Satisfiablity is* NP*-complete even if*

- *every variable appears in exactly* 3 *clauses,*
- *there is no negation in the formula, and*
- *the underlying bipartite graph of the formula (where the vertex representing a clause is connected to the vertices representing the variables appearing in the clause) is planar.*

Theorem 2.1.2. *List edge coloring is* NP*-complete for planar* 3*-regular bipartite graphs.*

Proof. We construct variable-setting gadgets and satisfaction-testing gadgets, and connect them in such a way that the resulting graph can be colored if and only if the given formula is satisfiable (in 1-in-3 sense). If the original formula satisfies the requirements of Theorem 2.1.1, then the resulting graph is planar and 3-regular.

Figure 2.1 shows the variable-setting gadget. It is easy to verify that it has only two colorings: the coloring that assigns the first (resp. second) color of the list to each edge. Therefore in every coloring of the gadget, the pendant edges receive the same color, either 1 or 2. The coloring that assigns color 1 to the pendant edges corresponds to setting the variable to "true," and the coloring that assigns 2 to these edges corresponds to "false."

The satisfaction-testing gadget is show on Figure 2.2. We claim that it has only three colorings, the numbers in the frames are the colors assigned to the edge in the three colorings. Let ψ be a coloring of this gadget. First, let us verify that $\psi(A_iC_i) = \psi(B_iD_i)$ for $i = 1, 2, 3$ in every coloring ψ. For this purpose, it is sufficient to follow the implications of say, $\psi(A_1C_1) = 1$ and $\psi(B_1D_1) = 2$, to arrive to a contradiction. Furthermore, it can be shown that $\psi(A_1C_1) = \psi(B_1D_1) = 1$ implies that ψ is the first coloring defined on Figure 2.2. Similarly, $\psi(A_2C_2) = \psi(B_2D_2) = 1$ (resp. $\psi(A_3C_3) = \psi(B_3D_3) = 1$) implies that ψ is the second (resp. third) coloring. Thus in every coloring of the gadget, exactly one of the pairs A_iC_i and B_iD_i is colored with color 1, the others with color 2. A coloring that assigns color 1 to A_iC_i and B_iD_i corresponds to a variable assignment where the clause is satisfied by its ith literal.

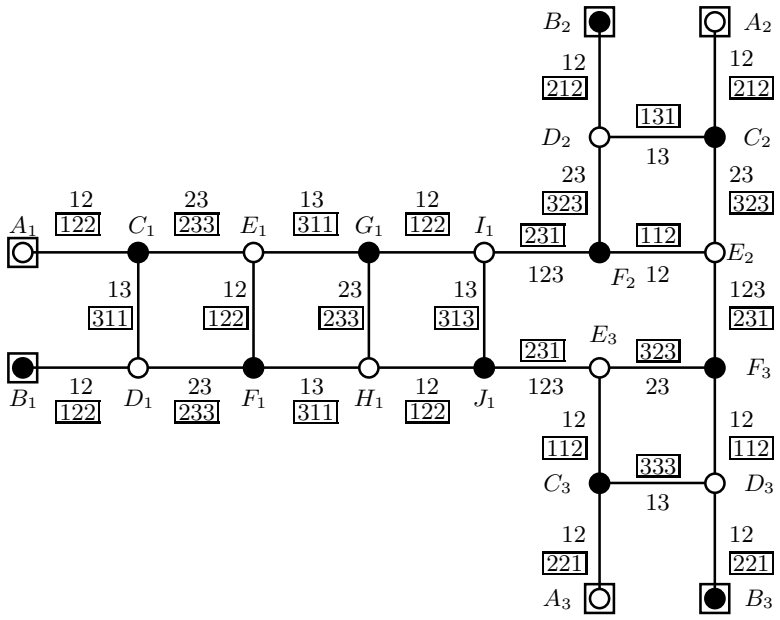


Figure 2.2: The satisfaction-testing gadget. The numbers on the edges show the list of available colors on the edge. The three numbers in each frame show the three colors assigned to the edge in the three possible colorings of the graph.

Figure 2.3 shows the overview of the construction. Take a copy of the variable-setting gadget G_x for each variable x and a satisfaction-testing gadget G_C for each clause C of the formula. If x appears in clause C, then G_x and G_C are connected by a pair of edges. Assume that x is the ith variable in clause C, and C is the jth clause where x appears, then connect G_x and G_C by identifying the edges A_jC_j and B_jD_j of G_x with A_iC_i and B_iD_i of G_C, respectively.

The resulting graph is bipartite since the gadgets are bipartite (Figure 2.1 and 2.2 show the two color classes) and when we identified two edges, we only identified vertices that belong to the same color class. Because every variable appears in exactly three clauses and every clause has exactly three literals, the resulting graph is 3-regular. The gadgets are planar, and because of the planarity of the formula, the graph can be embedded in the plane such that two pairs of edges do not cross each other. Note that in the variable-setting gadget, edge B_jD_j is in clockwise direction from the edge A_jC_j, while in the satisfaction-testing gadget B_iD_i is in counterclockwise direction from A_iC_i. Thus the two edges connecting a variable-setting gadget with a satisfaction-testing gadget do not cross each other, as shown on Figure 2.3. Therefore the resulting graph G is planar, bipartite and 3-regular.

It is clear from the construction that G has a proper list edge coloring if and only if the formula has a satisfying variable assignment (in 1-in-3 sense). Obviously, the graph can be built in polynomial time, thus we have proved that the problem is NP-complete. ■

2.1.2 Outerplanar graphs

A graph is *outerplanar* if it has a planar embedding such that all the vertices lie on the exterior face. A graph is *series-parallel* if it can be created from K_2 by repeatedly duplicating and subdividing the edges. If an outerplanar graph is 2-connected, then it is series-parallel.

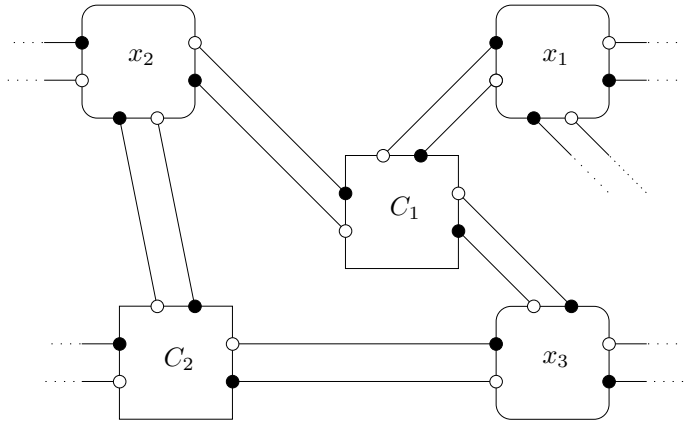


Figure 2.3: The reduction from 1-in-3 SAT to the list edge coloring problem. We connect the variable setting components and the satisfaction-testing components with pairs of edges.

List edge coloring can be solved in linear time for bounded degree outerplanar and series-parallel graphs as follows. It is well-known that these graphs have treewidth at most 2. We show that if both the treewidth and the maximum degree of a graph are bounded by a constant, then list edge coloring can be solved in linear time. If a graph has treewidth at most w and maximum degree d, then the treewidth of its line graph is at most $(w+1)d-1$ (see [Bod98, Lemma 32]). List edge coloring is the same as list coloring in the line graph. By [JS97], list coloring can be solved in $O(|V|^{k+2})$ time if the treewidth of the graph is at most k. Thus Theorem 2.1.2 cannot be strengthened to outerplanar graphs. However, if we drop 3-regularity, then the problem remains NP-complete for bipartite outerplanar graphs. In the proof we use the following version of the satisfiability problem:

Proposition 2.1.3. 3SAT *remains* NP*-complete even if every variable occurs exactly twice positively and exactly twice negated, and every clause contains exactly three literals.*

Proof. It is well-known that 3SAT remains NP-complete if every variable occurs exactly twice positively, exactly once negated, and every clause contains two or three literals (cf. [Pap94]). Let us assume that the number of variables is even, if not, then duplicate every variable and every clause. Let x_1, x_2, ..., x_n be the variables of ϕ. We add $n/2$ new variables y_1, y_2, ..., $y_{\frac{n}{2}}$ and n new clauses $(\bar{x}_1 \vee y_1 \vee \bar{y}_1)$, $(\bar{x}_2 \vee y_1 \vee \bar{y}_1)$, $(\bar{x}_3 \vee y_2 \vee \bar{y}_2)$, $(\bar{x}_4 \vee y_2 \vee \bar{y}_2)$, ..., $(\bar{x}_{n-1} \vee y_{\frac{n}{2}} \vee \bar{y}_{\frac{n}{2}})$, $(\bar{x}_n \vee y_{\frac{n}{2}} \vee \bar{y}_{\frac{n}{2}})$ to the formula. Now every variable occurs exactly twice positively and twice negated. These new clauses are satisfied in every variable assignment, hence the new formula is satisfiable if and only if the original is satisfiable. Furthermore, if there is a clause $(x \vee y)$ containing only two literals, then add a new variable z, and replace this clause with $(x \vee z \vee z) \wedge (\bar{z} \vee \bar{z} \vee y)$. It is easy to see that this transformation does not change the satisfiability of the formula. ■

Theorem 2.1.4. *List edge coloring is* NP*-complete for bipartite outerplanar graphs.*

Proof. The proof is by reduction from 3SAT. Given a formula ϕ in conjunctive normal form with n variables and m clauses, we construct an instance of the list edge coloring problem in such a way that the graph can be colored if and only if ϕ is satisfiable. By Prop. 2.1.3, we can assume that every variable occurs exactly twice positively and exactly twice negated in ϕ, and every clause contains exactly three literals.

The set of colors C contains $4n$ colors: there is one color corresponding to each occurence of a variable. For $1 \leq i \leq n$, color $4i$ corresponds to the first positive occurence of x_i, color $4i-1$ corresponds to the

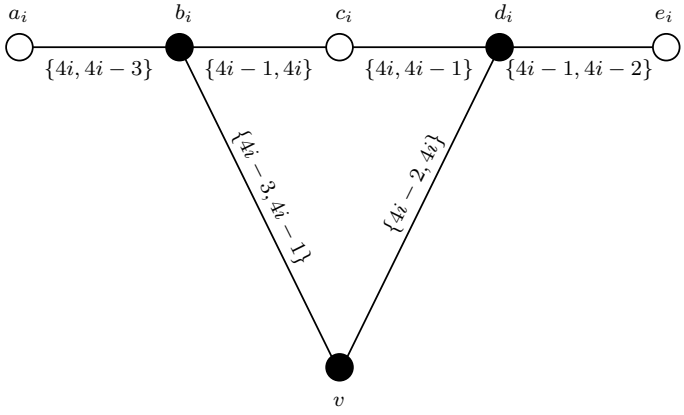


Figure 2.4: The edges corresponding to variable x_i of the formula.

second positive occurence of x_i, color $4i-2$ corresponds to the first negated occurence of x_i, and color $4i-3$ corresponds to the second negated occurence of x_i.

We construct the construc the list edge coloring instance as follows. Let v be a vertex. For every $1 \leq i \leq n$, attach 5 new vertices a_i, b_i, c_i, d_i, e_i to v as show on Figure 2.4. These edges correspond to variable x_i. The lists are as shown on the figure. For every $1 \leq j \leq m$, a new vertex u_j is attached to v. The list of vu_j contains three colors: the colors that correspond to the three literals in the jth clause of ϕ.

Given a satisfying assignment of ϕ, we construct a coloring ψ of G. If variable x_i is true, then the 6 edges corresponding to x_i receive the first color from their list. If x_i is false, then the edges receive the second color from the lists. Notice that with this coloring the colors used on edges incident to v are exactly those colors that correspond to the *false* literals in the assignment. By assumption, the assignment satisfies every clause of the formula, every clause contains at least one true literal, hence the list of every edge vu_j contains at least one color not used on v. Therefore we can extend ψ to the edges vu_j, and the graph can be colored.

To prove the other direction assume that there is a coloring ψ of G. We show that for every $1 \leq i \leq n$, either

- $\psi(vb_i) = 4i-3$ and $\psi(vd_i) = 4i-2$, or
- $\psi(vb_i) = 4i-1$ and $\psi(vd_i) = 4i$ hold.

If $\psi(vb_i) = 4i-3$, then $\psi(a_ib_i) = 4i \Rightarrow \psi(b_ic_i) = 4i-1 \Rightarrow \psi(c_id_i) = 4i \Rightarrow \psi(vd_i) = 4i-2$. A similar argument shows that if $\psi(vb_i) = 4i-1$, then $\psi(vd_i) = 4i$ follows. We set variable x_i to true in the first case, and to false in the second case. This yields a satisfying variable assignment of ϕ: if the jth clause is not satisfied, then all of its literals are false, which implies that the colors corresponding to these literals appear at v. However, this means that the three colors in the list of vu_j are already used at v, therefore edge vu_j cannot receive any color, contradicting the assumption that ψ is a list coloring of G. ■

Every outerplanar graph can be extended to a series-parallel graph by adding edges. For each new edge, let its list contain a new color that does not appear elsewhere. This does not change the solvability of the problem. Thus we obtain that list edge multicoloring is NP-complete for series-parallel graphs:[1]

Corollary 2.1.5. *List edge coloring is* NP*-complete for series-parallel graphs.* ■

[1]The astute reader will notice that Corollary 2.1.5 is a consequence of the stronger result Theorem 3.4.3. The argument to prove Corollary 2.1.5 was presented here only for the sake of completeness.

2.2 List multicoloring of trees

In this section we study the list multicoloring problem, which is defined as follows:

List Multicoloring

Input: A graph $G(V, E)$, a *demand function* x: $V \to \mathbb{N}$, a set of colors C and a *color list* L: $V \to 2^C$ for each vertex.

Question: Is there a *multicoloring* Ψ: $V \to 2^C$ such that

- $\Psi(v) \subseteq L(v)$ for every $v \in V$,
- $\Psi(u) \cap \Psi(v) = \emptyset$ if u and v are neighbors in G, and
- $|\Psi(v)| = x(v)$ for every $v \in V$?

We show that list multicoloring is NP-hard for trees, even if the degree of every node is at most three. But before that we briefly discuss how list coloring can be solved for trees, and why the algorithm for list coloring cannot be generalized to list multicoloring. We present two algorithms from [JS97, Tuz97], but neither of them works in the case of multicoloring.

Algorithm 1. A list coloring of a tree can be found the following way. First, if there is a vertex v whose list contains only one color c, then this vertex can be removed from the problem: assign color c to v, delete c from the lists of the neighbors of v, and remove v from the graph. Therefore it can be assumed that every list contains at least two colors. We claim that the tree can be colored with the lists. We prove this by induction on the number of vertices. Let v be a leaf of the tree. Delete v from the tree, the remaining tree can be colored by the induction hypothesis. This coloring assigns some color c to the neighbor of v. Since the list of v contains at least two colors, thus we can assign to v a color different from v, which extends the coloring to the whole tree, completing the induction.

Algorithm 2. Another, slightly more complicated possibility is to use dynamic programming for the subtrees of the tree. This approach has the advantage that it readily generalizes to partial k-trees. Assume that the tree is rooted. Let T_v be the subtree rooted at node v. The set $L'(v)$ consists of those colors c for which there is a list coloring of T_v such that node v receives color c. Clearly, the tree has a list coloring if and only if $L'(r)$ is not empty for the root r. We can determine the sets $L'(v)$ in a bottom-up fashion. If v is a leaf, then trivially $L'(v) = L(v)$. Now assume that $v_1, v_2, \ldots, v_t$ are the children of v, and $L'(v_1), \ldots, L'(v_t)$ are already determined. A color $c \in L(v)$ is in $L'(v)$ if each of the sets $L'(v_1), \ldots, L'(v_t)$ contain a color different from c. In this case the subtrees $T_{v_1}, \ldots, T_{v_t}$ can be colored such that their roots do not have color c, thus v can receive this color. This way we can determine the sets $L'(v)$ one by one, and when the root is reached, it can be checked whether $L'(r)$ is empty or not.

This approach breaks down if the demands of the vertices can be greater than one. The problem is that we would have to determine all the possible color sets that can appear on v in a coloring of T_v. However, if the demand of v is large, then there could be exponentially many such color sets, thus it would be too much work to enumerate all of them. The following theorem shows that list multicoloring is NP-complete for binary trees, thus it seems that there is no way to get around this problem.

Theorem 2.2.1. *The list multicoloring problem is* NP*-complete for trees.*

Proof. The reduction is from the maximum independent set problem. For every graph $G(V, E)$ and integer k, we will construct a tree T (in fact, a star), a demand function, and a color list for each node, such that the tree can be colored with the lists if and only if G has an independent set of size k. The colors

correspond to the vertices of G, the leaves of the star correspond to the edges of G. The construction will ensure that the colors given to the central node correspond to an independent set in G.

Let $e_1, e_2, \ldots, e_m$ be the edges of G and denote by $u_{i,1}$ and $u_{i,2}$ the two end vertices of edge e_i. The tree T is a star with a central node v and m leaves $v_1, \ldots, v_m$. The demand of v is k and the demand of every leaf is 1. The set of colors C corresponds to the vertex set V. The color list of the central node v is the set C, the list of node v_i is the set $\{u_{i,1}, u_{i,2}\}$.

Assume that there is a proper list coloring of T. It assigns k colors to v. The corresponding set of k vertices will be independent in G: at least one end vertex of each edge e_i is not contained in this set since node v_i must be colored with either $u_{i,1}$ or $u_{i,2}$. On the other hand, if there is an independent set of size k in G, then we can assign this k colors to v and extend the coloring to the nodes v_i: either $u_{i,1}$ or $u_{i,2}$ is not contained in the independent set, thus it can be assigned to v_i. ■

In order to prove that the problem is NP-complete for binary trees, we use a "color copying" trick to split a high-degree node into several nodes:

Theorem 2.2.2. *The list multicoloring problem remains* NP*-complete restricted to binary trees.*

Proof. The proof is essentially the same as in Theorem 2.2.1, but the degree m central node of the star is replaced by a path $v'_1, v'_2, \ldots, v'_{2m-1}$ of $2m-1$ nodes. The m neighbors of v are connected to the m nodes $v'_1, v'_3, \ldots, v'_{2m-1}$ one by one. The list of every node v'_i is C, the demands are $x(v'_{2i+1}) = k$ and $x(v'_{2i}) = |C| - k$. It is easy to see that in every proper multicoloring of the tree, the nodes $v'_1, v'_3, \ldots, v'_{2m-1}$ receive the same set of k colors. Furthermore, as in the previous proof, this set corresponds to an independent set in G. ■

We remark here that list multicoloring is polynomial-time solvable for paths [KG02]. Therefore Theorem 2.2.2 cannot be strengthened to trees with maximum degree 2.

2.3 Graphs with few cycles

In this section we consider the edge coloring version of list multicoloring:

List edge multicoloring

Input: A graph $G(V, E)$, a *demand* function x: $E \to \mathbb{N}$ and a *color list* L: $E \to 2^{\mathbb{N}}$ for each edge.

Question: Is there a *multicoloring* Ψ: $E \to 2^{\mathbb{N}}$ such that

- $\Psi(e) \subseteq L(e)$ for all $e \in E$,
- $\Psi(e_1) \cap \Psi(e_2) = \emptyset$ if e_1 and e_2 are incident to the same vertex in G and
- $|\Psi(e)| = x(e)$ for all $e \in E$?

In this section "coloring" will always mean list edge multicoloring. Marcotte and Seymour gave a good characterization for this problem in the special case when G is a tree. Denote by $E_c \subseteq E$ the set of those edges whose lists contain the color c, and for all $X \subseteq E$, let $\nu_c(X) = \nu(X \cap E_c)$ be the maximum number of independent edges in X whose lists contain c.

Theorem 2.3.1 (Marcotte and Seymour, 1990, [MS90]). *Let G be a tree. The list edge multicoloring problem has a solution if and only if for every $X \subseteq E$ we have*

$$\sum_{c \in \mathbb{N}} \nu_c(X) \geq \sum_{e \in X} x(e). \tag{2.1}$$

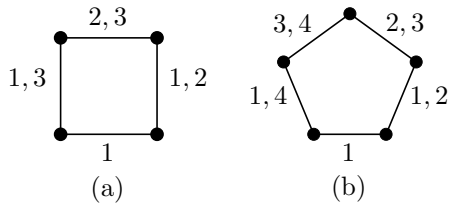


Figure 2.5: Theorem 2.3.1 does not hold for (a) even cycles and (b) odd cycles. Every edge has demand 1, the numbers on an edge are the colors contained in the list of the edge.

The necessity of the condition is obvious for any graph, since color c can be used at most $\nu_c(X)$ times in X, thus at most $\sum_{c\in\mathbb{N}} \nu_c(X)$ colors can be assigned to the edges in X.

This theorem, in general, does not remain valid on cycles. Figure 2.5 shows two uncolorable instances of the problem. The reader can easily verify that inequality (2.1) holds for every subset X of the edges, but the graphs are not colorable.

The proof of Theorem 2.3.1 is based on the total unimodularity of a network matrix, thus, using standard techniques, the proof can be turned into a polynomial time algorithm by reducing the task to a maximum flow problem. Here we present another polynomial time algorithm, which solves the problem for a slightly more general class of graphs, including trees and odd cycles. Moreover, with some further modifications, it can be turned into a randomized polynomial time algorithm working on an even more general class of graphs, which also includes even cycles.

In Section 2.3.1, a polynomial time solvable variant of the list edge multicoloring problem is introduced. This gives us a polynomial time solution of the original list edge multicoloring problem in some special cases (e.g., trees, odd cycles). Section 2.3.2 presents a modified randomized algorithm for list edge multicoloring arbitrary connected graphs having at most $|V| + O(1)$ edges.

2.3.1 A polynomial case

We introduce a new variant of list edge multicoloring. The requirement that edge e has to receive $x(e)$ colors is replaced by the requirement that the edges incident to v have to receive $y(v)$ colors in total. It turns out that in certain cases list edge multicoloring can be reduced to this new problem. Moreover, this problem can be solved in polynomial time for *any* graph (Theorem 2.3.4).

List edge multicoloring with demand on the vertices

Input: A graph $G(V, E)$, a demand function y: $V \to \mathbb{N}$ and a color list L: $E \to 2^{\mathbb{N}}$ for each edge

Question: Is there a *multicoloring* Ψ: $E \to 2^{\mathbb{N}}$ such that

- $\Psi(e) \subseteq L(e)$ for all $e \in E$,
- $\Psi(e_1) \cap \Psi(e_2) = \emptyset$ if e_1 and e_2 are incident to the same vertex in G, and
- $\sum_{e\ni v} |\Psi(e)| = y(v)$ for all $v \in V$?

The incidence matrix $\mathbf{B}$ of an undirected simple graph $G(V, E)$ has $|V|$ rows and $|E|$ columns, and for every $v \in V$ and $e \in E$, the element in row v and column e is 1 if e is incident to v and 0 otherwise. It will be convenient to think of the demand function x: $E \to \mathbb{N}$ in the list edge multicoloring problem as a vector $\mathbf{x}$ with $|E|$ (integer) components. Similarly, the demand function y: $V \to \mathbb{N}$ corresponds to a vector $\mathbf{y}$ with $|V|$ components. From now on, the demand function and its vector will be used

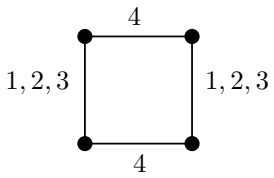


Figure 2.6: The list edge multicoloring with $x(e) \equiv 2$ has no solution, but with $y(v) \equiv 4$ there is a coloring valid for the vertices.

interchangeably. A coloring Ψ is *valid for the edges* if $|\Psi(e)| = x(e)$ for every edge e. It is *valid for the vertices* if $\sum_{e \ni v} |\Psi(e)| = y(v)$ for every vertex v. When using these terms, the demand functions $x(e)$ and $y(v)$ will be clear from the context.

Let $\mathbf{x}$ be an arbitrary demand function on the edges of G, and define $\mathbf{y} = \mathbf{Bx}$. Let L be an arbitrary list assignment. If the list edge multicoloring problem with demand $\mathbf{x}$ has a solution, then list edge multicoloring with demand $\mathbf{y}$ on the vertices has a solution as well. To see this, observe that any coloring Ψ valid for the edges is also valid for the vertices: $\sum_{e \ni v} |\Psi(e)| = \sum_{e \ni v} x(e)$ equals the component of $\mathbf{Bx} = \mathbf{y}$ corresponding to v, as required. The converse is not necessarily true: a coloring Ψ valid for the vertices is not always valid for the edges. In fact, as shown on Figure 2.6, it is possible that that there is a coloring satisfying the demand $\mathbf{y}$ on the vertices, but there is no coloring valid for the edges.

However, there is an important special case where every coloring valid for the vertices is also valid for the edges. We say that a graph $G(V, E)$ has *full edge rank* if the rank of $\mathbf{B}$ is $|E|$, that is, the characteristic vectors of the edges of G are linearly independent over $\mathbb{Q}$.

Lemma 2.3.2. *Let $\mathbf{x}$ be an arbitrary demand function on the edges of G, and let $\mathbf{y} = \mathbf{Bx}$, where $\mathbf{B}$ is the incidence matrix of G. If G has full edge rank, then for every list assignment L, any coloring valid for the vertices is also valid for the edges.*

Proof. Let Ψ be a coloring valid for the vertices. Define $x'(e) = |\Psi(e)|$, and let $\mathbf{x}'$ be the corresponding vector with $|E|$ components. Since $\sum_{e \ni v} |\Psi(e)| = y(v)$ holds, vector $\mathbf{x}'$ satisfies $\mathbf{Bx}' = \mathbf{y}$. However, the columns of $\mathbf{B}$ are linearly independent, thus $\mathbf{x}$ is the unique vector satisfying $\mathbf{Bx} = \mathbf{y}$. Hence $\mathbf{x} = \mathbf{x}'$, and $|\Psi(e)| = x(e)$ follows. ■

It is well-known that every tree and odd cycle has full edge rank. From the definition it is clear that a graph has full edge rank if and only if all of its connected components have full edge rank. It is not difficult to characterize those connected graphs that have full edge rank. For completeness we include a proof here:

Lemma 2.3.3. *A connected simple graph $G(V, E)$ has full edge rank if and only if it does not contain even cycles and it has at most one odd cycle.*

Proof. We prove the lemma by induction on the number of vertices. Assume first that G has a degree 1 vertex v, let e be the edge incident to v. In the incidence matrix $\mathbf{B}$ of G, there is only one non-zero element in row v, thus deleting row v and column e decreases the rank by exactly one. The resulting matrix is the incidence matrix of $G - v$, thus G has full edge rank if and only if $G - v$ has full edge rank. Deleting a degree 1 vertex does not change any of the cycles.

Next assume that every vertex has degree at least 2. If G has full edge rank, then it has at most $|V|$ edges, thus the degree of every vertex is exactly 2 and G is a cycle. A cycle has full edge rank if and only if it is odd, therefore the lemma follows from the induction hypothesis. ■

We show that list edge multicoloring with demand on the vertices can be solved in polynomial time. Together with Lemma 2.3.2, this implies that the list edge multicoloring problem also can be solved in polynomial time if the graph has full edge rank.

Theorem 2.3.4. *For every simple graph G, list edge multicoloring with demand on the vertices can be solved in polynomial time.*

Proof. Let $C = |\bigcup_{e\in E} L(e)|$ be the total number of different colors appearing in the lists. It can be assumed that $L(e) \subseteq \{1, 2, \ldots, C\}$ for every $e \in E$. We construct a graph $G'(U, F)$ as follows (see Fig. 2.7). For every $v \in V$, there are $2C - y(v)$ vertices $v_1, v_2, \ldots, v_C, v'_1, v'_2, \ldots, v'_{C-y(v)}$ corresponding to v in G'. If $uv \in E$ and $c \in L(uv)$, then there is an edge $u_c v_c$ in G'. Furthermore, for every $v \in V$, the vertices $v'_1, v'_2, \ldots, v'_{C-y(v)}$ are connected to every vertex $v_1, v_2, \ldots, v_C$. This completes the description of the graph G'.

We show that G' has a perfect matching if and only if there is a coloring of G valid for the edges. This implies the theorem, since there are polynomial time algorithms for finding perfect matchings in arbitrary graphs (cf. [LP86, MV80]).

First assume that Ψ is a coloring valid for the vertices. If $c \in \Psi(uv) \subseteq L(uv)$, then include the edge $u_c v_c$ into the set M'. Since $\Psi(uv)$ is a proper coloring, every vertex is covered at most once by the edges in M'. Furthermore, from the C vertices $v_1, v_2, \ldots, v_C$, exactly $y(v)$ is covered by M'. The remaining $C - y(v)$ vertices can be matched with the $C - y(v)$ vertices $v'_1, v'_2, \ldots, v'_{C-y(v)}$. Thus we can extend M' to a perfect matching M of G'.

On the other hand, assume that $M \subseteq F$ is a perfect matching of G'. Let $c \in \Psi(uv)$ if and only if $u_c v_c \in M$. Clearly $\Psi(uv) \subseteq L(uv)$, since $u_c v_c \in M \subseteq F$ implies $c \in L(uv)$. Furthermore, $\Psi(uv) \cap \Psi(uw) = \emptyset$, since $c \in \Psi(uv)$ and $c \in \Psi(uw)$ would imply $u_c v_c \in M$ and $u_c w_c \in M$, which is impossible. What remains to be shown is that $\sum_{e\ni v} |\Psi(e)| = y(v)$. From the C vertices $v_1, v_2, \ldots, v_C$ there are exactly $C - y(v)$ that are matched with the vertices $v'_1, v'_2, \ldots, v'_{C-y(v)}$. Thus the total size of the sets $\Psi(e)$ on the edges incident to v is exactly $y(v)$. ■

Corollary 2.3.5. *If G has full edge rank, then the list edge multicoloring problem can be solved in polynomial time.* ■

Corollary 2.3.6. *The list edge multicoloring problem can be solved in polynomial time for trees and odd cycles.* ■

The algorithm of Micali and Vazirani [MV80] can be used to find a perfect matching in $O(p^{\frac{1}{2}}q)$ time if the graph has p vertices and q edges. The constructed graph G' in Theorem 2.3.4 has $O(C|V|)$ vertices and $O(C|E|+C^2|V|)$ edges, therefore list edge multicoloring with demands on the vertices can be solved in $O(C^{\frac{3}{2}}|V|^{\frac{1}{2}}(|E|+C|V|))$ time. This leads to an $O(C^{5/2}|V|^{3/2})$ time algorithm for the list edge multicoloring of trees and odd cycles.

We note that if G is bipartite, then the constructed graph G' is bipartite as well, and the bipartite matching algorithm of [HK73] can be used in Theorem 2.3.4.

Let us try to estimate the practical performance of the algorithm. First observe that the algorithm has simple structure and is easy to implement, there are no theoretically easy but practically difficult concepts involved. We have to do the following steps:

1. Construct the graph G'.

2. Find a perfect matching of G'.

3. Construct the multicoloring Ψ: if edge $u_c v_c$ is in the matching, then add color c to the set $\Psi(uv)$.

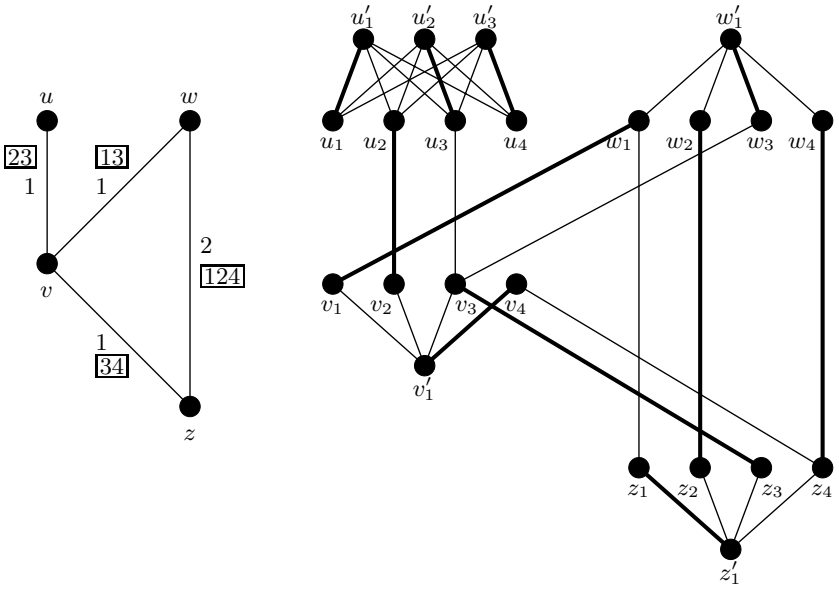


Figure 2.7: An instance of list edge multicoloring and the constructed matching problem. The number on each edge is the demand, the framed numbers are the lists. The matching shown with strong edges give a solution to list edge multicoloring.

The running time of the algorithm is dominated by the second step. Thus the running time depends mainly on the practical performance of the matching algorithm. As discussed above, we have to find a perfect matching in a graph with $O(C|V|)$ vertices and $O(C^2|V|)$ edges. However, most of the edges of the graph G' are in the complete bipartite graphs connecting vertices $v_1, \ldots, v_C$ with vertices $v'_1, \ldots, v'_{C-y(v)}$ (for some vertex v). The vertices $v'_1, \ldots, v'_{C-y(v)}$ have the same neighborhood, hence using some technical tricks (turning the problem into b-matching) we can simplify these complete bipartite graphs. Therefore the graph can be reduced to a sparse graph with $O(C|V|)$ vertices and $O(C|V|)$ edges. A computation study in 1998 [KP98] reported that matching can be solved for 50000 vertex sparse graphs in well under a minute. Thus we can expect that currently the algorithm is efficient if $C|V|$ is in the range of 500000, for example, if there are 500 colors and 1000 vertices.

2.3.2 Graphs with few cycles

In this section we try to extend the results of Section 2.3.1 to graphs that are "almost trees": to graphs that have only a small number of cycles. However, Lemma 2.3.2 is best possible:

Proposition 2.3.7. *For any graph G that does not have full edge rank, then there is a list assignment L and demand function $\mathbf{x}$ such that there is no coloring valid for the edges, but there is a coloring valid for the vertices (with $\mathbf{y} = \mathbf{B}\mathbf{x}$).*

Proof. If G does not have full edge rank, then there is a nonzero integer vector $\mathbf{z}$ with $\mathbf{B}\mathbf{z} = 0$. Since the columns of $\mathbf{B}$ are nonnegative vectors, at least one component of $\mathbf{z}$ is negative. Suppose that $z(e^*) < 0$. Let $d = \min_{e \in E} z(e)$ and let $L(e)$ be a set of $z(e) - d \geq 0$ colors such that every color in the union of the $L(e)$'s appears in only one list. By setting $x(e) \equiv -d$, it is clear that there is no coloring valid for the edges, since $|L(e^*)| < -d = x(e^*)$. On the other hand, the coloring $\Psi(e) = L(e)$ is valid for the vertices. Coloring Ψ assigns $z(e) - d$ colors to edge e. Therefore the number of colors appearing at the vertices is given by the vector $\mathbf{B}(\mathbf{z} - d \cdot \mathbf{1}) = \mathbf{B}(-d \cdot \mathbf{1}) = \mathbf{B}\mathbf{x}$, as required. ■

On the other hand, we show that if a coloring Ψ is valid for the vertices and it satisfies some additional constraints, then it is also valid for the edges.

Lemma 2.3.8. *Let $G(V, E)$ be an arbitrary graph, and let $E' \subseteq E$ be a subset of edges such that the graph $G'(V, E \setminus E')$ has full edge rank. For an arbitrary demand function $\mathbf{x}$ and list assignment L, if coloring Ψ is valid for the vertices and it satisfies $|\Psi(e)| = x(e)$ for every $e \in E'$, then Ψ is also valid for the edges.*

Proof. It can be assumed that the edges in E' correspond to the first $|E'|$ columns of $\mathbf{B}$. Therefore $\mathbf{B}$ can be written as $\mathbf{B} = (\mathbf{B}_1\ \mathbf{B}_2)$, where $\mathbf{B}_1$ has $|E'|$ columns. Similarly $\mathbf{x} = \binom{\mathbf{x}_1}{\mathbf{x}_2}$ and $\mathbf{x}_1$ has $|E'|$ components. Clearly, $\mathbf{y} = \mathbf{B}\mathbf{x} = \mathbf{B}_1\mathbf{x}_1 + \mathbf{B}_2\mathbf{x}_2$.

Let $x'(e) = |\Psi(e)|$ and let $\mathbf{x}' = \binom{\mathbf{x}'_1}{\mathbf{x}'_2}$ be the corresponding vector. Since Ψ is valid for the vertices, it follows that $\mathbf{B}\mathbf{x}' = \mathbf{B}\mathbf{x} = \mathbf{y}$, that is

$$\mathbf{B}_1\mathbf{x}'_1 + \mathbf{B}_2\mathbf{x}'_2 = \mathbf{B}_1\mathbf{x}_1 + \mathbf{B}_2\mathbf{x}_2 = \mathbf{y}.$$

Moreover, since $|\Psi(e)| = x(e)$ for every $e \in E'$, we have that $\mathbf{x}'_1 = \mathbf{x}_1$, $\mathbf{B}_1\mathbf{x}'_1 = \mathbf{B}_1\mathbf{x}_1$, and $\mathbf{B}_2\mathbf{x}'_2 = \mathbf{B}_2\mathbf{x}_2$ follows. Since $G'(V, E \setminus E')$ has full edge rank, the columns of the matrix $\mathbf{B}_2$ are linearly independent, hence $\mathbf{x}'_2 = \mathbf{x}_2$. Therefore $\mathbf{x}' = \mathbf{x}$, and Ψ is valid for the edges. ■

Since every coloring that is valid for the edges is also valid for the vertices, the list edge multicoloring problem has a solution if and only if there is a coloring valid for the vertices that satisfies the requirements in Lemma 2.3.8. We reduce the problem of finding such a coloring to a variant of the exact matching problem, which can be solved in randomized polynomial time. Before presenting the reduction (which is essentially the same as in the proof of Theorem 2.3.4), we briefly overview exact matching and some related problems.

There are strongly polynomial time algorithms for finding a maximum weight perfect matching in a graph (cf. [LP86]). That is, it can be decided in polynomial time whether a perfect matching with weight *at least* K exists. In the *exact matching* problem we have to find a perfect matching whose weight *equals* K. The exact matching problem can be solved in randomized polynomial time if every weight is an integer smaller than some polynomial of the size of the graph [MVV87]. That is, there is a polynomial time algorithm that answers "no" if there is no matching with weight exactly K, and answers "yes" with probability at least $\frac{1}{2}$ if there is such a matching. It is an open question whether there exists a deterministic polynomial time algorithm for this problem.

A related problem is the following: given a graph $G(V, E)$, a set of edges $F \subseteq E$, and an integer k, find a perfect matching M of G with $|M \cap F| = k$. This can be reduced to the exact matching problem: let every edge in F have weight 1 and every other edge weight 0. Now a matching M has $|M \cap F| = k$ if and only if its weight equals k. More generally, we can consider more than one subset of the edges:

Proposition 2.3.9. *Assume that we are given a graph $G(V, E)$, pairwise disjoint subsets $F_0, F_1, \ldots, F_\ell \subseteq E$, and integers $k_0, k_1, \ldots, k_\ell$. If ℓ is fixed, then it can be decided in randomized polynomial time whether there is a perfect matching M with $|M \cap F_i| = k_i$ for every $0 \leq i \leq \ell$.*

Proof. This problem can be reduced to exact matching: let $e \in F_i$ have weight $(|E| + 1)^i$, edges in $E \setminus (F_0 \cup \cdots \cup F_\ell)$ have weight 0, and set $K = \sum_{i=0}^{\ell} k_i(|E|+1)^i$. It is easy to see that a matching satisfies the requirements if and only if it has weight K. If ℓ is a fixed constant, then the weight of every edge is polynomially bounded in the size of the graph, thus the problem can be solved in randomized polynomial time. ■

Theorem 2.3.10. *For every fixed ℓ, there is a randomized polynomial time algorithm for list edge multicoloring in connected graphs having at most $|V| + \ell$ edges.*

Proof. Let T be a spanning tree of $G(V, E)$, and let $E' = \{e_0, e_1, \ldots, e_\ell\}$ be the $|E| - (|V| - 1) = \ell + 1$ edges not in T. Notice that $E \setminus E'$ has full edge rank (it is a tree). Construct the graph G', as in the proof of Theorem 2.3.4. For every $0 \leq i \leq \ell$, let F_i contain the $|L(e_i)|$ edges in G' that corresponds to edge e_i. That is, if $e_i = uv$ and $L(e_i) = \{c_1, c_2, \ldots, c_r\}$, then $F_i = \{u_{c_1}v_{c_1}, u_{c_2}v_{c_2}, \ldots, u_{c_r}v_{c_r}\}$. Set k_i to $x(e_i)$.

We show that the list edge multicoloring problem has a solution if and only if G' has a perfect matching M with $|F_i \cap M| = k_i$ for every $0 \leq i \leq \ell$. By Prop. 2.3.9, the latter problem can be solved in randomized polynomial time, hence the theorem follows.

If there is a perfect matching M in G such that $|F_i \cap M| = k_i$ for every $0 \leq i \leq \ell$, then construct a coloring Ψ valid for the vertices, as in the proof of Theorem 2.3.4. Clearly, $|\Psi(e_i)| = k_i = x(e_i)$. Thus by Lemma 2.3.8, Ψ is valid for the edges.

The other direction also follows easily: since any coloring Ψ valid for the edges is also valid for the vertices, one can find a perfect matching M of G' based on Ψ. It is clear from the construction that M has exactly k_i edges from F_i, since $|\Psi(e_i)| = x(e_i) = k_i$. ■

Corollary 2.3.11. *List edge multicoloring can be solved in randomized polynomial time for even cycles.* ■

2.3.3 Applications and extensions

List edge multicoloring can be applied to model the scheduling of file transfers between processors. Let the vertices be the processors, if there is an edge e with demand $x(e)$ between two vertices, then this means that there is a direct connection between the two processors and a file has to be transfered on this connection. The file transfer represented by the edge e requires $x(e)$ time slots. The colors correspond to the time slots, thus by assigning color sets to the edges we can schedule the file transfers. The requirement that every color appears at most once at a vertex ensures that a processor performs at most one transfer at the same time. Furthermore, by setting the lists of the edges appropriately, we can express additional constraints. For example, we can require that a transfer is not started before a certain time, or it is finished before a certain deadline. Corollary 2.3.6 and Theorem 2.3.10 give efficient algorithms for this problem if the topology of the network is almost a tree. The same method can be applied to model the mutual diagnostic testing of processors: in this case an edge e with demand $x(e)$ means that the two processors have to test each other for $x(e)$ units of time.

In the application examples of the previous paragraph we have assumed that a processor can do only one thing at the same time. This requirement can be weakened: we can consider the variant of the problem where a processor can participate in at most f file transfers simultaneously. More generally, this number f can be different for the different processors. This leads to the f-coloring problem [ZN99, ZFN00]: given a function f on the vertices, we have to find a coloring where every color appears at most $f(v)$ times at vertex v. This more general problem can be handled with a similar reduction to matching. However, now we have to allow that at most $f(v)$ edges incident to a vertex v_c are selected into the matching. Thus the problem can be formulated as a b-matching, we omit the straightforward details. (The b-matching problem is a generalization of matching: we have a function b on the vertices, and the task is to find a set of edges such that exactly $b(v)$ edges are incident to vertex v.)

List edge multicoloring turns out to be useful in a different scheduling model as well. Let us assume that there are n days and m workers. On the ith day $x(i)$ workers are required. The jth worker is available only on days $W(i) \subseteq \{1, \ldots, n\}$ and it is not allowed to use a worker on two consecutive days. This problem can be formulated as list edge multicoloring on a path: the edges e_1, e_2, …, e_n of the path correspond to the days, the colors correspond to the workers. The demand of edge e_i is $x(i)$ and its list is $\{j : i \in W(j)\}$, the set of workers available on the ith day. A list edge multicoloring of the path is clearly equivalent to an allowed job assignment. List multicoloring of paths, which is the same problem as the list edge multicoloring of paths, was investigated [KG02].

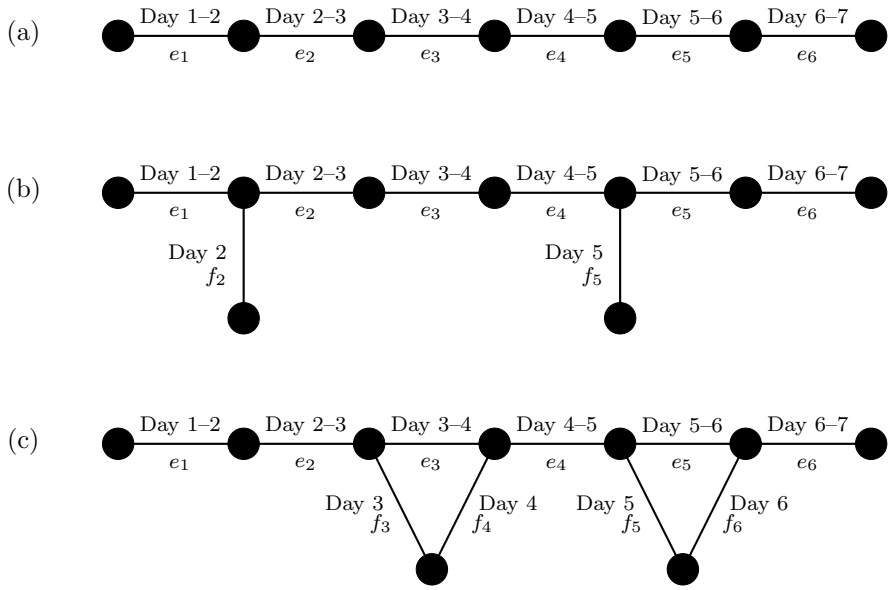


Figure 2.8: (a) The edges of the path correspond to the two-day jobs. (b) The pendant edges represent one-day jobs. (c) A worker can be assigned to the one-day jobs on day 3 and day 4, but not to both (and similarly with the one-day jobs on day 5 and 6).

The requirement that a worker cannot work on two consecutive day appears naturally if we are considering two-day jobs. That is, $x(i)$ denotes the number of workers required for the two-day job starting on day i, and assigning a worker to day i means that he will work on this job on day i and $i+1$ (see Figure 2.8a). Clearly, this means that a worker cannot be assigned to both day i and day $i+1$.

We can extend the problem by allowing both one-day and two-day jobs. Let vertex v be the vertex shared by edge e_i and edge e_{i+1}. Let us attach a new pendant edge f_{i+1} to vertex v (see Figure 2.8b). The edge f_{i+1} represents a one-day job on day $i+1$. If a color is assigned to edge f_{i+1}, then this color cannot be used on edges e_i and e_{i+1}. This corresponds to the requirement that a worker assigned to the one-day job on day $i+1$ cannot work on the two-day jobs starting on day i and $i+1$.

By slightly modifying the graph, we can express more complex requirements (see Figure 2.8c). If edges f_i and f_{i+1} have a common end point, then this ensures that a worker cannot be assigned to the one-day jobs on both day i and $i+1$. However, now the graph is no longer a tree. But if there is only a small number of such requirements, then the graph has only few cycles, and Theorem 2.3.10 gives an efficient algorithm for the problem.

CHAPTER 3

Precoloring extension

To finish a work? To finish a picture? What nonsense! To finish it means to be through with it, to kill it, to rid it of its soul, to give it its final blow ... the coup de grâce for the painter as well as for the picture.
Pablo Picasso (1881–1973)

In the *precoloring extension* problem we are given a graph $G(V, E)$ with a subset W of precolored vertices, and it has to be decided whether this coloring can be extended to the whole graph. The problem is formally defined as follows:

Precoloring Extension (PrExt)

Input: A graph $G(V, E)$, a subset $W \subseteq V$, a coloring ψ' of W and an integer k.

Question: Is there a proper k-coloring ψ of G extending the coloring ψ' (that is, $\psi(v) = \psi'(v)$ for every $v \in W$)?

The special case of the problem where every color is used at most d times in the precoloring will be denoted by d-PrExt. In particular, we will focus on the 1-PrExt problem.

Since vertex coloring is a special case of precoloring extension with $W = \emptyset$, thus PrExt is NP-complete for every class of graphs where vertex coloring is NP-complete. Therefore we can hope to solve PrExt efficiently only on graphs that are easy to color. In the 90s, Biró, Hujter and Tuza [BHT92, HT93, HT96] started a systematic study of precoloring extension in perfect graphs, where coloring can be done in polynomial time. It turns out that for some classes of perfect graphs, e.g., split graphs [HT93], complements of bipartite graphs [HT93], and cographs [HT96, JS97], the precoloring extension problem can be solved in polynomial time. On the other hand, for some other classes like bipartite graphs [HT93, BJW94], line graphs of bipartite graphs [EP01, Col84, Fia03], and interval graphs [BHT92], precoloring extension is NP-complete.

Precoloring extension can be also thought of as a special case of list coloring. The list of a precolored vertex contains only a single color, while if a vertex is not precolored, then its list contains all the available colors. Therefore any hardness result obtained for precoloring extension applies also for list coloring.

Motivated by the aircraft scheduling application presented in Chapter 1, Biró, Hujter, and Tuza studied the precoloring extension problem for interval graphs [BHT92, BHT93]. They have shown that PrExt (in fact, even 2-PrExt) is NP-complete for interval graphs, but 1-PrExt is polynomial-time solvable. In [HT96] they formulate two open questions concerning the possible strengthening of these results:

- Is it possible to solve 1-PrExt in polynomial time for chordal graphs? (A graph is chordal if it does not contain induced cycles of length greater than 3. Every interval graph is chordal.)

- Does PrExt remain NP-complete restricted to unit interval graphs? (A unit interval graph is an interval graph that can be represented by unit length intervals.)

We give positive answers to both questions. In Section 3.1 a polynomial-time algorithm is presented for 1-PrExt on chordal graphs, a class generalizing interval graphs. In Section 3.3 we show that PrExt is NP-complete not only for interval graphs, but even for the smaller class of unit interval graphs. The proof is by reduction from a disjoint paths problem. The NP-completeness of this disjoint paths problem is proved in Section 3.2. The result on the disjoint paths problem is of independent interest: it answers an open question of Vygen [Vyg94].

Finally, Section 3.4 gives complexity results on the edge coloring version of precoloring extension. Easton and Parker [EP01], and independently Fiala [Fia03] have shown that the problem is NP-complete for bipartite graphs of maximum degree 3. In Section 3.4 we strengthen this result by showing that the problem remains NP-complete for planar 3-regular bipartite graphs. This result will be used in Section 5.2 to prove that minimum sum edge coloring is NP-hard for planar bipartite graphs. Section 3.4 also shows that precoloring edge extension is NP-complete for outerplanar and series-parallel graphs. The reductions are from the corresponding list edge coloring problems investigated in Section 2.1.1.

The results in Section 3.1 were presented in [Mar04j]. The results of Section 3.2 appeared in [Mar04d]. The results of Section 3.3 and Section 3.4 will appear in [Mar04k] and [Mar04i], respectively.

3.1 Chordal graphs

The aim of this section is to prove that 1-PrExt is polynomial-time solvable for chordal graphs, which answers an open question of Hujter and Tuza [HT96]. It is easy to reduce PrExt to 1-PrExt: collapse the vertices precolored with the same color to a single vertex. Therefore 1-PrExt is not easier than PrExt on classes of graphs that are closed for this operation. One can also show that 1-PrExt is NP-complete on bipartite graphs [HT93, BJW94].

However, there are cases where 1-PrExt is strictly easier than PrExt. For planar bipartite graphs, if the set of colors C contains only 3 colors, then PrExt is NP-complete [Kra93], while 1-PrExt can be solved in polynomial time [MTW98]. For interval graphs already 2-PrExt is NP-complete [BHT92], but 1-PrExt can be solved in polynomial time [BHT92].

Every chordal graph is perfect and interval graphs form a subset of chordal graphs (cf. [Gol80]). Therefore from the NP-completeness of 2-PrExt for interval graphs [BHT92] it follows that the problem is NP-complete for chordal graphs as well. The complexity of 1-PrExt on chordal graphs is posed by Hujter and Tuza as an open question [HT96]. Here we show that 1-PrExt can be solved in polynomial time also for chordal graphs. The algorithm is a generalization of the method of [BHT92] for interval graphs. As in [BHT92], 1-PrExt is reduced to a network flow problem, but for chordal graphs a more elaborate construction is required than for interval graphs.

In Section 3.1.1 we review some known properties of chordal graphs. In Section 3.1.2 we define a set system that will be crucial in the analysis of the algorithm. The algorithm is presented in Section 3.1.3. In Section 3.1.4 we discuss some connections of the problem with matroid theory.

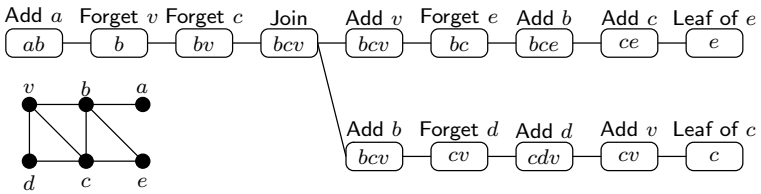


Figure 3.1: Nice tree decomposition of a chordal graph.

3.1.1 Tree decomposition

A graph is *chordal* if every cycle of length greater than 3 contains at least one chord, i.e., an edge connecting two vertices not adjacent in the cycle. Equivalently, a graph is chordal if and only if it does not contain a cycle of length greater than 3 as an induced subgraph. This section summarizes some well-known properties of chordal graphs. First, chordal graphs can be also characterized as the intersection graphs of subtrees of a tree (see e.g., [Gol80]):

Theorem 3.1.1. *The following two statements are equivalent:*

1. *$G(V, E)$ is chordal.*
2. *There exists a tree $T(U, F)$ and a subtree $T_v \subseteq T$ for each $v \in V$ such that $u, v \in V$ are neighbors in $G(V, E)$ if and only if $T_u \cap T_v \neq \emptyset$.*

The tree T together with the subtrees T_v is called the *tree decomposition* of G. Given a chordal graph G, a tree decomposition can be found in polynomial time (see [Gol80, RTL76]).

For clarity, we will use the word "vertex" when we refer to the graph $G(V, E)$, and "node" when referring to $T(U, F)$. We assume that T is a rooted tree with some root $r \in U$. For a node $x \in U$, let T^x be the subtree of T rooted at x. Consider those subtrees T_v that contain at least one node of T^x, denote by V_x the set of corresponding vertices v. The subgraph of G induced by V_x will be denoted by $G_x = G[V_x]$. For a node $x \in U$ of T, denote by K_x the union of v's where $x \in T_v$. Clearly, the vertices of K_x are in V_x, and they form a clique in G_x, since the corresponding trees intersect in T at node x. The tree decomposition has the following property: for every node $x \in U$, the clique K_x separates $V_x \setminus K_x$ and $V \setminus V_x$. That is, among the vertices of V_x, only the vertices in K_x can be adjacent to $V \setminus V_x$.

Every inclusionwise maximal clique of a chordal graph is a clique K_x of the tree decomposition. This is a consequence of the fact that subtrees of a tree satisfy the Helly property (a family of sets is said to satisfy the Helly property if for each pairwise intersecting collection of sets from the family it follows that the sets in the collection have a common element). If K is a clique, then its vertices correspond to pairwise intersecting subtrees, hence by the Helly property, these trees have a common node x, implying $K \subseteq K_x$.

Since every chordal graph is perfect, the chromatic number of G equals its clique number, and it follows that G is k-colorable if and only if $|K_x| \leq k$ for every node $x \in T$. Clearly, the precoloring can exist only if G is $|C|$-colorable, hence we assume in the following that $|K_x| \leq |C|$ holds for every $x \in T$.

A tree decomposition will be called *nice* [Klo94], if it satisfies the following additional requirements:

- Every node $x \in U$ has at most two children.
- If $x \in U$ has two children $y, z \in U$, then $K_x = K_y = K_z$ (x is a *join* node).
- If $x \in U$ has only one child $y \in U$, then either $K_x = K_y \cup \{v\}$ (x is the *add* node of v) or $K_x = K_y \setminus \{v\}$ (x is the *forget* node of v) for some $v \in V$.
- If $x \in U$ has no children, then K_x contains exactly one vertex (x is a *leaf* node).

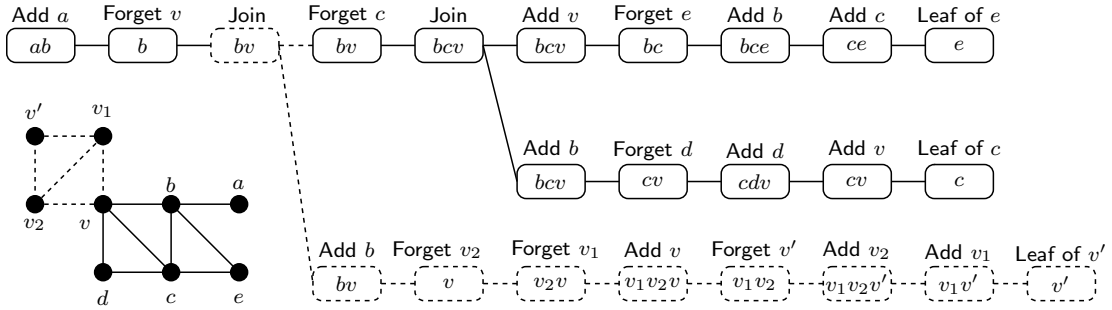


Figure 3.2: Nice tree decomposition of the graph shown on Figure 3.1, after adding the vertices v', v_1, v_2 to the graph. Dashed lines show the new parts of the tree decomposition.

Figure 3.1 shows a nice tree decomposition. It is easy to see that by splitting the nodes of the tree in an appropriate way, a tree decomposition of G can be transformed into a nice tree decomposition in polynomial time. A vertex v can have multiple add nodes, but at most one forget node (the vertices in clique K_r of the root r have no forget nodes, but every other vertex has exactly one). For a vertex v, its subtree T_v is the subtree rooted at the forget node of v (if it exists, otherwise at the root) and whose leaves are exactly the add nodes and leaf nodes of v.

Given a graph G and a precolored set of vertices, we modify the graph to obtain an even nicer tree decomposition. For each precolored vertex v, we add a clique K of $|C|-1$ new vertices, each vertex of K is connected to v; and we also add a new vertex v' that is connected to each vertex of K (but not to v). The precoloring of vertex v is removed and v' becomes a precolored vertex, the color of v is assigned to v'. It is easy to see that this transformation does not change the solvability of the instance: vertices v and v' receive the same color in every $|C|$-coloring of the new graph G' (since they are both connected to the same clique of $|C|-1$ vertices), thus a precoloring extension of G' induces a precoloring extension for G. Although the transformation increases the size of the graph, it will be useful, since now we can assume that the nice tree decomposition has the following additional properties:

- If $x \in U$ is the add node of v, then v is not a precolored vertex.
- If $x \in U$ is a join node, then K_x does not contain precolored vertices.

We show how a nice tree decomposition T of G can be modified to obtain a nice tree decomposition T' of G' satisfying these two additional properties. Let $v_1, v_2, \ldots, v_{|C|-1}$ be the neighbors of v' in G'. Let x be an arbitrary node containing vertex v, let $K_x = \{v, w_1, w_2, \ldots, w_t\}$. Insert a new join node y between x and its parent, we attach a new branch to y of x. This branch will contain the subtrees representing the vertices v', v_1, $\ldots$, $v_{|C|-1}$. The new branch is a path, containing the following nodes (see Figure 3.2):

- Leaf node containing v'.
- Add node of v_1, add node of v_2, $\ldots$, add node of $v_{|C|-1}$.
- Forget node of v'.
- Add node of v.
- Forget node of v_1, forget node of v_2, $\ldots$, forget node of $v_{|C|-1}$.
- Add node of w_1, add node of w_2, $\ldots$, add node of w_t.

It is clear that this modification results in a nice tree decomposition, and if we perform it for each precolored vertex v, then we obtain a decomposition of G'.

3.1.2 System of extensions

Let H be an induced subgraph of G, and let K be a clique of H. We define a set system $\mathscr{S}(H,K)$ over K that will play an important role in the analysis of the algorithm. Denote by $C_H \subseteq C$ those colors that the precoloring assigns to vertices in H. The set system $\mathscr{S}(H,K)$ is defined as follows:

Definition 3.1.2. *For $S \subseteq K$, the set S is in $\mathscr{S}(H,K)$ if and only if there is a precoloring extension $\psi: V(H) \to C$ of subgraph H such that*

- *$\psi(v) \in C_H$ for every $v \in S$, and*
- *$\psi(v) \notin C_H$ for every $v \in K \setminus S$.*

Thus the set system $\mathscr{S}(H,K)$ describes all the possible colorings that can appear on K in a precoloring extension of H, but this description only distinguishes between colors in C_H and colors not in C_H. In particular, the precoloring can be extended to H if and only if $\mathscr{S}(H,K)$ is not empty. If H contains no precolored vertices, but it can be colored with $|C|$ colors, then $\mathscr{S}(H,K)$ contains only the empty set.

The following observation bounds the possible size of a set in $\mathscr{S}(H,K)$:

Observation 3.1.3. *If $S \in \mathscr{S}(H,K)$, then*

$$|K| - |C \setminus C_H| \leq |S| \leq |C_H|$$

Proof. If $S \in \mathscr{S}(H,K)$, then there is a coloring ψ that assigns exactly $|S|$ colors from C_H to the vertices of K. Clearly, in ψ at most $|C_H|$ vertices of the clique K can receive colors from C_H, proving the upper bound. Coloring ψ assigns colors from $C \setminus C_H$ to the vertices in $K \setminus S$, hence $|C \setminus C_H| \geq |K| - |S|$, and the lower bound follows. ■

The definition of this set system is somewhat technical, but it precisely captures the information necessary for solving the precoloring extension problem. Let K be a *clique separator* of G, that is, K is a clique such that its removal separates the graph into two or more components. Let $V \setminus K = V_1 \cup V_2$ be a partition of the remaining vertices such that there is no edge between V_1 and V_2 (that is, each of V_1 and V_2 contains one or more connected components of $V \setminus K$). Let $G_1 = G[V_1 \cup K]$ and $G_2 = G[V_2 \cup K]$. Assume that we have already extended the precoloring to G_1 (coloring ψ_1) and to G_2 (coloring ψ_2). If $\psi_1(v) = \psi_2(v)$ for every vertex v of the clique K, then they can be merged to obtain a coloring of G. Therefore G has a precoloring extension if and only if there is a precoloring extension ψ_1 of G_1, and a precoloring extension ψ_2 of G_2 such that they agree on K. This means that if we have the list of all possible colorings that a precoloring extension of G_1 can assign to K, then to decide if G has a precoloring extension this list is all the information required from the graph G_1. More formally, if we replace G_1 with a graph that has the same list of possible colorings on K, then this does not change the existence of a precoloring extension on G.

However, the following lemma shows that even less information is sufficient: we do not need the list of all possible colorings that can appear on clique K in a coloring of G_1, the set system $\mathscr{S}(G_1,K)$ is sufficient. More precisely, the set system $\mathscr{S}(G,K)$ can be constructed from $\mathscr{S}(G_1,K)$ and $\mathscr{S}(G_2,K)$, hence these two systems are sufficient to decide whether G has a precoloring extension.

Lemma 3.1.4. *Let K be a clique separator of $G(V_1 \cup K \cup V_2, E)$ containing no precolored vertices, let $G_1 = G[V_1 \cup K]$ and $G_2 = G[V_2 \cup K]$. A set $S \subseteq K$ is in $\mathscr{S}(G,K)$ if and only if $|S| \geq |K| - |C \setminus C_G|$ and S can be partitioned into disjoint sets $S_1 \in \mathscr{S}(G_1,K)$ and $S_2 \in \mathscr{S}(G_2,K)$.*

Proof. Assume first that $S \in \mathscr{S}(G,K)$ and let ψ be a coloring corresponding to the set S. Observation 3.1.3 implies that $|S| \geq |K| - |C \setminus C_G|$, as required. Coloring ψ induces a coloring ψ_i of G_i, let $S_i \in \mathscr{S}(G_i,K)$ be the set corresponding to ψ_i $(i = 1,2)$. Coloring ψ can assign three different types of colors to the vertices in K:

- If $\psi(v) \notin C_G$ (i.e., $\psi(v)$ is not used in the precoloring), then $v \notin S, S_1, S_2$.
- If the precoloring uses $\psi(v)$ in V_1, then $v \in (S \cap S_1) \setminus S_2$. (Since each color is used at most once in the precoloring, $\psi(v)$ cannot appear in V_2 on a precolored vertex.)
- If the precoloring uses $\psi(v)$ in V_2, then $v \in (S \cap S_2) \setminus S_1$.

Note that v cannot be a precolored vertex, hence the precoloring cannot use $\psi(v)$ in K. Therefore S is the disjoint union of S_1 and S_2, as required.

Now assume that S can be partitioned into disjoint sets $S_1 \in \mathscr{S}(G_1, K)$ and $S_2 \in \mathscr{S}(G_2, K)$, let ψ_1 and ψ_2 be the two corresponding colorings. In general, ψ_1 and ψ_2 might be different on K, thus they cannot be combined to obtain a coloring of G. However, with some permutations of colors we modify the two colorings in such a way that they assign the same color to every vertex of K. Let C_1 (resp. C_2) be the colors of the precolored vertices in V_1 (resp. V_2). Notice that both ψ_1 and ψ_2 assign colors from $C \setminus C_1$ to S_2, (since S_1 and S_2 are disjoint). Modify coloring ψ_1: permute the colors of $C \setminus C_1$ such that $\psi_1(v) = \psi_2(v)$ holds for every $v \in S_2$ (this can be done since K is a clique, hence both ψ_1 and ψ_2 assign distinct colors to the vertices in S_2). Since the precolored vertices in V_1 have colors only from C_1, coloring ψ_1 remains a valid precoloring extension for G_1. Similarly, in coloring ψ_2, permute the colors of $C \setminus C_2$ such that $\psi_1(v) = \psi_2(v)$ for every $v \in S_1$. Now we have that ψ_1 and ψ_2 agree on S, there might be differences only on $K \setminus S$. Moreover, ψ_1 uses only colors from $C \setminus C_1$ on $K \setminus S$, and ψ_2 uses colors only from $C \setminus C_2$ on this set. Now select a set $C' \subseteq C \setminus C_G$ such that $|C'| = |K \setminus S|$ (here we use the assumption $|S| \geq |K| - |C \setminus C_G|$, which implies that there are enough colors in $C \setminus C_G$). Permute again the colors of $C \setminus C_1$ in coloring ψ_1 such that ψ_1 assigns to $K \setminus S$ exactly the colors in C'. Similarly, permute the colors of $C \setminus C_2$ in coloring ψ_2 such that ψ_2 also uses C' on $K \setminus S$. Now the colorings ψ_1 and ψ_2 agree on K, hence we can combine them to obtain a coloring ψ of G. This coloring proves that $S = S_1 \cup S_2$ is in $\mathscr{S}(G, K)$, what we had to show. ■

Lemma 3.1.4 implies that if we know the set systems $\mathscr{S}(G_1, K)$ and $\mathscr{S}(G_2, K)$, then the set system $\mathscr{S}(G, K)$ can be also determined. This suggests the following algorithm: for each node x of the tree decomposition, determine $\mathscr{S}(G_x, K_x)$. In principle, this can be done in a bottom-up fashion: the set system for node x can be determined from the systems of its children. Unfortunately, the size of $\mathscr{S}(G_x, K_x)$ can be exponential, thus it cannot be constructed explicitly during the algorithm. However, if G is a chordal graph, then these set systems have nice combinatorial structure that allows a compact representation. The main idea of the algorithm in Section 3.1.3 is to use network flows to represent the set systems $\mathscr{S}(G_x, K_x)$. In Section 3.1.4 we discuss formally what is this nice structure that makes possible the representation with flows: it turns out that if G is chordal and K is a clique of G, then $\mathscr{S}(G, K)$ is the projection of a matroid.

3.1.3 The algorithm

Here we prove the main result of this section:

Theorem 3.1.5. 1-PrExt *can be solved in polynomial time for chordal graphs.*

Given an instance of the 1-PrExt problem, we construct a network flow problem that has a feasible flow if and only if there is a solution to 1-PrExt. We use the following variant of the flow problem. The *network* is a directed graph $D(U, A)$, each arc $e \in A$ has an integer capacity $c(e)$. The set of arcs entering (resp. leaving) node v will be denoted by $\delta^-(v)$ (resp. $\delta^+(v)$). The set of *sources* is $S \subseteq U$, and $T \subseteq U$ is the set of *terminals* in the network (we require $S \cap T = \emptyset$). Every source $v \in S$ produces exactly one unit amount of flow, and every terminal $v \in T$ has a capacity $w(v)$, it can consume up to $w(v)$ units. Formally, a *feasible flow* is a function $f\colon A \to \mathbb{Z}^+$ that satisfies $0 \leq f(e) \leq c(e)$ for every arc $e \in A$, and the following holds for every node $v \in U$:

- If $v \in S$, then $\sum_{e \in \delta^-(v)} f(e) - \sum_{e \in \delta^+(v)} f(e) = -1$.
- If $v \in T$, then $0 \leq \sum_{e \in \delta^-(v)} f(e) - \sum_{e \in \delta^+(v)} f(e) \leq w(v)$.
- If $v \in U \setminus (T \cup S)$, then $\sum_{e \in \delta^-(v)} f(e) = \sum_{e \in \delta^+(v)} f(e)$.

Using standard techniques, the existence of a feasible flow can be tested by a maximum flow algorithm. It is sufficient to add two new vertices s and t, an arc with capacity 1 from s to every vertex $v \in S$, and an arc with capacity $w(v)$ to t from every vertex $v \in T$. Clearly, there is a feasible flow in the original network if and only if there is an $\overrightarrow{st}$ flow with value $|S|$ in the modified network. The maximum flow can be determined using at most $|S|$ iterations of the Edmonds-Karp augmenting path algorithm, hence the existence of a feasible flow in a network $D(U, A)$ can be tested in $O(|S||A|)$ time.

Given a chordal graph $G(V, E)$, its nice tree decomposition $T(U, F)$, $\{T_v \mid v \in V\}$, and the set of precolored vertices $W \subseteq V$, we construct a network as follows. Direct every edge of T towards the root r. For every $v \in V$ and for every $x \in T_v$ add a node x_v to the network. Denote by U_x the $|K_x|$ nodes corresponding to x. If the edge xy is in T_v, then connect $x_v \in U_x$ and $y_v \in U_y$ by an arc. If y is the child of x, then direct this arc from y_v to x_v. These new arcs $\overrightarrow{y_v x_v}$ have capacity 1, while the arcs $\overrightarrow{yx}$ of the tree T have capacity $|C| - |K_y|$ (recall that if the graph is $|C|$-colorable, then $|K_y| \leq |C|$).

For each node $x \in T$, depending on the type of x, we do one of the following:

- If x is an add node of some vertex $v \notin W$, and y is the child of x, then add an arc $\overrightarrow{yx_v}$ to the network.
- If leaf node x contains some vertex $v \in W$, then add a new node x'_v to the network, add an arc $\overrightarrow{x'_v x_v}$ with capacity 1, and set x'_v to be a source.
- If x is a forget node of some vertex v (either in W or not), and y is the child of x, then add an arc $\overrightarrow{y_v x}$ to the network.

For join nodes and for leaf nodes containing vertices outside W we do nothing. Figure 3.3 sketches the construction for the different types of nodes.

So far there are no terminals in the network. The definition of the network is completed by adding terminals as follows. Here we define not only a single network, but several subnetworks that will be useful in the analysis of the algorithm. For every node $x \in U$ of the tree T, the network N_x contains only those nodes of the network that correspond to nodes in T^x (recall that T^x is the subtree of T rooted at x). Formally, the network N_x has the node set $T^x \cup \bigcup_{y \in T^x} U_y$, and the sources nodes (if available) corresponding to the leaves of T^x. Moreover, in network N_x the nodes in U_x are set to be terminals with capacity 1, and node x is a terminal with capacity $|C| - |K_x|$. This completes the description of the network N_x.

Notice that there are sources only at the leaf nodes of precolored vertices. Therefore the number of sources in network N_x is the same as the number of precolored vertices in V_x (recall that V_x is the set of those vertices v whose tree T_v has at least one node in T^x, and $G_x = G[V_x]$). We will denote by C_x the set of colors that appear on the precolored vertices of V_x. In network N_x, there are terminals only at x and U_x, these terminals must consume all the flow.

Observation 3.1.6. *The number of sources in N_x equals the number of precolored vertices in V_x, which is $|C_x|$. Consequently, in every feasible flow of N_x, the amount of flow consumed by the terminals at x and U_x is exactly $|C_x|$.* ■

To prove Theorem 3.1.5, we show that the precoloring of G can be extended to the whole graph if and only if there is a feasible flow in N_r, where r is the root of T. This gives a polynomial-time algorithm for 1-PrExt in chordal graphs, since constructing network N_r and finding a feasible flow in N_r can be done

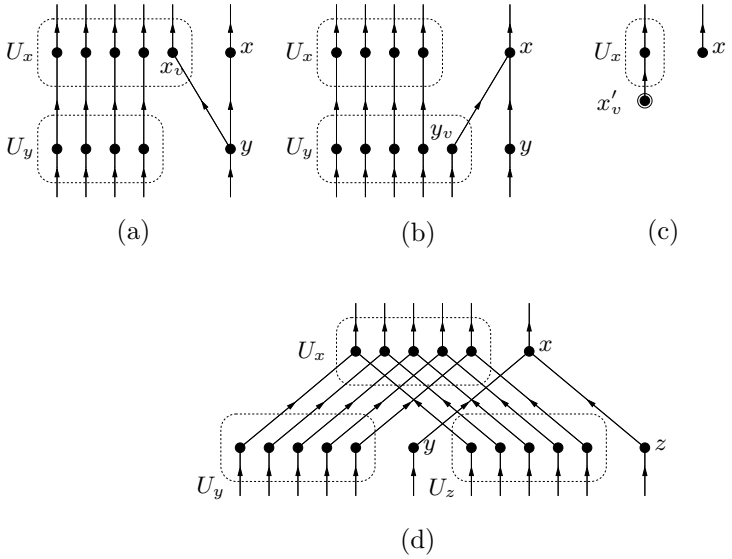


Figure 3.3: Construction of the network when node x is of the following types: (a) add node for $v \notin W$, (b) forget node for v (either in W or not), (c) leaf node for $v \in W$, (d) join node.

in polynomial time. The proof of this claim uses induction on the tree decomposition of the graph. For every node $x \in U$ of T, we prove the following more general statement: the network N_x has a feasible flow if and only if the precoloring of G can be extended to G_x.

More precisely, we show that the network N_x represents (in some well-defined sense) the set system $\mathscr{S}(G_x, K_x)$: every feasible flow corresponds to a set in the system. Therefore N_x has no feasible flows if and only if $\mathscr{S}(G_x, K_x)$ is empty, or, equivalently, the precoloring cannot be extended to G_x.

We say that a feasible integer flow of N_x *represents the set* $S \subseteq K_x$ if for every $v \in S$, the terminal at x_v consumes one unit of flow, while for every $v \notin S$, there is no flow entering x_v. The following lemma establishes the connection between the constructed networks and the set systems $\mathscr{S}(G_x, K_x)$. The proof of this lemma completes the proof of Theorem 3.1.5, as it reduces the 1-PrExt problem to finding a feasible flow in N_r.

Lemma 3.1.7. *For an arbitrary node $x \in U$ of T, the network N_x has a feasible flow representing a set $S \subseteq K_x$ if and only if $S \in \mathscr{S}(G_x, K_x)$.*

Proof. The lemma is proved for every node x of T by a bottom-up induction on the tree T. After checking the lemma for the leaf nodes, we show that it is true for a node x assuming that it is true for the children of x. The proof is done separately for the different types of nodes. If the node is an add or forget node of some vertex v, then we have to consider two further cases depending on whether v is in W or not (recall that W is the set of precolored vertices). Verifying the lemma in each case is tedious, but it does not require any new ideas. The way the networks are constructed ensures that the set systems represented by the networks have the required properties.

Leaf node. For a leaf node x, the lemma is trivial: if the vertex v in K_x is precolored, then every flow of N_x represents $\{v\}$, otherwise N_x contains no sources, and every flow represents $\emptyset$.

Add node for $v \notin W$. Let x be an add node of $v \notin W$, and let y be the child of x. For every $S \in \mathscr{S}(G_x, K_x)$, it has to be shown that there is a feasible flow of N_x representing the set S. Assume first that $v \notin S$. Since $G_y = G_x \setminus v$ and $K_y = K_x \setminus \{v\}$, it follows that $S \in \mathscr{S}(G_y, K_y)$. Therefore by the induction hypothesis, there is a flow f_y in N_y representing S. We modify this flow to obtain a flow

f_x of N_x also representing S. For every $u \in S$, in flow f_y there is one unit of flow consumed by the terminal at y_u. To obtain flow f_x, direct this unit flow towards x_u, and consume it by the terminal at that node. Similarly, in the flow f_y, there is some amount of flow consumed by the terminal at y, direct this flow to x, and consume it by that terminal. By Observation 3.1.6, the amount of flow consumed at x is exactly $|C_x| - |S|$. Moreover, the lower bound of Observation 3.1.3 implies that this is at most $|C_x| - |K_x| + |C \setminus C_x| = |C| - |K_x| < |C| - |K_y|$, hence the capacity of the arc $\overrightarrow{yx}$ and the terminal at x is sufficient for the flow. Thus we obtained a feasible flow of N_x, and obviously it represents S.

We proceed similarly if $v \in S$. In this case $S \setminus \{v\} \in \mathscr{S}(G_y, K_y)$, thus N_y has a flow f_y representing $S \setminus \{v\}$. To obtain a flow f_x of N_x representing S, the flow consumed at y_u is directed to x_u, as in the previous paragraph. However, now we do not direct all the flow consumed at y to x, but we direct one unit amount through the arc $\overrightarrow{yx_v}$, and only the rest goes through arc $\overrightarrow{yx}$. Therefore the amount of flow consumed by the terminal at x is one unit less than the flow consumed at y in flow f_y, hence the capacity of the terminal at x is sufficient. Clearly, this results in a flow f_x of N_x representing S, as required. The only thing to verify is that there is at least one unit of flow consumed at y in flow f_y. The flow f_y represents $S \setminus \{v\}$, and by Observation 3.1.6, the amount of flow consumed in $U_y \cup \{y\}$ is exactly $|C_y|$, hence the flow consumed at y is $|C_y| - |S| + 1$. Since v is not a precolored vertex, we have that $C_y = C_x$. We know that $S \in \mathscr{S}(G_x, K_x)$, therefore by the upper bound of Observation 3.1.3, $|C_y| - |S| + 1 \geq 1$, hence there is nonzero flow consumed at y in flow f_y.

Now assume that there is a flow f_x in N_x representing $S \subseteq K_x$, it has to be shown that $S \in \mathscr{S}(G_x, K_x)$. Let y be the child of x. Assume first that $v \notin S$, we show that N_y has a flow f_y in N_y representing S. To obtain this f_y, the flow f_x is modified the following way. For every vertex $x_w \in U_x$, where $w \neq v$, if there is flow on the arc $\overrightarrow{y_w x_w}$, then consume it by the terminal at y_w. Similarly, the flow on the arc $\overrightarrow{yx}$ can be consumed by the terminal at y (the capacity of the terminal at y equals the capacity of arc $\overrightarrow{yx}$). It is clear that these modifications result in a feasible flow for N_y that represents S. By the induction hypothesis, this means that $S \in \mathscr{S}(G_y, K_y)$, and there is a corresponding coloring ψ. Since v is the only vertex in $V_x \setminus V_y$, to prove $S \in \mathscr{S}(G_x, K_x)$ it is sufficient to show that coloring ψ can be extended to v in such a way that v receives a color not in C_x. If there is no such extension, then this means that ψ uses every color of $C \setminus C_x$ on the neighbors of v, that is, on the clique K_y. By construction, ψ assigns exactly $|S|$ colors from C_x to the clique K_y, hence if every color of $C \setminus C_x$ is used on K_y, then $|K_y| = |C \setminus C_x| + |S|$. Therefore $|K_x| = |K_y| + 1 = |C \setminus C_x| + |S| + 1$ and the capacity of the terminal at x is $|C| - |K_x| = |C_x| - |S| - 1$. However, in flow f_x of N_x that represents S, exactly $|C_x| - |S|$ unit of flow is consumed at x (Observation 3.1.6), a contradiction. Thus ψ can be extended to v, and $S \in \mathscr{S}(G_x, K_x)$ follows.

The case $v \in S$ can be handled similarly. If there is a flow f_x in N_x that represents S, then this is only possible if there is flow on the arc $\overrightarrow{yx_v}$. Therefore by restricting the flow to N_y as in the previous paragraph, we can obtain a flow representing $S \setminus \{v\}$. (Notice that the capacity of the terminal at y equals the combined capacity of the terminals at x and x_v, hence it can consume the flow on the arcs $\overrightarrow{yx_v}$ and $\overrightarrow{yx}$.)

By the induction hypothesis, it follows that $S \setminus \{v\} \in \mathscr{S}(G_y, K_y)$, and there is a corresponding coloring ψ. Now it has to be shown that ψ can be extended to vertex v such that v receives a color from C_x. Coloring ψ assigns exactly $|S| - 1$ colors from C_x to K_y. The extension is not possible only in the case if every color of C_x is already used on K_y, that is, if $|C_x| = |S| - 1$. This would imply that in flow f_x of N_x, the amount of flow consumed at U_x is $|S| = |C_x| + 1$. However, by Observation 3.1.6, this is strictly larger than the number of sources in N_x, a contradiction.

Forget node for v (vertex v is either in W or not). Let x be the forget node of v, and let y be the child of x. Let $S \in \mathscr{S}(G_x, K_x)$. Since $G_x = G_y$, either S or $S \cup \{v\}$ is in $\mathscr{S}(G_y, K_y)$. In the first case, the flow f_y in N_y that represents S can be extended to a flow in N_x that also represents S. As before, the flow consumed at y_w is directed to x_w, and the flow consumed at y is directed to x. Recall that the capacity of the arc $\overrightarrow{yx}$ equals the capacity of the terminal at y, while the capacity of the terminal at x is strictly greater. Therefore the resulting flow is feasible in N_x, and clearly it represents S. If $S \cup \{v\} \in \mathscr{S}(G_y, K_y)$,

then we do the same, but the flow consumed at y_v is directed to x through the arc $\overrightarrow{y_v x}$. The resulting flow is feasible in N_x and represents S.

To prove the other direction, assume that N_x has a flow f_x representing $S \subseteq K_x$. Restrict this flow to N_y, that is, modify the flow such that the terminals at y and U_y consume all the flow. This results in a feasible flow f_y of N_y that represents S or $S \cup \{v\}$. Notice that the terminal at y has the same capacity as the arc $\overrightarrow{yx}$, hence this terminal can consume all the flow going through the arc. Therefore, by the induction hypothesis, either S or $S \cup \{v\}$ is in $\mathscr{S}(G_y, K_y)$, depending on whether there is flow consumed at y_v or not. In either case, $S \in \mathscr{S}(G_x, K_y)$ follows since $G_x = G_y$ and $K_x = K_y \setminus \{v\}$.

Join node. Let y and z be the two children of the join node x. Let $S \in \mathscr{S}(G_x, K_x)$. By Lemma 3.1.4 this means that

$$|S| \geq |K_x| - |C \setminus C_x| \tag{3.1}$$

and S can be partitioned into disjoint sets $S_1 \in \mathscr{S}(G_y, K_y)$ and $S_2 \in \mathscr{S}(G_z, K_z)$. By the induction hypothesis, this implies that there are flows f_y, f_z in N_y and N_z that represent the sets S_1 and S_2, respectively. We combine these two flows to obtain a flow f_x of N_x that represents the set S. If there is flow consumed at a node $y_v \in U_y$ (resp. $z_v \in U_z$) in f_y (resp. f_z), then direct this flow on the arc $\overrightarrow{y_v x_v}$ (resp. $\overrightarrow{z_v x_v}$) to node x_v, and consume it there. The capacity of the terminal at x_v is 1, but the disjointness of S_1 and S_2 implies that at most one unit of flow is directed to x_v. The flow consumed at node y and z is directed to x on the arc $\overrightarrow{yx}$, $\overrightarrow{zx}$, respectively. Since there are exactly $|S|$ units of flow consumed in U_x, therefore $|C_x| - |S|$ units of flow has to be consumed at x. By (3.1), this is at most $|C_x| - |K_x| + |C \setminus C_x| = |C| - |K_x|$, thus the capacity of the terminal at x is sufficient for consuming this flow. Therefore we have obtained a flow f_x in network N_x that represents S.

Now assume that N_x has a flow f_x that represents S. Since the terminal at x has capacity at most $|C| - |K_x|$, and by Observation 3.1.6, the amount of flow consumed in $x \cup U_x$ is $|C_x|$, it follows that

$$|S| \geq |C_x| - (|C| - |K_x|) = |K_x| - |C \setminus C_x|. \tag{3.2}$$

If flow is consumed at a node $x_v \in U_x$, then the flow arrives to this node either from y_v or from z_v. Define the sets $S_1, S_2 \subseteq K_x$ such that $v \in S_1$ (resp. $v \in S_2$) if there is flow on arc $\overrightarrow{y_v x_v}$ (resp. $\overrightarrow{z_v x_v}$).

Based on the flow f_x of N_x representing S, we create a flow f_y of N_y that represents S_1 and a flow f_z of N_z that represents S_2. The flows f_y and f_z are constructed as follows. For every $y_v \in U_y$, if there is flow going through the arc $\overrightarrow{y_v x_v}$, then consume this flow at y_v, and similarly for the nodes $z_v \in U_z$. The flow on arcs $\overrightarrow{yx}$ and $\overrightarrow{zx}$ are consumed at y and z, respectively (the capacity of nodes x, y, and z are the same $|C| - |K_x| = |C| - |K_y| = |C| - |K_z|$). Clearly, flows f_y and f_z represent S_1 and S_2, respectively. By the induction hypothesis, the flows f_x and f_y imply that $S_1 \in \mathscr{S}(G_y, K_y)$ and $S_2 \in \mathscr{S}(G_z, K_z)$. Furthermore, it is clear that S_1 and S_2 are disjoint, and $S = S_1 \cup S_2$. Therefore by Lemma 3.1.4 and Inequality (3.2), this proves that $S \in \mathscr{S}(G_x, K_x)$, as required.

■

To determine the running time of the algorithm, we have to consider two main steps: the construction of the network and the solution of the flow problem. First of all, the transformation introduced at the end of Section 3.1.1 can increase the size of the graph by at most a factor of n. Given a chordal graph $G(V, E)$, its tree decomposition can be constructed by first finding a perfect vertex elimination scheme [Gol80, RTL76]. Based on this ordering of the vertices, one can build a tree $T(U, F)$ of size $|V|$, and one subtree for each vertex of the graph. This tree decomposition can be found in time linear in the size of the output, that is, in $O(|V|^2)$ time. Converting $T(U, F)$ to a nice tree decomposition can introduce an increase of factor at most $|V|$, thus it can be done in $O(|V|^3)$ time. The network defined by the algorithm has size linear in the total size of the tree decomposition (size of $T(U, F)$ and the sum of the size of the subtrees), and clearly it can be constructed in linear time. Therefore the constructed network has $O(|V|^3)$ nodes and $O(|V|^3)$ arcs, and the construction takes $O(|V|^3)$ time.

In a network with n nodes and m arcs, the maximum flow can be determined in $O(n^2m)$ or even in $O(n^3)$ time [AMO93]. Moreover, it can be determined in $O(km)$ time if a flow with value k exists: the Edmonds-Karp algorithm produces such a flow after finding the first at most k augmenting paths (assuming that the capacities are integer). As discussed in the beginning of the section, the existence of a feasible flow can be tested by finding an s-t flow with value $|S|$, hence it can be done in $O(|S| \cdot |V|^3)$ time. By Observation 3.1.6, this is at most $O(|C| \cdot |V|^3) = O(|V|^4)$. This implies that solving the problem for graphs with up to 100 vertices should be easy. We believe that the running time can be significantly improved by streamlining the construction, hence the algorithm can be made to work efficiently on larger graphs as well. However, our aim was only to prove that the problem can be solved in polynomial time, thus we preferred ease of presentation over efficiently.

The algorithm described above determines whether a precoloring extension exists, but does not find a coloring. However, based on the feasible flow of network N_r, one can construct a precoloring extension of the graph. We have seen that the feasible flow of network N_x represents a set $S_x \in \mathscr{S}(G_x, K_x)$. Recursively for each $x \in U$, we compute a coloring ψ_x corresponding to S_x. For the leaf nodes this is trivial. Let x be an add node of vertex v, and let y be the child of x. To obtain ψ_x, coloring ψ_y has to be extended to v: if there is flow on $\overrightarrow{yx_v}$, then v has to receive a color from C_x, otherwise from $C \setminus C_x$. The construction ensures that there is always such a color not already used on the neighbors of v. If x is a forget node with child y, then ψ_x can be selected to be the same as ψ_y. Finally, assume that x is a join node with children y and z. By the way the network was constructed, S_y and S_z are disjoint, $S_x = S_y \cup S_z$, and $S \geq |K_x| - |C \setminus C_x|$. Therefore the method described in the proof of Lemma 3.1.4 can be used to construct a coloring ψ_x of G_x that corresponds to $S_x \in \mathscr{S}(G_x, K_x)$.

3.1.4 Matroidal systems

The main idea of the algorithm in Section 3.1.3 is to represent the set system $\mathscr{S}(G, K)$ by a network flow. We have shown that for chordal graphs the set systems $\mathscr{S}(G_x, K_x)$ can be represented by network flows for every subgraph G_x and clique K_x given by the tree decomposition. The reason why these systems can be represented by flows is that they have nice combinatorial structure (the proof is given at the end of the section):

Theorem 3.1.8. *Let $G(V, E)$ be a chordal graph, and let $W \subseteq V$ be a arbitrary set of precolored vertices such that every color of C is used at most once in the precoloring. If H is an induced subgraph of G, and K is a clique of H, then the set system $\mathscr{S}(H, K)$ is the projection of the basis set of a matroid.*

Recall that a set system $\mathscr{B}$ is the basis set of a *matroid*, if it satisfies the following two conditions:

- Every set in $\mathscr{B}$ has the same size.
- For every $B_1, B_2 \in \mathscr{B}$ and $v \in B_1 \setminus B_2$, there is an element $u \in B_2 \setminus B_1$ such that $B_1 \cup \{u\} \setminus \{v\} \in \mathscr{B}$.

If $\mathscr{B}$ is a set system over X, then its *projection* to $Y \subseteq X$ is a set system over Y that contains $B' \subseteq Y$ if and only if there is a set $B \in \mathscr{B}$ with $B \cap Y = B'$. The projection of a matroid is always a so-called Δ-matroid [Mur00], hence Theorem 3.1.8 also says that $\mathscr{S}(G, K)$ is a Δ-matroid. For further notions of matroid theory, the reader is referred to e.g., [Rec89].

In general, if G is not chordal, then $\mathscr{S}(G, K)$ is not necessarily the projection of a matroid. Figure 3.4 shows a graph G with two precolored vertices v_1 and v_2. The graph is not chordal, since vertices v_1, v_4, v_2, v_8 induce a cycle of length 4. If we have only four colors, then G has four precoloring extensions: vertex v_3 can have only color 3 or 4, vertex v_9 can have only color 1 or 2, and setting the color of these two vertices forces a unique coloring for the rest of the graph. For example, if coloring ψ assigns color 3 to v_3, and color 1 to v_9, then $\psi(v_3) = 3$, $\psi(v_9) = 1$, $\psi(v_2) = 2$ imply $\psi(v_4) = 4$; $\psi(v_9) = 1$, $\psi(v_2) = 2$, $\psi(v_4) = 4$ imply $\psi(v_6) = 3$; $\psi(v_1) = 1$, $\psi(v_2) = 2$, $\psi(v_6) = 3$ imply $\psi(v_8) = 4$; and finally $\psi(v_5) = 1$

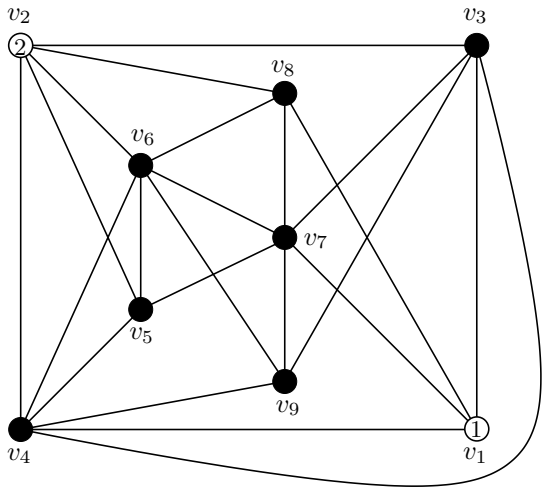


Figure 3.4: A non-chordal graph G and a clique $K = \{v_5, v_6, v_7\}$ such that $\mathscr{S}(G, K)$ is not the projection of a matroid ($|C| = 4$).

and $\psi(v_7) = 2$ follow in a similar fashion. Therefore the clique $K = \{v_5, v_6, v_7\}$ receives one of the four colorings $(3, 1, 4)$, $(1, 3, 2)$, $(4, 1, 3)$, $(1, 4, 2)$ in every precoloring extension. Since $C_G = \{1, 2\}$, it follows that $\mathscr{S}(G, K) = \{\{v_6\}, \{v_5, v_7\}\}$, which cannot be the projection of a matroid (for example, it is not even a Δ-matroid).

The proof of Theorem 3.1.8 uses the following result of matroid theory. In a directed graph $D(U, A)$, we say that $Y \subseteq U$ can be *linked* onto $X \subseteq U$, if $|X| = |Y|$ and there are $|X|$ pairwise node disjoint paths from the nodes in X to the nodes in Y. The sets X and Y do not have to be disjoint, and the zero-length path consisting of a single node is also allowed. Hence X can be linked onto X in particular. The following theorem states that the graph G together with a set $X \subseteq U$ induces a matroid on the vertices of the graph (see e.g., [Rec89]):

Theorem 3.1.9. *If $D(U, A)$ is a directed graph and $X \subseteq U$ is a fixed subset of nodes, then those subsets $Y \subseteq U$ that can be linked onto X form the bases of a matroid M over U.*

Considering the line graph of the directed graph, one can state an arc disjoint version of Theorem 3.1.9:

Theorem 3.1.10. *If $D(U, A)$ is a directed graph, $s \in U$ is a fixed vertex and r is a positive integer, then those r-element subsets $A' \subseteq A$ whose arcs can be reached from s by r pairwise arc disjoint paths form the bases of a matroid M over A.* ■

To prove Theorem 3.1.8, we use the fact that $\mathscr{S}(G_x, K_x)$ can be represented by the network N_x (Lemma 3.1.7). Then Theorem 3.1.10 is used to show that the set system represented by a network is the projection of a matroid.

Proof (of Theorem 3.1.8). Clearly, it is sufficient to consider only the case when $H = G$, since every induced subgraph of a chordal graph is also chordal. Moreover, it can be assumed that K is a maximal (non-extendable) clique: if $K_1 \subseteq K_2$ are two cliques, then $\mathscr{S}(G, K_1)$ is the projection of $\mathscr{S}(G, K_2)$. Therefore if $\mathscr{S}(G, K_2)$ is the projection of the basis set of a matroid, then this also follows for $\mathscr{S}(G, K_1)$. We have seen in Section 3.1.1 that given a tree decomposition $T(U, F)$, $\{T_v\}_{v \in V(G)}$ of the chordal graph G, every maximal clique of G is a clique K_x for some $x \in U$. Furthermore, since the choice of the root node of T is arbitrary, it can be assumed that x is the root, thus we have $G = G_x$ and $\mathscr{S}(G, K) = \mathscr{S}(G_x, K_x)$.

By Lemma 3.1.7, the sets in $\mathscr{S}(G_x, K_x)$ are exactly the sets represented by the feasible flows of the network N_x. Now, as described at the beginning of Section 3.1.3, add two new nodes s, t to the network, add an arc with unit capacity from s to every source, and for every terminal x, add an arc from x to t that has capacity equal to the capacity of x. Furthermore, replace every arc e having capacity $c(e)$ with $c(e)$ parallel arcs of unit capacity, clearly this does not change the problem. Call the resulting network N'_x. By Observation 3.1.6, the number of sources in N_x is $r = |C_x|$, hence every feasible flow of N_x corresponds to an $\overrightarrow{st}$ flow with value r in N'_x. Since every arc has unit capacity in N'_x, an integral $\overrightarrow{st}$ flow with value r corresponds to r arc disjoint paths from s to t. Now consider the matroid M given by Lemma 3.1.10. Denote by A_t the arcs incident to t, and let matroid M_t be the restriction of matroid M to A_t. Let $A'_t \subseteq A_t$ be those arcs of A_t that originate from some node $x_v \in U_x$ (and not from x). We claim that $\mathscr{S}(G, K_x)$ is isomorphic to the projection of M_t to A'_t (vertex $v \in K_x$ maps to arc $\overrightarrow{x_v t}$). By Lemma 3.1.7, if $S \in \mathscr{S}(G, K_x)$, then there is a feasible flow in N_x where flow is consumed only by those terminals of U_x that correspond to the elements in S. Based on this flow, one can find r arc disjoint $\overrightarrow{st}$ paths in N'_x, and it follows that the matroid M_t has a base whose intersection with A'_t is exactly S, hence S is in the projection of M_t to A'_t. It is easy to show the other direction as well: if S is in the projection of M_t, then there is a feasible flow of N_x where only the terminals corresponding to S consume flow in U_x. Thus by Lemma 3.1.7, $S \in \mathscr{S}(G, K_x)$, as required. ■

3.1.5 Applications

A possible application area for Theorem 3.1.5 is the design and configuration of Wavelength Division Multiplexing (WDM) optical networks. WDM technology allows us to establish several data channels in a single optical fiber using the different wavelengths. All-optical switches can route the different channels of an incoming fiber to different outgoing fibers. Therefore by configuring the switches appropriately, we can create direct optical connections between distant nodes of the network. However, in order to configure the network, we have to assign a wavelength to each connection such that connections that use the same fiber receive different wavelengths. This wavelength assignment problem is a coloring problem. The *conflict graph* of the network has one vertex for each connection, and two vertices are neighbors if the corresponding two connections share a fiber. If the number of wavelengths in a fiber is k, then the wavelength assignment problem can be solved if and only if the conflict graph is k-colorable.

The WDM network configuration problem was studied by several papers in the literature, with a particular emphasis on tree and tree-like networks [EJ01, Erl99, EJK$^+$99, EJ98, ART01]. However, the model becomes slightly different if we consider another type of optical switches. The simplest optical switch is the *passive star* that selects one incoming fiber for each wavelength, and transmits the data on this wavelength to every outgoing fiber. If we have this type of switches, then connections having the same wavelength cannot go through the same switch. That is, the connections with the same wavelength have to be vertex disjoint, not only edge disjoint. We can take into account this restriction by modifying the definition of the conflict graph: connect those vertices where the corresponding connections go through the same switch. As discussed in Section 3.1.1, the intersection graph of paths in a tree is always chordal, hence the conflict graph in this problem will be a chordal graph. Vertex coloring is polynomial-time solvable for chordal graphs, hence we can determine in polynomial time whether the wavelength assignment problem can be solved. The algorithm in Theorem 3.1.5 gives a method for solving the slightly more general problem where some of the connections already have a wavelength, but every wavelength is assigned to at most one connection. For example, this is the case if all the connections going through a particular switch are already assigned a wavelength, but we are free to assign any wavelength to the remaining connections.

We remark that with chordal graphs we can model a more general problem as well. The intersection graph of the subtrees in tree gives a chordal graph, thus if the connections are not paths but trees, then the same method works. This observation might be useful if we have to establish broadcast connections between multiple parties.

3.2 The Eulerian disjoint paths problem

Disjoint paths problems arise naturally in practical applications such as network routing and VLSI-design. The problem is also interesting from the theoretical point of view: there are several beautiful good characterization theorems for some restricted cases. The restriction to planar graphs, and in particular to planar grid graphs is both of practical and theoretical interest. Here we prove the NP-completeness of a planar case of the problem, settling an open question of Vygen [Vyg94]. This complements the good characterization theorem of Okamura and Seymour. Moreover, this result will be used in Section 3.3 to prove the NP-completeness of precoloring extension on unit interval graphs.

In the disjoint paths problem we are given a graph G and a set of source–destination pairs (s_1, t_1), (s_2, t_2), ..., (s_k, t_k) called the *terminals*, and we have to find k disjoint paths P_1, ..., P_k such that path P_i connects vertex s_i to vertex t_i. There are four basic variants of the problem: the graph can be directed or undirected, and we can require edge disjoint or vertex disjoint paths. The problem is often described in terms of a supply graph and a demand graph, as follows:

Disjoint Paths

Input: The *supply graph* G and the *demand graph* H on the same set of vertices.

Task: Find a path P_e in G for each $e \in E(H)$ such that these paths are pairwise disjoint and path P_e together with edge e forms a circuit.

The graphs G and H can have parallel edges but no loops. For vertex disjoint paths we allow their endpoints to be the same. In the directed version of the problem both G and H are directed. With a slight abuse of terminology, we say in the directed case that a demand $\overrightarrow{uv} \in H$ starts in v and ends in u (since the directed path satisfying this demand starts in v and ends in u). Moreover, given a solution of the disjoint paths problem, we identify a demand with the path satisfying it. That is, we say that "demand α uses supply edge e" instead of "the path satisfying demand α uses edge e". An undirected graph is called *Eulerian* if every vertex has even degree, and a directed graph is Eulerian if the indegree equals the outdegree at every vertex.

The disjoint paths problem and its variants were intensively studied, for an overview see [Fra90, Vyg94]. In particular, all four variants of the problem (directed/undirected, edge disjoint/vertex disjoint) are NP-complete, even when G is planar. In this paper we consider only the (directed and undirected) edge disjoint paths problem in the grid, thus henceforth disjoint means edge disjoint.

If there exist disjoint paths in $G(V, E)$ with the given endpoints, then every cut $(V', V \setminus V')$ has to contain at least as many edges from G as from H, otherwise there would be more demands crossing this cut than edges connecting V' and $V \setminus V'$. We say that the *cut criterion* holds for G and H if this is true for every cut $(V', V \setminus V')$. In general, the cut criterion is only a necessary condition, but in an important special case it is also sufficient:

Theorem 3.2.1 (Okamura and Seymour, 1981, [OS81]). *Assume that G is planar, undirected, $G + H$ is Eulerian, and every edge of H lies on the outer face of G. The edge disjoint paths problem has a solution if and only if the cut criterion is satisfied.*

A graph is a *grid graph* if it is a finite subgraph of the rectangular grid. A directed grid graph is a grid graph with the horizontal edges directed to the right and the vertical edges directed to the bottom. Clearly, every directed grid graph is acyclic. A *rectangle* is a grid graph with $n \times m$ nodes such that $v_{i,j}$ $(1 \le i \le n,\ 1 \le j \le m)$ is connected to $v_{i',j'}$ if and only if $|i - i'| = 1$ and $j = j'$, or $i = i'$ and $|j - j'| = 1$. The study of grid and rectangle graphs is motivated by applications in VLSI-layout.

The special case of Theorem 3.2.1 when G is a rectangle is investigated in [Fra82]. The requirement that the edges of H lie on the outer face of G cannot be dropped even in this restricted case: Figure 3.5 shows an example where $G + H$ is Eulerian, the cut criterion holds, but the terminals cannot be connected

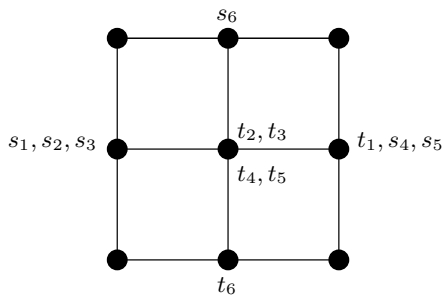


Figure 3.5: Theorem 3.2.1 does not hold when the terminals do not have to lie on the outer face.

by edge disjoint paths (it is left to the reader to verify these claims). In Section 3.2.1, we prove that the edge disjoint paths problem is NP-complete on rectangles even if $G + H$ is Eulerian. This answers an open question of Vygen [Vyg94]. Moreover, this also implies that (unless coNP = NP) a generalization of Theorem 3.2.1 cannot give a good characterization to the case when we drop the requirement that the terminals have to lie on the outer face.

There are several good characterization theorems in the literature [Fra90, Vyg94] for the case when G is planar, $G + H$ is Eulerian, and some additional constraint holds (as in Theorem 3.2.1). Previously no NP-completeness result was known for G planar and $G + H$ Eulerian. To the best of our knowledge, the only negative result for $G + H$ Eulerian is the theorem of Vygen [Vyg95] stating that the disjoint paths problem is NP-complete if $G + H$ is Eulerian, and G is an undirected (nonplanar) graph or a DAG.

In the directed case, Vygen proved that the edge disjoint paths problem is NP-complete even if the supply graph G is planar and acyclic [Vyg95] or even if G is a directed rectangle [Vyg94], and asked whether the problem remains NP-complete with the additional constraint that the graph $G + H$ is Eulerian. We settle this question by proving that the problem, similarly to the undirected version, is indeed NP-complete. This result will be used in Section 3.3 to show that precoloring extension is NP-complete for unit interval graphs.

3.2.1 The reduction

In this section we prove that the edge disjoint paths problem on directed and undirected rectangle graphs remains NP-complete even in the restricted case when $G+H$ is Eulerian. First we prove that the problem is NP-complete on directed grid graphs with $G + H$ Eulerian. Using standard techniques, this result is extended to rectangle graphs and undirected graphs.

The following observation will be useful:

Lemma 3.2.2 ([Vyg95]). *In the directed disjoint paths problem, if $G + H$ is Eulerian and G is acyclic, then every solution uses all the edges of G.*

Proof. Assume that a solution is given. Take a demand edge of H and delete from $G + H$ the directed circuit formed by the demand edge and its path in the solution. Continue this until the remaining graph contains no demand edges, then it is a subgraph of G. Since we deleted only directed circuits, it remains Eulerian, but the only Eulerian subgraph of the acyclic graph G is the empty graph with no edges, thus the solution used all the edges. ■

Proving the NP-completeness of a planar problem is usually done in one of two ways: either the reduction is from a planar problem (such as planar SAT, planar independent set etc.) or the reduction constructs a planar instance by locally replacing crossings with copies of some crossover gadget (as in

[GJS76] for planar graph coloring). Our reduction is none of these two types: there are crossings, but the global structure of the construction ensures that the crossings "behave nicely." This resembles the way [Vyg95] proves the NP-completeness of the disjoint paths problem on planar DAGs.

Theorem 3.2.3. *The edge disjoint paths problem is* NP*-complete on directed grid graphs, even if* $G + H$ *is Eulerian.*

Proof. The proof is by polynomial reduction from a restricted case of 1-in-3 SAT, where a formula is given in conjunctive normal form, and our task is to find a variable assignment such that in every clause of the formula, exactly one of the three literals is true. In *monotone* 1-in-3 SAT every literal is positive (not negated), and in the *cubic* version of the problem every variable occurs exactly three times. In [MR01] it is shown that monotone, cubic 1-in-3 SAT is NP-complete.

Let n be the number of variables in the given monotone, cubic 1-in-3 SAT formula, this obviously equals the number of clauses. It can be assumed that every clause contains three different literals. The reduction is of the component design type: we construct variable setting gadgets and satisfaction testing gadgets, and connect them in such a way that the disjoint paths problem has a solution if and only if the given formula is satisfiable (in 1-in-3 sense). The constructed graph G is a grid graph, and the construction ensures that $G + H$ is Eulerian in the resulting instance.

First we present how the gadgets are connected, the structure of the gadgets itself will be described later. Going diagonally from top left to bottom right, place a sequence of n copies of the variable setting gadget. The component corresponding to x_n is in the top left corner. Continue this sequence by n copies of the satisfaction testing gadget (see Figure 3.6). Denote by p_t the lower right vertex of the component corresponding to the tth clause, and let p_0 be the top left vertex of the component of the first clause. Three paths leave each variable gadget to the right and three to the bottom, they will be called the *right exits* and the *lower exits* of the gadget. The exits are numbered, the topmost right exit is the first right exit, and the leftmost lower exit is the first lower exit. Similarly, the satisfaction testing gadgets have three *upper entries* (the first is the leftmost) and three *left entries* (the first is the topmost). Assume that the literals in a clause are sorted, the variable of the first literal has the smallest index, i.e., as in the clause $(x_1 \vee x_2 \vee x_7)$. The occurrences of a variable are numbered in such a way that the first occurrence of the variable is in the clause with the *largest* index.

The components are connected as follows. If the ith occurrence ($i = 1, 2, 3$) of variable x_s is the jth literal ($j = 1, 2, 3$) in clause C_t, then connect the ith right exit of the component of x_s to the jth upper entry of the component of C_t, and similarly with lower exits and left entries. Each connection is a path in the grid consisting of several directed edges. The connections are done by first going to the right (below) and then to below (right), there is only one turn in each connection. There will be exactly $6n$ demands: if variable x_s appears in clause C_t, then there are two demands that start from the component of x_s and end in the component of C_t. (The exact location of the start and end vertices of the demands will be defined later.)

The connections described above can cross each other at a vertex, there may be several such crossings in the resulting grid graph. Given a solution of the disjoint paths problem, we call a vertex a *bad crossing*, if the demand entering this vertex from the left leaves to the bottom, and the demand entering from above leaves to the right. (Note that by Lemma 3.2.2, exactly two demands go through a crossing). We show by induction that a solution in this graph cannot contain a bad crossing. Clearly, there are no crossings to the left and above of the vertex p_0. Assume that there are no bad crossings to the left and above of the vertex p_{t-1}. Figure 3.7 shows the paths entering the component of clause C_t, the dashed lines show other possible paths that may cross these six paths. By the way the literals are ordered in the clause, the six paths entering a clause component do not cross each other (recall that the component of x_n is in the upper left corner). Furthermore, because of the way the occurrences of a variable are ordered, the paths leaving a variable component do not cross each other either.

By the induction hypotheses, the same demand goes through vertices c_1 and c_2, through vertices b_1 and b_2, through vertices a_1 and a_2, through vertices c'_1 and c'_2, and so on. For example, the demand going

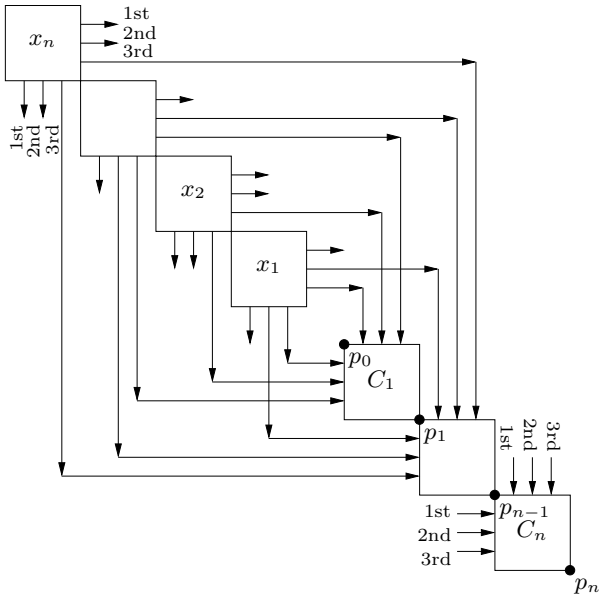


Figure 3.6: Overview of the reduction. The variable setting gadgets are on the left, three paths leave each of them to the right and to the bottom. These paths lead to the satisfaction testing gadgets below.

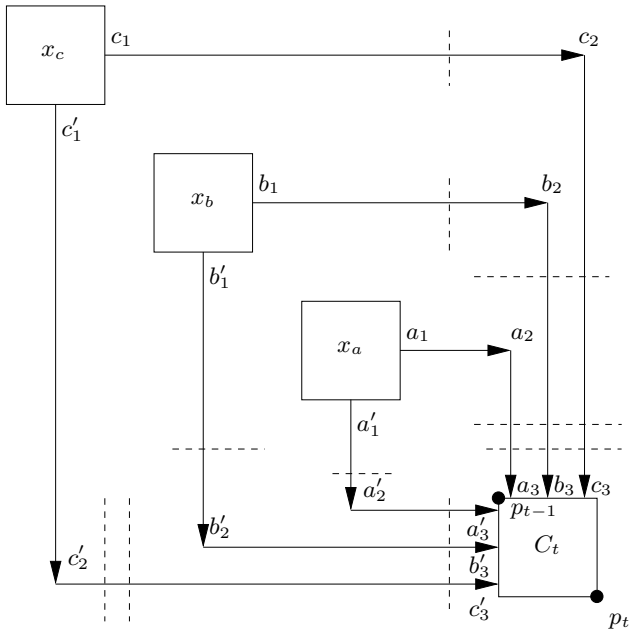


Figure 3.7: The paths entering a given satisfaction testing component.

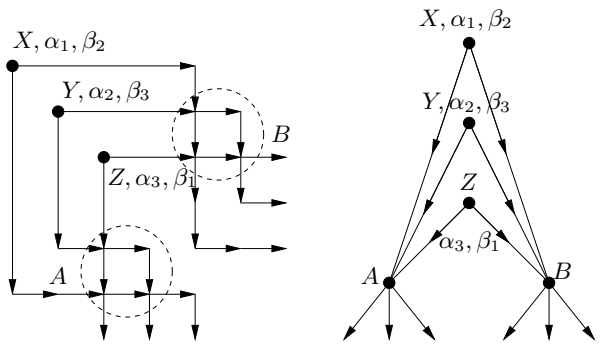


Figure 3.8: The variable setting gadget.

through c_1 can leave the path c_1c_2 only if there is a bad crossing on c_1c_2, but there are no bad crossings to the left and above of p_{t-1}. There are two demands that start in the component of x_c and have C_t as destination. They cannot leave x_c both to the right: only one of them can reach C_t through the path from c_2 to c_3, and there is no other way of reaching C_t without a bad crossing to the left of p_{t-1}. Similarly, the two demands cannot leave both to the bottom. Thus exactly one of the demands going to C_t leaves to the right and the other to the bottom, furthermore, these demands leave x_c through c_1 and c'_1. By a similar argument, this also holds for the components of x_b and x_a. Clearly, the demand going through c_1 and c_2 can reach C_t only through c_3, thus there are no bad crossings on the path from c_2 to c_3. The demand going through b_2 can reach C_t only through b_3 or c_3, but since c_3 is already used, only b_3 remains. Finally, the demand going through a_2 has to enter C_t in a_3. Therefore there are no bad crossings above C_t, and a similar argument shows that there are no bad crossings to the left of C_t. Thus there are no bad crossings to the left and above of p_t, which completes the induction.

Now we describe the gadgets used in the reduction. The *variable setting gadget* (Figure 3.8) has three output edges to the right, and three output edges to the bottom. On the right of the figure a simplified version of the gadget is shown, which is not a grid graph, just a planar DAG. The structure of the real gadget is the same, but in order to make it a grid graph some of the edges have to be twisted and the high-degree vertices A and B have to be split. We will show how the simplified version works, it is easy to show that the same holds for the real gadget.

Demands α_1, β_2 start in X, demands α_2, β_3 start in Y, and demands α_3, β_1 start in Z. The destination of demands α_i and β_i are in the clause component corresponding to the clause of the ith occurrence of the variable. We have seen that in every solution either α_i leaves the gadget to the right and β_i leaves to the bottom, or the opposite. However, more is true: either all of $\alpha_1, \alpha_2, \alpha_3$ leave to the right (through B) and $\beta_1, \beta_2, \beta_3$ leave to the bottom (through A), or the other way. To see this, first assume that α_1 uses $\overrightarrow{XA}$, then β_2 uses $\overrightarrow{XB}$. This implies that α_2 cannot go through B, thus α_2 uses $\overrightarrow{YA}$ and β_3 uses $\overrightarrow{YB}$. Demand α_3 cannot go through B, hence it uses $\overrightarrow{ZA}$ and β_1 uses $\overrightarrow{ZB}$. Thus $\alpha_1, \alpha_2, \alpha_3$ go through A, and $\beta_1, \beta_2, \beta_3$ go through B, what we had to show. By a similar argument, if α_1 uses $\overrightarrow{XB}$, we get that all three demands α_i go through B. Therefore in every solution of the disjoint paths problem, the component of x_s has two possible states: either the demands α_i leave to the bottom (we call this state "true") or they leave to the right ("false"). Recall that if the demands α_i leave to the bottom, then they reach their respective clause components from the left, while if they leave to the right, then they reach the clause components from the top.

The *satisfaction testing gadget* and its simplified equivalent is shown on Figure 3.9. The three paths that enter K correspond to the three paths that enter the gadget from the left, while the paths entering L correspond to those entering from the top. The gadget contains the endpoints of six demands corresponding

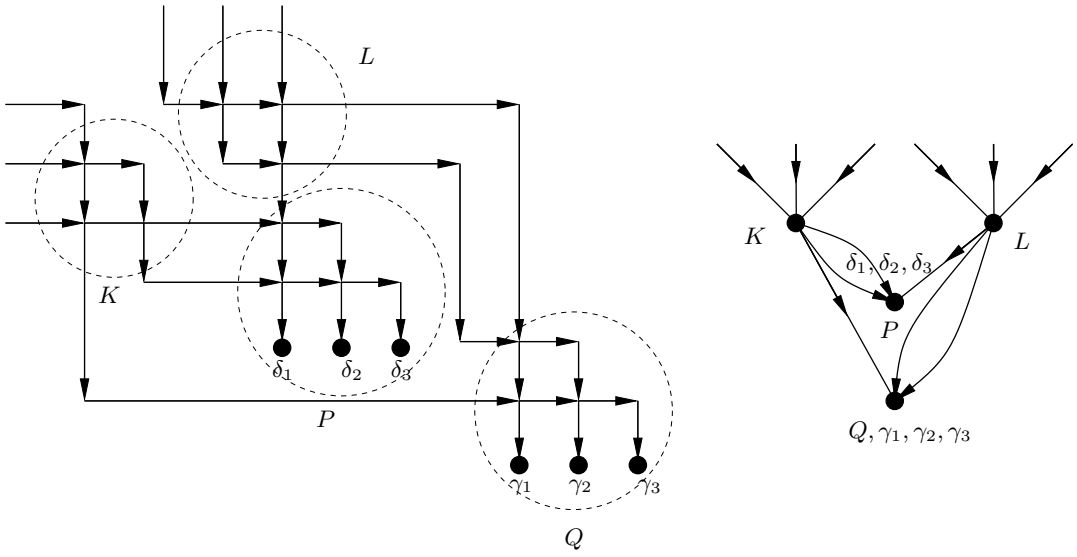


Figure 3.9: The satisfaction testing gadget.

to the three variables. Demands γ_j and δ_j start in the variable component corresponding to the jth literal of the clause. More precisely, if the jth literal of clause C_t is the ith occurrence of variable x_s, then demand α_i starting in gadget x_s is the same as demand γ_j terminating in gadget C_t, and β_i is the same as δ_j.

Vertex P is the endpoint of the three demands δ_1, δ_2, δ_3, and vertex Q is the endpoint of the demands γ_1, γ_2, γ_3. We have seen that in every solution, exactly one of γ_j and δ_j leaves the variable component to the right, the other one leaves to the bottom, hence exactly one of them enters the clause component from the top, the other one enters from the left. Furthermore, there is exactly one j such that γ_j enters from the left and δ_j enters from the top, for the remaining two $j' \neq j$, demand $\gamma_{j'}$ enters from the top and demand $\delta_{j'}$ enters from the left. To see this, notice that from K only one demand can reach Q and only two demands can reach P. Thus the satisfaction testing gadget effectively forces that exactly one of the three variable gadgets is in the state "true."

It can be easily verified that $G + H$ is Eulerian in the constructed instance. Given a solution to the disjoint paths problem, we can find a satisfying assignment of the formula: assign to the variable x_s "true" or "false" depending on the state of the gadget corresponding to x_s. By the construction, every clause will be satisfied (in 1-in-3 sense). On the other hand, given a satisfying variable assignment, we can find a solution to the disjoint paths problem: the values of the variables determine how the demands leave the variable setting gadgets and this can be extended to the whole graph. ■

It is noted in [Vyg94] that the disjoint paths problem is not easier in rectangle graphs than in general grid graphs: if we add a new edge $\overrightarrow{uv}$ to G and a new demand from u to v, then the new demand can reach v in the grid only using the new edge. Thus we can add new edges and demands until we get a full rectangle graph without changing the solvability of the instance. Clearly, $G + H$ remains Eulerian after adding the supply edge $\overrightarrow{uv}$ to G, and the demand edge $\overrightarrow{vu}$ to H.

Corollary 3.2.4. *The edge disjoint paths problem is* NP*-complete on directed rectangle graphs, even if* $G + H$ *is Eulerian.* ■

A reduction from the directed case to the undirected one was described by Vygen:

Lemma 3.2.5 ([Vyg95]). *If (G, H) is an instance of the directed edge disjoint paths problem, where G is acyclic, $G + H$ is Eulerian, and the undirected graphs G', H' result from neglecting the orientation of G, H, then every solution of (G, H) is also the solution of (G', H') and vice versa.*

Combining Corollary 3.2.4 and Lemma 3.2.5, we obtain the following corollary, settling another open question from [Vyg94]:

Corollary 3.2.6. *The undirected edge disjoint paths problem is* NP*-complete on rectangle graphs, even if $G + H$ is Eulerian.* ■

For technical reasons, we introduce the following variant of the disjoint paths problem. For every demand, not only the terminals are given, but here also the first and last edge of the path is also prescribed:

Directed Edge Disjoint Paths with Terminal Edges

Input: The supply graph G and the demand graph H on the same set of vertices (both of them directed), and for every edge $e \in H$, a pair of edges (s_e, t_e) of G.

Question: Find a path P_e in G for each $e \in E(H)$ such that these paths are edge disjoint, P_e and e form a directed circuit and the first/last edge of P_e is s_e, t_e, respectively.

As shown in the following theorem, this variant of the problem is NP-complete as well. This problem will be used in Section 3.3: it will be the basis of the reduction when proving the NP-completeness of precoloring extension for unit interval graphs.

Theorem 3.2.7. *The Directed Edge Disjoint Paths with Terminal Edges problem is* NP*-complete on rectangle graphs, even when $G + H$ is Eulerian.*

Proof. The reduction in Theorem 3.2.3 constructs grid graphs with the following additional properties:

- at most one demand ends in each vertex v,
- if a demand ends in v, then exactly one edge of G enters v,
- at most two demands start in each vertex u,
- if a demand stars in u, then no edge of G enters u.

If two demands α and β start at a vertex u, then we slightly modify G and H. Two new supply edges $\overrightarrow{xu}$ and $\overrightarrow{yu}$ are attached to u, there is place for these edges since no edge enters u in G. Demand graph H is modified such that the start vertex of demand α is set to x, the start vertex of β is set to y. Clearly, these modifications do not change the solvability of the instance, and G remains a grid graph. Moreover, $G + H$ remains Eulerian. Therefore we can assume that the instance has the following two properties as well:

- At most one demand starts from each vertex u,
- If a demand starts in u, then exactly one edge of G leaves u.

If these properties hold, then in every solution of the disjoint paths problem a demand going from u to v has to leave u on the unique edge leaving u, and has to enter v on the unique edge entering v. Therefore prescribing the first and the last edge of every demand does not change the problem. Thus we can conclude that the disjoint paths with terminal edges problem is NP-complete in grid graphs. Furthermore, when we add new edges to G and H to make G a rectangle (as described in the remark before Theorem 3.2.4), then obviously it can be prescribed that the first and the last edge of the new demand is the new edge, hence it follows that the problem is NP-complete on directed rectangles as well. ■

3.3 Unit interval graphs

A graph is an *interval graph* if it can be represented as the intersection graph of a set of intervals: a vertex corresponds to each interval, and two vertices are connected if the corresponding intervals have nonempty intersection. Furthermore, the graph is a *unit interval graph* if it can be represented by intervals of unit length, and it is a *proper interval graph* if it can be represented in such a way that no interval is properly contained in another. It can be shown that these two latter classes of graphs are the same [BW99], in fact they are exactly the interval graphs that are claw-free [Rob69] (contain no induced $K_{1,3}$). These interval graphs are also called *indifference graphs*.

Interval graph coloring arises in various applications including scheduling [BHT93] and single row VLSI routing [Rec92]. There is a simple greedy algorithm that colors an interval graph with minimum number of colors. However, Biró, Hujter and Tuza [BHT92] proved that the precoloring extension problem is NP-complete on interval graphs, even if every color is used at most twice in the precoloring (they also gave a polynomial time algorithm for the case where every color is used only once). In [HT96] they asked what is the complexity of the precoloring extension problem in the more restricted case of unit interval graphs. In Section 3.3.1, we prove that this problem is also NP-complete. The proof is by reduction from a disjoint paths problem whose NP-completeness was proved in Section 3.2.

3.3.1 The reduction

In [BHT92] the NP-completeness of precoloring extension on interval graphs is proved by a reduction from circular arc graph coloring. A similar reduction is possible from *proper* circular arc coloring to the precoloring extension of *proper* interval graphs, but the analogy doesn't help here, because proper circular arc coloring can be done in polynomial time [OBB81, TT85]. In this section, we follow a different path: the NP-completeness of precoloring extension on proper interval graphs is proved by reduction from a planar disjoint paths problem investigated in Section 3.2.

An important idea of the proof is demonstrated on Figure 3.10. Every interval graph in this section is assumed to be open. In any k-coloring of the intervals in (a), for all i, interval $I_{1,i}$ has the same color as $I_{0,i}$: interval $I_{1,0}$ must receive the only color not used by $I_{0,1}, I_{0,2}, \ldots, I_{0,k-1}$; interval $I_{1,1}$ must receive the color not used by $I_{1,0}, I_{0,2}, I_{0,3}, \ldots, I_{0,k-1}$, and so on. In case (b), the intervals are slightly modified. If all the $I_{0,i}$ intervals are colored, then there are two possibilities: either the color of $I_{1,i}$ is the same as the color of $I_{0,i}$ for $i = 0, \ldots, k-1$, or we swap the colors of $I_{1,1}$ and $I_{1,2}$. Blocks of type (a) and (b) will be the building blocks of our reduction.

Theorem 3.3.1. *The precoloring extension problem is* NP*-complete on proper interval graphs.*

Proof. The reduction is from the Eulerian directed edge disjoint paths with terminal edges problem on rectangle graphs, whose NP-completeness was shown in Theorem 3.2.7. First we modify the given rectangle graph G. As in the remark before Theorem 3.2.4, new edges are added to the rectangle $KLMN$ to obtain the shape shown on Figure 3.11, without changing the solvability of the instance. Hereinafter it is assumed that G has such a form. The entire G is contained between the two diagonal lines X and Y, the vertices on X have outdegree 1, the vertices on Y have indegree 1 and the vertices of G between X and Y are Eulerian. If the rectangle $KLMN$ contains $r \times s$ vertices, then there are $m = r + s$ vertices on both X and Y, and every directed path from a vertex of X to a vertex of Y has length m. Now consider the parallel diagonal lines A_0, A_1, ..., A_{m-1} as shown on the figure, and denote by E_i the set of edges intersected by A_i. Clearly this forms a partition of the edges, and every set E_i has size m. Let $E_i = \{e_{i,0}, \ldots, e_{i,m-1}\}$, ordered in such a way that $e_{i,0}$ is the lower left edge.

We can assume that H is a DAG, otherwise there would be no solution, since G is acyclic. Exactly one demand starts from each vertex on line X, exactly one demand terminates at each vertex on Y, and the indegree equals the outdegree in every other vertex of H, this follows from $G + H$ Eulerian. From these facts, it is easy to see that H can be decomposed into m disjoint paths $D_1, \ldots, D_m$ such that every

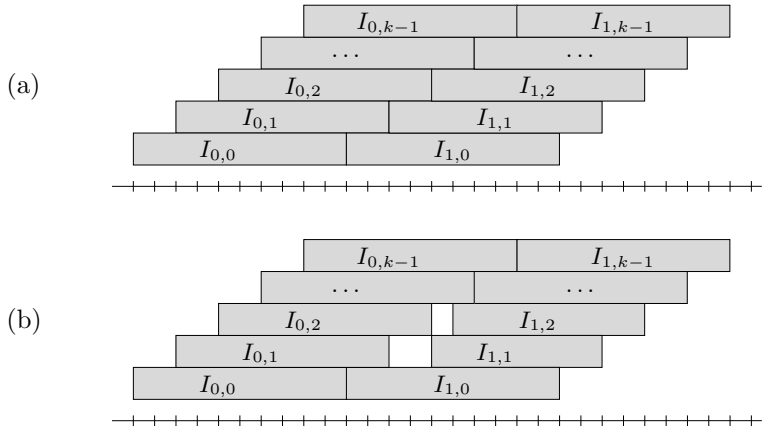


Figure 3.10: (a) In every k-coloring c of the (open) intervals, $I_{0,i}$ and $I_{1,i}$ receive the same color for $0 \leq i \leq k-1$ (b) In every k-coloring c of the intervals, $c(I_{0,i}) = c(I_{1,i})$ for $i \neq 1, 2$ and either $c(I_{0,1}) = c(I_{1,1})$, $c(I_{0,2}) = c(I_{1,2})$ or $c(I_{0,1}) = c(I_{1,2})$, $c(I_{0,2}) = c(I_{1,1})$ holds.

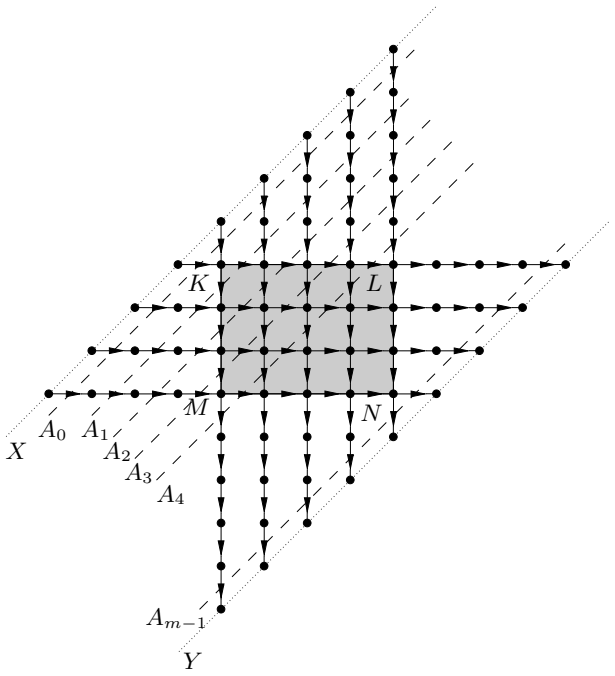


Figure 3.11: Partitioning the edges of the extended grid ($r = 4$, $s = 5$).

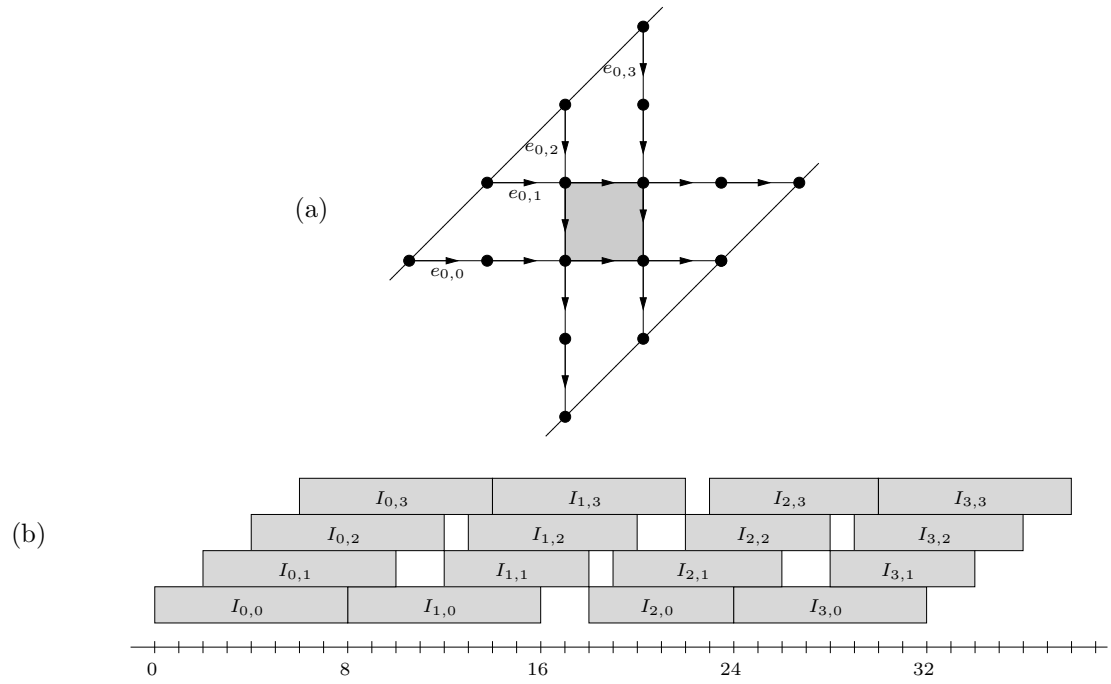


Figure 3.12: An example of the reduction with $r = s = 2$: (a) the grid graph, (b) the corresponding proper interval graph.

path connects a vertex on X with a vertex on Y. We assign a color to each demand: if demand α is in D_i, then give the color i to α.

Based on the disjoint paths problem, we define a set of intervals and a precoloring. Every interval $I_{i,j}$ corresponds to an edge $e_{i,j} \in E_i$ of the supply graph G. Let $v_{i,j}$ be the tail vertex of $e_{i,j}$, and denote by $\delta_G(v_{i,j})$ the outdegree of $v_{i,j}$ in G. The intervals $I_{i,j}$ $(0 \le i \le m-1, 0 \le j \le m-1)$ are defined as follows (see Figure 3.12):

$$I_{i,j} = \begin{cases} \big(2(im+j), 2(im+j)+2m\big) & \text{if } \delta_G(v_{i,j}) = 1, \\ \big(2(im+j)+2, 2(im+j)+2m\big) & \text{if } \delta_G(v_{i,j}) = 2 \text{ and } e_{i,j} \text{ is vertical}, \\ \big(2(im+j)+1, 2(im+j)+2m\big) & \text{if } \delta_G(v_{i,j}) = 2 \text{ and } e_{i,j} \text{ is horizontal}. \end{cases}$$

Notice that the intervals are open, hence two intervals that share only an endpoint do not intersect.

If the prescribed start edge and end edge of a demand with color c is e' and e'', then precolor the intervals corresponding to e' and e'' with color c. This assignment is well defined, since it can be assumed that the start and end edges of the demands are different, otherwise it is trivial that the problem has no solution. This completes the description of the reduction, we claim that the precoloring of the constructed interval graph can be extended to a coloring with m colors if and only if the disjoint paths problem has a solution.

First we observe certain properties of the intervals. Let $I_i = \{I_{i,0}, \ldots, I_{i,m-1}\}$, that is, the set of intervals corresponding to E_i. The set I_i forms a clique in the graph, and the elements of I_i and $I_{i'}$ are not intersecting if $i' \ge i+2$. The interval $I_{i,j}$ does not intersect $I_{i-1,j'}$ for $j' \le j$, and it does intersect $I_{i-1,j'}$ for $j' > j+1$. It may or may not intersect $I_{i-1,j+1}$.

Assume that $P_1, \ldots, P_n$ is a solution of the disjoint paths problem. If an edge $e_{i,j}$ is used by a demand with color c, then color the edge $e_{i,j}$ and the corresponding interval $I_{i,j}$ with color c. Since by Lemma 3.2.2

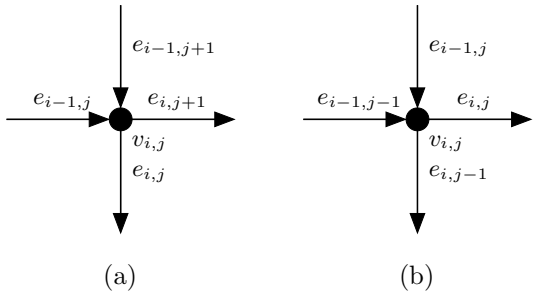


Figure 3.13: The edges incident to the tail of $e_{i,j}$. (a) $e_{i,j}$ is vertical (b) $e_{i,j}$ is horizontal

every edge of the graph is used by a demand, every interval receives a color. Furthermore, the demands use the prescribed start and end edges, and so this coloring is compatible with the precoloring given above. Notice that the set of edges in the grid graph that receive the color c forms a directed path from a vertex on X to a vertex on Y. Thus all m colors appear on the intervals in I_i, every interval has different color in this set.

It has to be shown that this coloring is proper. By the observations made above, it is sufficient to verify that two intersecting intervals $I_{i,j}$ and $I_{i-1,j'}$ do not have the same color. Since the edges having color c form a path, if $e_{i-1,j'}$ and $e_{i,j}$ have the same color, then the head of $e_{i-1,j'}$ and the tail of $e_{i,j}$ must be the same vertex $v_{i,j}$. Assume first that $\delta_G(v_{i,j}) = 1$, then $j = j'$, which implies that $I_{i,j}$ and $I_{i-1,j'}$ are not intersecting. For the case $\delta_G(v_{i,j}) = 2$, it will be useful to refer to Figure 3.13. If $\delta_G(v_{i,j}) = 2$ and $e_{i,j}$ is vertical, then $e_{i,j+1}$ is horizontal and its tail is also $v_{i,j}$ (see Figure 3.13a). Moreover, in this case $e_{i-1,j}$ is horizontal, $e_{i-1,j+1}$ is vertical, and $v_{i,j}$ is the head of both edges. Therefore if $\delta_G(v_{i,j}) = 2$ and $e_{i,j}$ is vertical, then $j' = j$ or $j' = j + 1$, which implies that the right endpoint of $I_{i-1,j'}$ is not greater than $2((i-1)m + j + 1) + 2m = 2(im + j) + 2$, the left endpoint of $I_{i,j}$. If $e_{i,j}$ is horizontal, then $j' = j$ or $j' = j - 1$ (see Figure 3.13b), hence intervals $I_{i-1,j'}$ and $I_{i,j}$ are clearly not intersecting.

On the other hand, assume that there is a proper extension of the precoloring with m colors. Color every edge $e_{i,j}$ of the grid graph with the color assigned to the corresponding interval $I_{i,j}$. First we prove that the set of edges having color c forms a directed path R_c in the graph. Since the intervals in I_i have different colors, every one of the m colors appears exactly once on the edges in E_i. Thus it is sufficient to prove that the tail $v_{i,j}$ of the unique edge $e_{i,j} \in E_i$ having color c is the same as the head of the unique edge $e_{i-1,j'} \in E_{i-1}$ having color c.

Assume first that $\delta_G(v_{i,j}) = 1$, then we have to show that $j = j'$. Denote by $x = 2(im + j)$, the left endpoint of $I_{i,j}$, which is also the right endpoint of $I_{i-1,j}$ (as an example, consider interval $I_{1,3}$ on Figure 3.12b). If $j' > j$, then $I_{i-1,j'}$ and $I_{i,j}$ intersect (both of them contain $x + \epsilon$), which contradicts the assumption that $I_{i,j}$ and $I_{i-1,j'}$ have the the same color. Assume therefore that $j' < j$. It is clear from the construction that the left endpoint of every interval $I_{i,1}, \ldots, I_{i,j-1}$ is strictly smaller than x (it is not possible that $\delta_G(v_{i,j-1}) = 2$ and $e_{i,j-1}$ is vertical, since that would imply $v_{i,j-1} = v_{i,j}$ and $\delta_G(v_{i,j}) = 2$). The right endpoint of every interval $I_{i-1,j}, \ldots, I_{i-1,m-1}$ is not smaller than x, thus $\{I_{i,0}, \ldots, I_{i,j-1}, I_{i-1,j}, I_{i-1,j+1}, \ldots, I_{i-1,m-1}\}$ is a clique of size m in the interval graph, since they all contain $x - \epsilon$. Now $I_{i,0}, \ldots, I_{i,j-1}$ intersect $I_{i,j}$, and $I_{i-1,j}, \ldots I_{i-1,m-1}$ intersect $I_{i,j'}$, thus color c cannot appear in this clique, a contradiction.

Now assume that $\delta_G(e_{i,j}) = 2$ and $e_{i,j}$ is vertical, we have to show that $j' = j$ or $j' = j + 1$ holds (see for example $I_{1,1}$ on Figure 3.12b). If $j' > j + 1$, then $I_{i-1,j'}$ intersects $I_{i,j}$, a contradiction. Assume therefore that $j' < j$ and let $y = 2(im + j)$, the right endpoint of $I_{i-1,j}$. It can be verified that $\{I_{i,0}, \ldots, I_{i,j-1}, I_{i-1,j}, \ldots, I_{i-1,m-1}\}$ is a clique of size m, since all of them contain $y - \epsilon$. Color c cannot

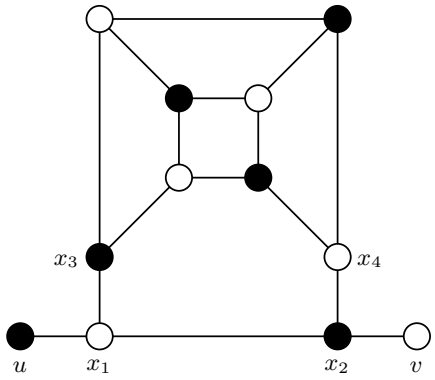


Figure 3.14: The gadget used to replace those edges that have 2 element lists.

appear on $I_{i,0}, \ldots, I_{i,j-1}$ because of $I_{i,j}$, and it cannot appear on $I_{i-1,j}, \ldots, I_{i-1,m-1}$ because of $I_{i-1,j'}$. Thus there is a clique of size m without color c, a contradiction.

If $\delta_G(e_{i,j}) = 2$ and $e_{i,j}$ is horizontal, then we have to show that j' is either j or $j-1$ (see for example $I_{3,2}$ on Figure 3.12b). If $j' \geq j+1$, then $I_{i,j}$ intersects $I_{i-1,j'}$, therefore it can be assumed that $j' < j-1$. Let $z = 2((i-1)m + j - 1) + 2m$, the right endpoint of $I_{i-1,j-1}$. Point $z - \epsilon$ is contained in each of $I_{i,0}$, $\ldots$, $I_{i,j-2}$, $I_{i-1,j-1}$, $\ldots$, $I_{i-1,m-1}$, hence they form a clique of size m. However, intervals $I_{i-1,j'}$ and $I_{i,j}$ forbid the use of color c on this clique, a contradiction.

We have shown that the set of edges with color c are contained in a path R_c. Because of the precoloring, the path R_c goes through the prescribed start and end edges of every demand with color c. Furthermore, since the demands with color c correspond to a directed path D_c in H, all these demands can be satisfied using only the edges of R_c, without two demands using the same edge. Thus there is solution to the disjoint path problem, proving the correctness of the reduction.

Since the precoloring extension problem is obviously in NP and the reduction above can be done in polynomial time, we have proved that it is NP-complete on unit interval graphs. ■

3.4 Complexity of edge precoloring extension

In this section we consider the edge coloring version of precoloring extension. Colbourn [Col84] has shown that it is NP-complete to decide whether a partially filled matrix can be completed to a full Latin square. This result is equivalent to saying that edge precoloring extension is NP-complete for complete bipartite graphs. Easton and Parker [EP01], and independently Fiala [Fia03] have shown that edge precoloring extension is NP-complete for bipartite graphs of maximum degree 3. Here we strengthen this result by showing that the problem remains NP-complete for planar 3-regular bipartite graphs. The proof is by reduction from the list edge coloring problem (Theorem 2.1.2). We will use this result in Section 5.2 to prove that minimum sum edge coloring is NP-hard for planar bipartite graphs.

Theorem 3.4.1. *Precoloring extension is* NP*-complete on the edges of planar* 3*-regular bipartite graphs.*

Proof. We reduce list edge coloring to precoloring extension as follows. By the proof of Theorem 2.1.2, it can be assumed that only the three colors 1, 2, 3 appear in the lists. If edge uv has a 2 element list, say color i is not allowed ($1 \leq i \leq 3$), then we replace uv by the gadget shown on Figure 3.14. The edges x_1x_3 and x_2x_4 are precolored with color i.

We claim that the list edge coloring problem has a solution in the original graph G if and only if the precoloring in the constructed graph G' can be extended to a proper 3-coloring. First, a coloring of G can

be used to obtain a precoloring extension of G': if edge uv has color c, then the gadget corresponding to uv can be colored such that both ux_1 and x_2v receive color c. On the other hand, in every 3-coloring of G' edges ux_1 and x_2v receive the same color, a color different from the forbidden color i of edge uv: since both x_1x_3 and x_2x_4 are precolored to i, only two colors are available for edges ux_1, x_1x_2, x_2v. Therefore a precoloring extension of G' determines a list coloring of G. ■

In a similar manner, we strengthen Theorem 2.1.4 by showing that for outerplanar graphs not only list edge coloring, but even its special case edge precoloring extension is NP-complete.

Theorem 3.4.2. *Precoloring extension is* NP*-complete on the edges of bipartite outerplanar graphs.*

Proof. The proof is by reduction from the list edge coloring problem. Consider a bipartite outerplanar graph constructed by the reduction of Theorem 2.1.4. Notice that every edge lies on the outer face. Let the color set C be the union of the lists. If the list of an edge xy contains 2 colors, then this edge is replaced as follows. We add two new vertices x', y', and 3 new edges xx', $x'y'$, $y'y$. Furthermore, we attach $|C|-2$ new edges $x'x'_1, \ldots, x'x'_{|C|-2}$ to vertex x', and another $|C|-2$ new edges $y'y'_1, \ldots, y'y'_{|C|-2}$ to y'. Denote by $c_1, \ldots, c_{|C|-2}$ the $|C|-2$ colors in C that do not appear in the list of xy. The edges $x'x'_i$ and $y'y'_i$ are precolored with color c_i. These precolored edges ensure that in every edge coloring xx', $x'y'$, $y'y$ receive a color from the list of xy, which implies that the color of xx' and $y'y$ is the same. Therefore edges xx' and yy' effectively act as a single edge with the same list as xy. Moreover, since xy is on the outer face, the graphs remains outerplanar and bipartite.

Consider an edge vu_j having list size 3 (see the proof of Theorem 2.1.4). Here we use the fact that u_j has degree 1. We attach $|C|-3$ new edges $u_ju_{j,1}, \ldots, u_ju_{j,|C|-3}$ to u_j, and precolor them with the colors not in the list of vu_j. It is clear that now edge vu_j has to receive a color from the list of vu_j. Therefore we have constructed an instance of the precoloring extension problem that has a solution if and only if the list edge coloring problem has a solution. ■

With some further work, we can prove that edge precoloring extension is NP-complete also for series-parallel graphs:

Corollary 3.4.3. *Precoloring extension is* NP*-complete on the edges of bipartite series-parallel graphs.*

Proof. We show how to make a bipartite outerplanar graph 2-connected without changing the solvability of the precoloring extension instance. Together with Theorem 3.4.2 and the fact that 2-connected outerplanar graphs are series-parallel, this proves that precoloring edge extension is NP-complete for series-parallel graphs.

The graph is outerplanar, hence by traversing the boundary of the outer face we visit every vertex at least once. Since there can be cutvertices in the graph, there might be vertices that are visited more than once. Let $v_0, v_1, \ldots, v_{n-1}$ the order of vertices as they are first encountered while traversing the boundary of the outer face. We add new vertices and edges to the graph to make it Hamiltonian, and therefore 2-connected. If v_i and v_{i+1} (indices are taken modulo n) belong to the same bipartition class, then add a new vertex w_i and two new edges v_iw_i and w_iv_{i+1}. Edge v_iw_i is precolored with a new color α, and w_iv_{i+1} is precolored with a new color β. If v_i and v_{i+1} are in different classes, then add two new vertices w'_i, w''_i, and three edges $v_iw'_i$ (precolored to α), $w'_iw''_i$ (not precolored), w''_iv_{i+1} (precolored to β). It can be shown that the graph remains outerplanar after these modifications. The two new colors α and β appear at every vertex of the original graph, hence they cannot be used for the original edges. Thus the solvability of the instance did not change. ■

CHAPTER 4

Minimum sum coloring

For a long time I limited myself to one color—as a form of discipline.
Pablo Picasso (1881–1973)

Given a coloring of the vertices of a graph G, the *sum* of the coloring is the sum of the colors assigned to the vertices (here we assume that the colors are the positive integers). The *chromatic sum* $\Sigma(G)$ of G is the smallest sum that can be achieved by any proper coloring of G. In the *minimum sum coloring* problem we have to find a coloring of G with sum $\Sigma(G)$.

An application of minimum sum coloring is scheduling of dependent jobs with the goal of minimizing the sum of completion times. Assume that we have a set of jobs, each requires a unit amount of time. There are pairs of jobs that cannot be performed at the same time because they require a shared resource, or they interfere in some other way. The problem can be interpreted as finding a coloring of the *conflict graph*, where the vertices are the jobs and two vertices are connected by an edge if the corresponding two jobs cannot be performed simultaneously. Each color corresponds to a unit-length time slot. A coloring of the conflict graph gives a possible scheduling of the jobs: if a vertex receives color i, then the corresponding job can be done in the ith time slot. The coloring ensures that interfering jobs are not scheduled to the same time slot.

By minimizing the number of colors used on the conflict graph, we can minimize the length of the schedule, that is, we minimize the maximum of the completion times. This objective favors the system: we want that the collection of jobs is finished as quickly as possible, but we do not care when the individual jobs are finished. This approach is justified for example if the jobs are only substeps of a larger job. However, if the jobs are independent projects, possibly belonging to different users, then it can be advantageous to find a schedule that is longer, but most of the jobs are completed earlier. Minimizing the average completion times of the jobs (or equivalently, the sum of the completion times) is a well-studied goal in scheduling theory. By minimizing the sum of the coloring of the conflict graph, we find a schedule that minimizes the sum of the completion times.

Minimum sum coloring has a completely different application in VLSI design. In the *single row routing* problem (see Figure 4.1) there are terminals on a straight line, and some of these terminals have to be connected by wires. A *net* is a subset of the terminals that have to be connected: all the terminals in a net have to be connected, and terminals from different nets should not be connected. To simplify the presentation, here we assume that every net contains exactly two terminals. In the 2-*layer Manhattan model* the wires can be routed in two layers, but in one of the layers the wires can go only horizontally,

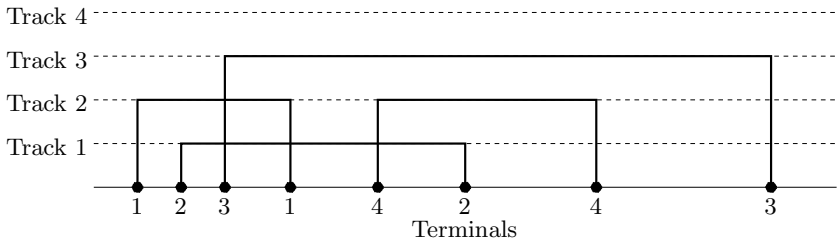


Figure 4.1: Single row routing problem.

in the other layer only vertically. Since they are on different layers, a horizontal and a vertical wire segment can cross each other without actually being connected. Moreover, in *dogleg-free routing* we also require that the wires connecting a net can contain only one horizontal wire segment. This means that the wires connecting a net consist of a horizontal segment going from the first terminal in the net to the last terminal, and vertical wire segments connecting the horizontal segment to the terminals themselves. Thus the only freedom we have in the routing is the selection of the vertical position for each horizontal segment. The horizontal segments cannot be placed at arbitrary positions, but only at the predefined *tracks*. The tracks are parallel horizontal lines at unit distance from each other.

The single row routing problem can be turned into a graph coloring problem the following way. The conflict graph contains one vertex for each net, and two nets are connected if their horizontal wire segments cannot be put on the same track (because they would overlap). A coloring of the conflict graph gives a possible way of routing: if a net receives the ith color, then use the ith track for the horizontal segment when connecting the two terminals in the net. It is easy to see that the conflict graph is always an interval graph. Therefore minimizing the number of tracks used is equivalent to finding the chromatic number of an interval graph, which is linear time solvable. In certain applications, however, it is more important to minimize the total wire length than to minimize the number of tracks. The total length of the horizontal wire segments is the same in every routing, and the total length of the vertical segments is proportional to the sum of track numbers assigned to the nets (assuming that track 1 is the track closest to the terminals). Therefore minimizing the total wire length is equivalent to minimizing the sum of the coloring in an interval graph. Szkaliczki [Szk99] has shown that minimum sum coloring interval graphs is NP-hard (a simpler proof can be found in [Mar03b]). A 2-approximation algorithm was presented in [NSS99], which was recently improved to a 1.796-approximation [HKS03].

Minimum sum coloring was introduced independently by Kubicka [Kub89] and by Supowit [Sup87]. In [KS89] it is shown that the problem is NP-hard in general, but polynomial time solvable for trees. The dynamic programming algorithm for trees can be extended to partial k-trees [Jan97]. Approximation algorithms were given for several graph classes: the minimum sum can be 4-approximated in perfect graphs [BNBH$^+$98], 1.796-approximated in interval graphs [HKS03], and 27/26-approximated in bipartite graphs [GJKM02]. Very recently, [GHKS04] gave a 3.591-approximation for perfect graphs. For further complexity results and approximation algorithms, see [BNBH$^+$98, BNK98, GJKM02, Sal03].

One can analogously define the edge coloring version of minimum sum coloring:

Minimum Sum Edge Coloring

Input: A graph $G(V, E)$ and an integer C.

Output: An edge coloring ψ: $E \to \mathbb{N}$ such that if e_1 and e_2 have a common vertex, then $\psi(e_1) \neq \psi(e_2)$.

Goal: Minimize $\Sigma'_\psi(E) = \sum_{e \in E} \psi(e)$, the *sum* of the coloring.

Minimum sum edge coloring can be solved in polynomial time for trees [GK00, Sal03, ZN04], but Giaro and Kubale [GK00] proved that it is NP-hard for bipartite graphs with maximum degree 3. Section 4.2.1 improves on this result by showing that the problem remains NP-hard for planar bipartite graphs.

The algorithm of [HKS03] can be used to find a 1.796-approximation of minimum sum coloring in every class of graphs where the maximum induced k-colorable subgraph problem is polynomial-time solvable. The line graphs of bipartite graphs form such a class, thus there is a 1.796-approximation algorithm for minimum sum edge coloring on bipartite graphs. For general graphs, [BNBH$^+$98] gives a 2-approximation. Section 4.2.2 proves that the problem is APX-hard for bipartite graphs, hence these constant factor approximations cannot be improved to a polynomial time approximation scheme (PTAS) for bipartite graphs.

The minimum sum edge coloring problem can be solved in polynomial time for trees [GK00, Sal03, ZN04] by a dynamic programming algorithm that uses weighted bipartite matching as a subroutine. In most cases, when a problem can be solved for trees by dynamic programming, then this easily generalizes to partial k-trees (see Appendix A.1). Here the situation is different. As shown in [Sal00], minimum sum edge coloring can be solved for partial k-trees in polynomial time if every vertex of the graph has bounded degree, but the complexity of the problem for arbitrary degrees remained open. In Section 4.3 we show that minimum sum edge coloring is NP-complete for partial 2-trees if there is no bound on the maximum degree.

The *chromatic strength* $s(G)$ of a graph G is the smallest number of colors that is required by a minimum sum coloring of the vertices. In Section 4.4 we determine the complexity of the question "Is $s(G) \leq k$?" for every k: it is coNP-complete for $k = 2$ (Theorem 4.4.3) and Θ_2^p-complete for $k \geq 3$ (Corollary 4.4.13). This improves on previous results of [Sal03], where it is shown that the question is NP-hard for $k \geq 3$, and the complexity of the case $k = 2$ was left as an open question.

We also study the complexity of the edge coloring version of the problem, with analogous definitions for the edge sum and the chromatic edge strength $s'(G)$. We show that for every $k \geq 3$, it is Θ_2^p-complete to decide whether $s'(G) \leq k$ holds (Corollary 4.4.12). As a first step of the proof, we present graphs for every $r \geq 3$ with chromatic index r and edge strength $r+1$. Such graph were known before only for $r = 4$ and for odd r greater than 3. Hajiabolhassan et al. [HMT00] asks as an open question to characterize those graphs where $s'(G) \neq \chi'(G)$. It follows from our results that we cannot hope for a nontrivial (NP or coNP) characterization of such graphs.

The results of Section 4.2.1 and Section 4.3 will appear in [Mar04c]. Section 4.4 will appear in [Mar04a].

4.1 Minimum sum edge coloring

In this section we introduce notations and new parameters, which will be used throughout this chapter. Let ψ be an edge coloring of $G(V, E)$, and let E_v be the set of edges incident to vertex v. For every $v \in V$, let $\Sigma'_\psi(v) = \sum_{e \in E_v} \psi(e)$ be the *sum* of v, and for a subset $V' \subseteq V$, let $\Sigma'_\psi(V') = \sum_{v \in V'} \Sigma'_\psi(v)$. Clearly, $\Sigma'_\psi(V) = 2\Sigma'_\psi(G)$, therefore minimizing $\Sigma'_\psi(V)$ is equivalent to minimizing $\Sigma'_\psi(G)$.

The degree of vertex v is denoted by $d(v) = |E_v|$. For every vertex v, let the *lower bound* of v be $\ell(v) = \sum_{i=1}^{d(v)} i = d(v)(d(v)+1)/2$, and for a set of vertices $V' \subseteq V$, let $\ell(V') = \sum_{v \in V'} \ell(v)$. Since $\Sigma'_\psi(v)$ is the sum of $d(v)$ distinct positive integers, thus $\Sigma'_\psi(v) \geq \ell(v)$ in every proper coloring ψ. Let $\epsilon_\psi(v) = \Sigma'_\psi(v) - \ell(v) \geq 0$ be the *error* of vertex v in coloring ψ. For $V' \subseteq V$ we define $\epsilon_\psi(V') = \sum_{v \in V'} \epsilon_\psi(v)$, and call $\epsilon_\psi(V)$ the *error of coloring* ψ. The error is always non-negative: $\Sigma'_\psi(V) \geq \ell(V)$, hence $\epsilon_\psi(V) = \Sigma'_\psi(V) - \ell(V) \geq 0$. Notice that $\epsilon_\psi(V)$ has the same parity for every coloring ψ. Minimizing the error of the coloring is clearly equivalent to minimizing the sum of the coloring. In particular, if ψ is a *zero error* coloring, that is, $\epsilon_\psi(V) = 0$, then ψ is a minimum sum coloring of G. In a zero error coloring, the edges incident to vertex v are colored with the colors $1, 2, \ldots, d(v)$.

However, in general, G does not necessarily have a zero error coloring. For every $V' \subseteq V$, the *error of* V' is $\epsilon(V') = \min_\psi \epsilon_\psi(V')$, the smallest error V' can have in a proper coloring of G. (Notice that

$\epsilon(V') = \sum_{v \in V'} \epsilon(\{v\})$ does not always hold, in fact, $\epsilon(\{v\}) = 0$ for every $v \in V$). Deciding whether G has a zero error coloring is a special case of the minimum sum edge coloring problem. It might be worth pointing out that finding a zero error coloring is very different from finding a minimum sum coloring: zero error is a local constraint on the coloring (every vertex has to have zero error), while minimizing the sum is a global constraint. In Section 4.2, when we prove the NP-completeness of minimum sum edge coloring for planar bipartite graphs, we will utilize this connection to reduce a local problem, edge precoloring extension, to the global problem of minimizing the sum.

Quasigraphs. Parallel edges are not allowed for the graphs considered here. However, for convenience we extend the problem by introducing *half-loops*. Following [Lov97], a half-loop is a loop that contributes only 1 to the degree of its end vertex. Every vertex has at most one half-loop. If a graph is allowed to have at most one half-loop at every vertex, then it will be called a *quasigraph*. In a quasigraph, the sum of an edge coloring is defined to be the sum of the color of the edges plus *half* the sum of the color of the half-loops, therefore the sum of a quasigraph is not necessarily integer. However, the error of a coloring is always integer, and with these definitions it remains true that the sum of the vertices is twice the sum of the edges. A quasigraph will be called bipartite if removing the half-loops gives a bipartite graph.

The following observation shows that allowing half-loops does not make determining the minimum edge sum or the chromatic edge strength more difficult:

Proposition 4.1.1. *Given a quasigraph G, one can create in polynomial time a graph G' such that $\Sigma'(G') = 2\Sigma'(G)$ and $s'(G') = s'(G)$.*

Proof. To obtain G', take two disjoint copies G_1, G_2 of G and remove every half-loop. If there was a half-loop at v in G, then add an edge v_1v_2 to G', where v_1 and v_2 are the vertices corresponding to v in G_1 and G_2, respectively. In graph G', to every edge give the color of the corresponding edge in G. If the sum of the coloring in G was S, then we obtain a coloring in G' with sum $2S$: two edges of G' correspond to every edge of G, but only one edge corresponds to every half-loop of G.

On the other hand, one can show that if G' has a k-coloring with sum S, than G has a k-coloring with sum at most $S/2$. The edges of G' can be partitioned into three sets E_1, E_2, E': set E_i contains the edges induced by G_i $(i = 1, 2)$, and E' contains the edges corresponding to the half-loops. It can be assumed that every edge in E_1 has the same color as its counterpart in E_2: otherwise if the sum of the edges in, say, E_1 is not greater than the sum of the edges in E_2, then recoloring every edge in E_2 with the color of its pair in E_1 results in a proper coloring without increasing the sum. Now color every edge of G with the color of its two corresponding edges in G', and color every half-loop with the color of its corresponding edge in G'. The sum of this coloring is exactly half the sum of the coloring of G'. ■

Therefore finding a minimum sum edge coloring on the quasigraph G is the same problem as finding a minimum sum edge coloring on the corresponding graph G'. In particular, the edge strength of G and G' are the same. In Section 4.4.4, we show that it is Θ_2^p-complete to determine the edge strength of a quasigraph. By the above construction, Θ_2^p-completeness follows for ordinary simple graphs as well.

Notice that if G is bipartite, then G' is bipartite as well. On the other hand, the transformation does not preserve planarity in general.

Gadgets. The reductions in this chapter are of the component design type: we build "gadgets" corresponding to vertices and edges, and in the reduction a larger graph is constructed from these smaller graphs. In some cases, these gadgets themselves are also built from smaller gadgets. Here we introduce the terminology and the notational conventions that will be used while working with gadgets.

A *gadget* is a graph whose vertices are divided into *external* and *internal* vertices. On the figures, the external vertices of the gadgets are framed (see for example Figure 4.5 or Figure 4.6). If an external vertex has degree one, then the edge incident to it will be called a *pendant edge* (for example, the gadget on Figure 4.5 has 3 pendant edges).

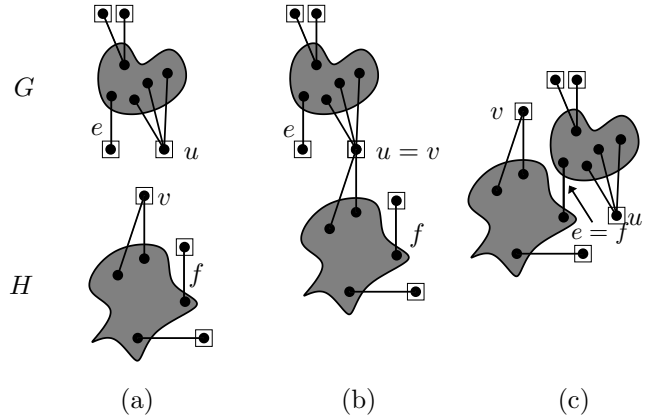


Figure 4.2: The two different ways of combining gadgets. (a) Two gadgets G and H. (b) Joining the vertices u and v. (c) Identifying the two edges e and f.

We will use two operations to create larger graphs from smaller components. If u and v are vertices of G and H, respectively, then the two gadgets can be *joined* by identifying these two vertices (see Figure 4.2b). In particular, if v is the end vertex of a pendant edge h, then this operation will be called *attaching* the pendant edge h of H to vertex u of G. If e is a pendant edge of G, and f is a pendant edge of H, then we can form a larger gadget by *identifying* these two edges (see Figure 4.2c).

4.2 Bipartite graphs

In this section we prove complexity results for the minimum sum edge coloring problem on bipartite graphs. Giaro and Kubale [GK00] have shown that minimum sum edge coloring of bipartite graphs is NP-hard. We strengthen this result in two ways. First, in Section 4.2.1 we show that the problem remains NP-hard for planar bipartite graphs. (As a side note, it is also proved that the problem is NP-hard also for planar 3-regular nonbipartite graphs.) Moreover, in Section 4.2.2 we show that minimum sum edge coloring is not only NP-hard for bipartite graphs, but APX-hard as well. That is, we cannot hope to have a polynomial-time approximation scheme (PTAS) for the problem on bipartite graphs.

4.2.1 Planar graphs

Recall that in the *precoloring extension* problem a graph G is given with some of the vertices having a preassigned color, and it has to be decided whether this precoloring can be extended to a proper k-coloring of the whole graph. One can analogously define the edge precoloring extension problem. We have shown in Theorem 3.4.1 that precoloring extension is NP-complete on the edges of 3-regular planar bipartite graphs. In the following theorem, we reduce the NP-complete edge precoloring extension problem (a problem with local constraints) to deciding whether a graph has a zero error coloring. This proves that the minimum sum edge coloring problem is NP-hard.

Theorem 4.2.1. *The minimum sum edge coloring problem is* NP*-hard for planar bipartite graphs having degrees at most three.*

Proof. Using simple local replacements, we reduce the edge precoloring extension problem to the problem of finding a zero error coloring. Given a 3-regular graph G with some of the edges having a preassigned color, construct a graph G' by replacing the precolored edges with the subgraphs shown on Figure 4.3.

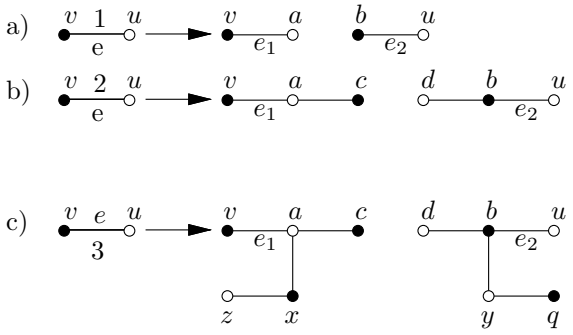


Figure 4.3: Each precolored edge is replaced by the corresponding subgraph on the right.

If we replace the edge $e = uv$ with such a subgraph, then the two new edges incident to v and u will be called e_1 and e_2. If G is planar and bipartite, then clearly G' is planar and bipartite as well.

We show that G' has a zero error coloring if and only if G has a precoloring extension with 3 colors. Assume that ψ is a zero error coloring. We show that for every precolored edge e, the edges e_1 and e_2 receive the color of e. If e is precolored to 1 (case a) of Figure 4.3), then $d(a) = d(b) = 1$, thus e_1, e_2 and receives color 1 in every zero error coloring. If e has color 2, the edges ac and bd must have color 1, thus edges e_1, e_2 have color 2 in every zero error coloring. Finally, if e has color 3, then ac and bd have color 1, edges ax and by have color 2, hence e_1 and e_2 have color 3. Therefore ψ extends the precoloring of G.

The converse is also easy to see: given a precoloring extension of G, for each edge e in G we assign the color of e to edges e_1 and e_2 in G', and extend this coloring the straightforward way. It can be verified that this is a zero error coloring of G', there is no vertex v that is incident to an edge with color greater than $d(v)$ (here we use that G is 3-regular). ■

It is tempting to try to strengthen Theorem 4.2.1 by replacing "degree at most 3" with "3-regular." However, the problem becomes polynomial time solvable for bipartite, regular graphs. In fact, every such graph has a zero error coloring: by the line coloring theorem of Kőnig, every bipartite graph G has a $\Delta(G)$-edge-coloring, which has zero error if G is regular. However, if we add the requirement of 3-regularity, but drop the requirement that the graph is bipartite, then the problem remains NP-hard.

Theorem 4.2.2. *Minimum sum edge coloring is* NP*-hard for planar* 3*-regular graphs.*

Proof. The reduction is from zero error coloring of planar graphs with degree at most 3 (Theorem 4.2.1). We attach certain gadgets to the graph G to make it a 3-regular graph G'. The gadgets are attached in such a way that G has a zero error coloring if and only if G' has a coloring with error K, where K is an integer determined during the reduction.

Figure 4.4 shows three gadgets R_1, R_2, R_3, each gadget has a pendant edge e. We show that gadget R_i has the following property: if its edges are colored in such a way that the error on the internal vertices is as small as possible, then the pendant edge receives color i. The figure shows such a coloring for each gadget, the circled vertices are the vertices where there are errors in the coloring.

Gadget R_1 (see Figure 4.4) has a pendant edge e, 5 internal vertices (denoted by S), and 7 edges connecting the internal vertices. Since every color can be used at most twice on these 7 edges, thus they have sum at least $2 \cdot 1 + 2 \cdot 2 + 2 \cdot 3 + 1 \cdot 4 = 16$ in every coloring. Therefore if a coloring assigns color i to edge e, then the vertices in S have sum at least $32 + i$ and error at least $32 + i - \ell(S) = 2 + i$. Thus the error of S is at least 3 and it can be 3 only if the pendant edge e is colored with color 1.

In the gadget R_3 (second graph on Figure 4.4) two copies of the R_1 gadget are attached to vertex v. The error on the internal vertices is at least 6 in every coloring: there are at least 3 errors in each of S_1 and S_2. However, the error is strictly greater than this: at least one of e_1 and e_2 is colored with a color

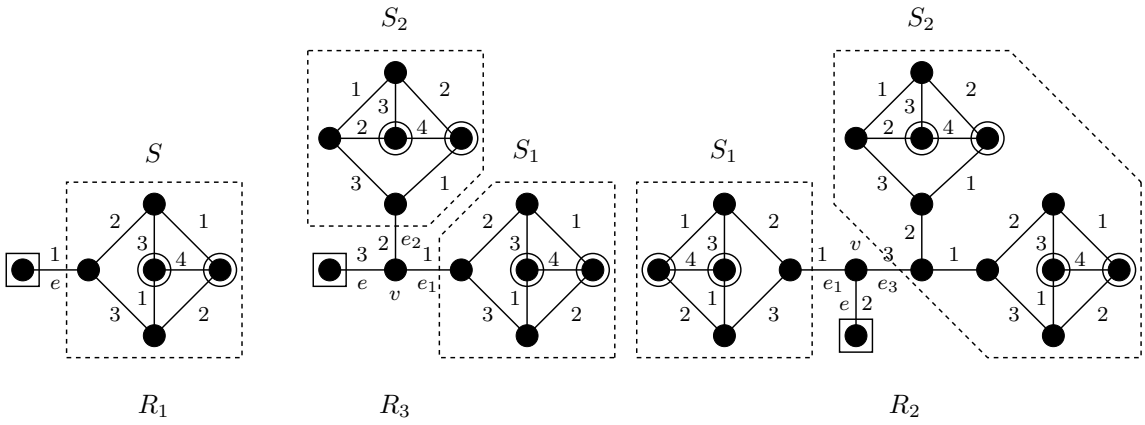


Figure 4.4: The gadgets R_1, R_2, R_3. The coloring given on the figure has as few errors on the internal vertices as possible. The circles show the errors on the internal vertices in this coloring.

greater than 1, hence either S_1 or S_2 has error at least 4. Moreover, if the error of the internal vertices in R_3 is 7, then one of e_1 and e_2 is colored with color 1, the other edge is colored with color 2, therefore edge e has to be colored with color 3.

The gadget R_2 contains a gadget R_1 and R_3 attached to a vertex v, thus it has error at least $3+7 = 10$, since the internal vertices of these gadgets have at least that much error in every coloring. Furthermore, if the error on the internal vertices of R_2 is 10, then this is only possible if the error in S_1 is 3 and the error in S_2 is 7. This implies that the edge e_1 has color 1 and edge e_2 has color 3, therefore edge e has color 2.

Given a planar graph G with degree at most 3, attach to every degree 1 vertex a gadget R_2 and a gadget R_3. Attach to every degree 2 vertex a gadget R_3. Clearly, the resulting graph G' is planar and 3-regular. Let n be the number of R_3 gadgets attached, and let m be the number of R_2 gadgets. We claim that G has a zero error coloring if and only if G' has a coloring with error at most $K = 7n + 10m$.

Assume first that G has zero error. This coloring can be extended in such a way that the error on every attached R_3 (resp. R_2) gadget is 7 (resp. 10), and the edge that connects an R_2 (resp. R_3) gadget to G has color 2 (resp. 3). If v is a vertex of G (not an internal vertex of a gadget), then the three colors 1, 2, and 3 appear at v. Therefore the error of the coloring is the total error of the gadgets, that is, $K = 7n + 10m$.

Assume now that G' has a coloring with error at most K. As we have seen, every gadget R_3 has error at least 7 in every coloring, and every gadget R_2 has error at least 10, therefore if the coloring has error $7n+10m$, then every R_3 gadget has error exactly 7, and every R_2 gadget has error exactly 10. This means that every edge connecting an R_2 (resp. R_3) gadget to G has color 2 (resp. 3). Since G is a subgraph of G', the coloring of G' induces a coloring of G. We show that this coloring is a zero error coloring of G. If v is a degree 1 vertex of G, then in G' two additional edges connect v to an R_2 and an R_3 gadget, these two edges have color 2 and 3. Since the error of v is zero in the coloring, therefore the edge incident to v in G receives color 1. Similarly, if v has degree 2 in G, then an additional edge with color 3 is connected to v in G, and the two edges incident to v in G have the colors 1 and 2, as required. ■

4.2.2 Approximability

In this section we prove that minimum sum edge coloring is APX-hard for bipartite graphs (see Appendix A.2 for background on approximability). In the proof we show hardness for the slightly more general problem of coloring quasigraphs. However, by Proposition 4.1.1 the problem of coloring a quasi-

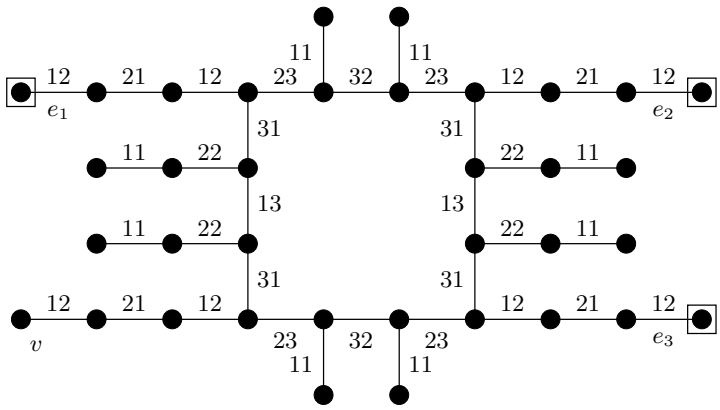


Figure 4.5: The vertex gadget.

graph can be reduced to the coloring of a graph without half-loops, hence the hardness result follows for ordinary simple graphs as well. We remark that the construction of Proposition 4.1.1 can destroy planarity. This is the reason why quasigraphs were not used in Section 4.2.1 to prove the NP-completeness of the planar problem.

First we prove the APX-hardness of minimum sum edge coloring for arbitrary graphs. Later we show how to modify the proof to show that the problem remains APX-hard for bipartite graphs.

Theorem 4.2.3. *Minimum sum edge coloring is* APX*-hard for graphs with maximum degree* 3.

Proof. The theorem is proved by an L-reduction from the minimum vertex cover problem in 3-regular graphs, which is shown to be APX-hard in [AK00]. For every graph $G(V, E)$ with minimum vertex cover of size $\tau(G)$, a graph G'' is constructed, which has edge chromatic sum $C = c_1|V| + c_2|E| + \tau(G)$, where c_1 and c_2 are constants to be determined later. To see that this is an L-reduction, notice that $|E| = \frac{3}{2}|V|$ and $\tau(G) \geq |V|/4$ hold, since G is 3-regular. Therefore $C \leq 4c_1\tau(G) + 6c_2\tau(G) + \tau(G) = c_3\tau(G)$, as required. Furthermore, we show that given an edge coloring of G'' with sum $c_1|V| + c_2|E| + t$, one can find a vertex cover of size t. This proves the correctness of the L-reduction.

The graph G'' is constructed in two steps: first we create a quasigraph G', then apply the transformation of Proposition 4.1.1 to obtain the graph G''. The graph G' contains vertex gadgets and edge gadgets. The vertex gadget shown on Figure 4.5 has 3 pendant edges e_1, e_2, e_3, and satisfies the following two properties:

- If a coloring has zero error on the internal vertices of the variable gadget, then it colors all three pendant edges with color 1.
- There is a coloring that colors all three pendant edges with color 2 and has only 1 error on the internal vertices.

Figure 4.5 shows two possible coloring of the gadget, the two numbers on each edge show the color of the edge in the two colorings. The first coloring is the unique coloring with zero error on the internal vertices. To see this, notice first that an edge incident to a degree 1 internal vertex has to be colored with color 1. Furthermore, if an edge of a degree 2 vertex is colored with color 1, then the other edge has to be colored with color 2. Applying these and similar implications repeatedly, we get the first coloring of Figure 4.5. In particular, edges e_1, e_2, e_3 have color 1, proving the first property. The second coloring has one error (at v), and colors e_1, e_2, e_3 with color 2, proving the second property.

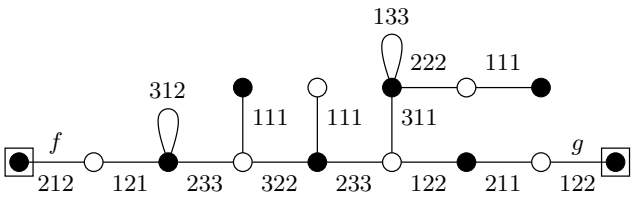


Figure 4.6: The edge gadget.

The edge gadget shown on Figure 4.6 has two pendant edges f and g. If a coloring has zero error on the internal vertices of the gadget, then clearly f and g have color 1 or 2. There are 4 different ways of coloring f and g with color 1 or 2. In 3 out of 4 of these combinations, when at least one of f and g is colored with color 2, the coloring can be extended to the whole gadget with zero error (Figure 4.6 shows these 3 colorings). On the other hand, if both f and g have color 1, there is at least one error on the internal vertices of the gadget. The reader can verify this by following the implications of coloring f and g with color 1, and requiring that every internal vertex has zero error.

The quasigraph $G'(V', E')$ is constructed as follows. A vertex gadget S_v corresponds to every vertex v of G, and an edge gadget S_e corresponds to every edge e of G. Direct the edges of G arbitrarily. If the ith edge incident to $v \in V$ $(i = 1, 2, 3)$ is the head of some edge $e \in E$, then identify edge e_i of S_v with edge f of S_e. If the ith edge incident to $v \in V$ is the tail of some edge $e \in E$, then identify edge e_i of S_v with edge g of S_e. Thus every vertex of G' is an internal vertex of a vertex gadget S_v or an edge gadget S_e. Denote by V_v the internal vertices of gadget S_v and by V_e the internal vertices of S_e, clearly these sets form a partition of V'.

We claim that G' has a coloring with error t if and only if G has a vertex cover with size t. Assume first that $D \subseteq V$ is a vertex cover of G. If $v \in D$, then color gadget S_v such that every pendant edge has color 2 (and there is one error), otherwise color S_v in such a way that every pendant edge has color 1, and there is no error on the internal vertices. Now consider a gadget S_e for some $e \in E$. The two pendant edges f and g are already colored with color 1 or 2. However, at least one of them is colored with 2, since at least one end vertex of e is in D. Therefore, using one of the three colorings shown on Figure 4.6, we can extend the coloring to every edge of S_e with zero error on the internal vertices of the gadget. Therefore errors appear only on the internal vertices of S_v for $v \in D$, and the total error is $|D|$.

On the other hand, consider a coloring of G' with error t. Let $\widehat{V} \subseteq V$ be the set of those $v \in V$ for which V_v is colored with error. Similarly, let $\widehat{E} \subseteq E$ be the set of those $e \in E$ for which V_e is colored with error. Clearly, the coloring has error at least $|\widehat{V}| + |\widehat{E}| \leq t$. Let $\overline{V}$ be a set of $|\widehat{E}|$ vertices in G that cover every edge in $\widehat{E}$. The set of vertices $\widehat{V} \cup \overline{V}$ has size at most $|\widehat{V}| + |\widehat{E}| \leq t$. We show that this set is a vertex cover of G. It is clear that every edge $e \in \widehat{E}$ is covered, since there is a $v \in \overline{V}$ covering e. Now consider an edge $e \notin \widehat{E}$, this means that V_e is colored with zero error, thus, as we have observed, at least one pendant edge of S_e is colored with color 2. If this edge is the pendant edge of the vertex gadget S_v, then there is at least one error in V_v and v is in $\widehat{V}$. However, if the pendant edge of S_e and S_v is identified in the construction, this means that e is incident to v, thus $v \in \widehat{V}$ covers e.

We have proved that the minimum sum edge coloring of G' has $\tau(G)$ errors. Furthermore, $\Sigma'(G') = (c_1/2)|V| + (c_2/2)|E| + \tau(G)/2$ for some constants c_1 and c_2. To see this, notice that the lower bound $\ell(V_v)$ is the same for every $v \in V$ (denote it by c_1), and $\ell(V_e)$ is the same for every $e \in E$ (denote it by c_2). Therefore the sum of the vertices in the optimum coloring is $\ell(V') + \tau(G) = c_1|V| + c_2|E| + \tau(G)$. The edge chromatic sum is the half of this value, $(c_1/2)|V| + (c_2/2)|E| + \tau(G)/2$. Now construct graph G'' from G' as in Proposition 4.1.1. We have that $\Sigma'(G'') = 2\Sigma'(G') = c_1|V| + c_2|E| + \tau(G)$. Furthermore, a coloring of G'' with sum $c_1|V| + c_2|E| + t$ gives a coloring of G' with sum $(c_1/2)|V| + (c_2/2)|E| + t/2$, that is a coloring with error t. It was shown above that given a coloring of G' with error t, one can find a vertex cover of G with size at most t. This completes the proof of the L-reduction. ■

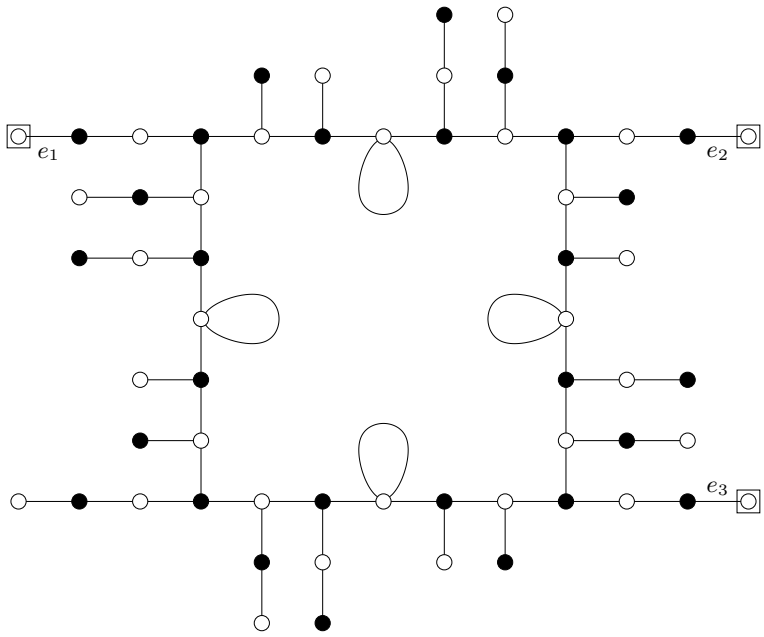


Figure 4.7: The bipartite quasigraph version of the vertex gadget.

Theorem 4.2.3 can be strengthened: the problem remains APX-hard for bipartite graphs. The graph contructed in the proof of Theorem 4.2.3 is not bipartite, since the vertex gadget on Figure 4.5 is not bipartite. However, the vertex gadget can be replaced by the slightly more complex quasigraph shown on Figure 4.7, which is bipartite and has the same properties. That is, if a coloring has zero error on the internal vertices, then the pendant edges have color 1, and there is a coloring that has error 1 on the internal vertices, and assigns color 2 to the pendant edges. The vertex and edge gadgets are bipartite, and they are connected in a way that ensures that the resulting graph G' is bipartite as well.

Theorem 4.2.4. *Minimum sum edge coloring remains* APX*-hard for bipartite graphs with maximum degree* 3*.* ■

4.3 Partial 2-trees

In this section we show that minimum sum edge coloring is NP-hard for partial 2-trees (see Appendix A.1 for definitions). This result is somewhat surprising: the problem is polynomial time solvable for trees [GK00, Sal03, ZN04], and one would expect that its dynamic programming approach generalizes to partial k-trees. However, it turns out that the minimum sum edge coloring is more difficult for partial 2-trees than for trees. This is similar to the case of the edge disjoint paths problem, which is trivial for trees and NP-complete for partial 2-trees and for series-parallel graphs [NVZ01].

Before presenting the proof of NP-completeness, we introduce some gadgets used in the reduction. These gadgets are trees with a single pendant edge, and have the following general property: if a coloring is "cheap," meaning that it has as small error on the internal vertices as possible, then the color of the pendant edge has to be one of the special allowed colors of the gadget. For the gadget F_n, this means that in every such cheap coloring, the pendant edge has color n. In the gadget L_n, the color of the pendant

edge has to be either $n-1$ or $n+1$ in such a coloring. In the gadget A_n, the color of the pendant edge has to be an odd number not greater than n.

The reduction is from 3SAT, therefore we need satisfaction-testing gadgets and variable-setting gadgets. The satisfaction-testing gadget has the property that in every cheap coloring the pendant edge has one of the three preassigned colors. The variable-setting gadget W_n is different from the other gadgets. First, it is not a tree, but a partial 2-tree. Moreover, there are *two* edges incident to its external vertex, instead of one. The crucial property of this gadget is that in every cheap coloring, these two edges either use the colors $n+1$, $n+3$, or they use the colors $n+5$, $n+7$.

In the following lemmas, we formally define the properties of the gadgets, describe how they are constructed, and prove the required properties.

Lemma 4.3.1. *For every $n \geq 2$, there is a tree F_n and an integer f_n, such that*

1. *F_n has one pendant edge e,*
2. *the internal vertices of F_n have error at least f_n in every coloring,*
3. *if a coloring has error f_n on the internal vertices of F_n, then this coloring assigns color n to the pendant edge e, and*
4. *F_n can be constructed in time polynomial in n.*

Proof. The tree F_n is a star with a central vertex v, and n leaves v_1, v_2, …, v_n. The pendant edge e is the edge $v_n v$, thus the internal vertices are v, v_1, v_2, …, v_{n-1}. Let $f_n = (n-1)(n-2)/2$. The $n-1$ edges $v_1 v$, …, $v_{n-1} v$ have different colors, hence the sum of the vertices v_1, …, v_{n-1} is at least $\sum_{i=1}^{n-1} i = n(n-1)/2$. Therefore the error on these vertices is at least $n(n-1)/2 - (n-1) = f_n$. There is equality if and only if the sum of these vertices is exactly $n(n-1)/2$ and there is no error on v. This implies that edge $v_n v$ has color n, as required. ■

Lemma 4.3.2. *For every even $n \geq 1$, there is a tree L_n and an integer k_n, such that*

1. *L_n has one pendant edge e,*
2. *the internal vertices of L_n have error at least k_n in every coloring,*
3. *if a coloring has error k_n on the internal vertices of L_n, then this coloring assigns either color $n-1$ or $n+1$ to the pendant edge e,*
4. *there are colorings ψ_{n-1} and ψ_{n+1} of L_n with $\psi_{n-1}(e) = n-1$, $\psi_{n+1}(e) = n+1$, such that they have error k_n on the internal vertices, and*
5. *L_n can be constructed in time polynomial in n.*

Proof. The tree L_n is constructed as follows (see Figure 4.8). The pendant edge e connects external vertex u and internal vertex v. The $n-2$ vertices v_1, v_2, …, v_{n-2} are connected to v, these vertices form the set V. There are two additional neighbors of v: vertices a and b. Besides v, vertex a has $n-1$ neighbors a_1, a_2, …, a_{n-1}, let A be the set containing these $n-1$ vertices. Similarly, vertex b has $n-1$ additional neighbors $B = \{b_1, b_2, \ldots, b_{n-1}\}$.

Since the edges $v_1 v$, $v_2 v$, …, $v_{n-2} v$ have different colors in every coloring of L_n, hence the sum of V is at least $\sum_{i=1}^{n-2} i = (n-2)(n-1)/2$ in every coloring. Therefore there is error at least $(n-2)(n-1)/2 - \ell(V) = (n-2)(n-1)/2 - (n-2) = (n-2)(n-3)/2$ on V in every coloring, this minimum is reached if and only if the edges $v_1 v$, …, v_{n-2} have the colors 1, …, $n-2$ (in some order). Similarly, there is error at least $(n-1)n/2 - (n-1) = (n-1)(n-2)/2$ on both A and B. Therefore there is error at least $(n-2)(n-3)/2 + 2 \cdot (n-1)(n-2)/2$ on the internal vertices in every coloring. However, the error is

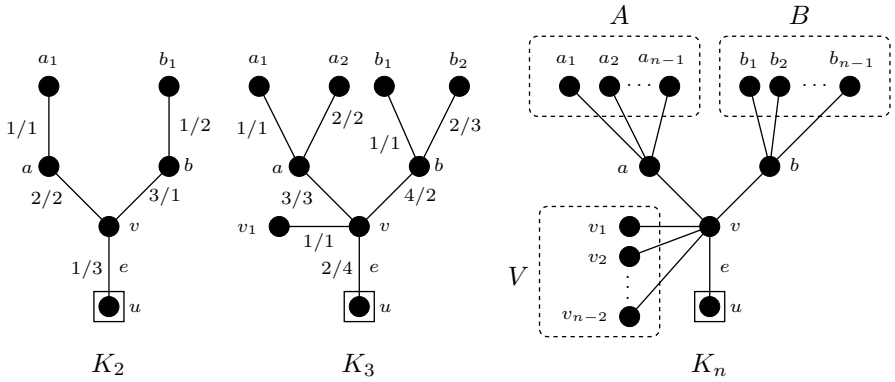


Figure 4.8: The gadget L_n.

always strictly greater than this. If the error is exactly $(n-1)(n-2)/2$ on both A and B, and there is zero error on a and b, then edges va and vb both have to receive color n. Thus we can conclude that there is error at least $k_n = (n-2)(n-3)/2 + 2 \cdot (n-1)(n-2)/2 + 1$ in every coloring.

The coloring ψ_{n-1} is defined as

- $\psi_{n-1}(e) = n-1$,
- $\psi_{n-1}(va) = n$,
- $\psi_{n-1}(vb) = n+1$,
- $\psi_{n-1}(v_i v) = i$ for $1 \leq i \leq n-2$,
- $\psi_{n-1}(a_i a) = i$ for $1 \leq i \leq n-1$, and
- $\psi_{n-1}(b_i b) = i$ for $1 \leq i \leq n-1$.

It can be verified that $\epsilon_{\psi_{n-1}}(V) = (n-2)(n-3)/2$, $\epsilon_{\psi_{n-1}}(A) = \epsilon_{\psi_{n-1}}(B) = (n-1)(n-2)/2$, $\epsilon_{\psi_{n-1}}(a) = \epsilon_{\psi_{n-1}}(v) = 0$, and $\epsilon_{\psi_{n-1}}(b) = 1$, therefore the error of ψ_{n-1} on the internal vertices of L_n is k_n. Coloring ψ_{n+1} is the same as coloring ψ_{n-1}, except that

- $\psi_{n+1}(e) = n+1$,
- $\psi_{n+1}(vb) = n-1$, and
- $\psi_{n+1}(b_{n-1} b) = n$.

This change decreases the error on b to zero, and increases the error on b_{n-1} to 1. Therefore ψ_{n+1} also has error k_n on the internal vertices, and this proves Property 4.

To show that Property 3 holds, assume that coloring ψ has error k_n on the internal vertices of L_n. As we have observed, $e_\psi(A \cup \{a\}) = (n-1)(n-2)/2$ implies $\psi(va) = n$. Similarly, $e_\psi(B \cup \{b\}) = (n-1)(n-2)/2$ implies $\psi(vb) = n$, therefore at least one of $A \cup \{a\}$ and $B \cup \{b\}$ have error strictly greater than $(n-1)(n-2)/2$. Assume, without loss of generality, that $e_\psi(A \cup \{a\}) > (n-1)(n-2)/2$, then the error of ψ can be k_n only if $e_\psi(B \cup \{b\}) = (n-1)(n-2)/2$, $e_\psi(V) = (n-2)(n-3)/2$, and there is no error on v. Therefore color n is used by edge vb, and the colors 1, 2, …, $n-2$ are used by the edges v_1, v_2, …, v_{n-2} (not necessarily in this order). Since there is zero error at v, and v has degree $n+1$, thus edge e has a color not greater than $n+1$. This can be only $n-1$ or $n+1$, since the other colors are already used by edges incident to v. ■

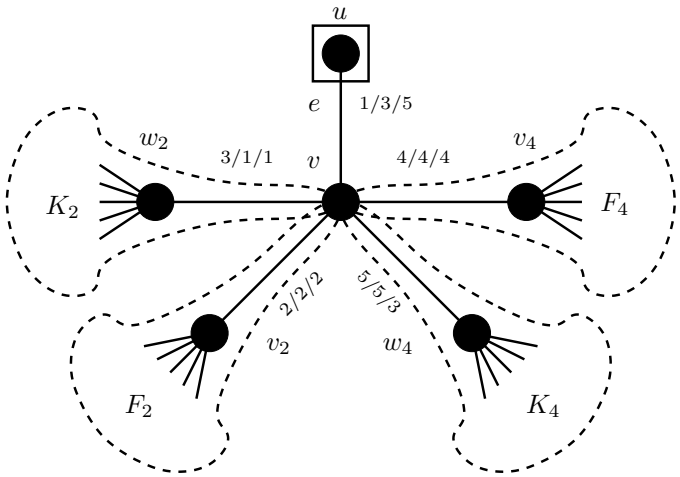


Figure 4.9: The gadget A_5.

Lemma 4.3.3. *For every odd $n \geq 1$, there is a tree A_n and an integer a_n such that*

1. *A_n has one pendant edge e,*
2. *the internal vertices of A_n have error at least a_n in every coloring,*
3. *if a coloring ψ has error a_n on the internal vertices of A_n, then $\psi(e)$ is odd and $\psi(e) \leq n$,*
4. *for every odd c not greater than n, there is a coloring ψ_c of A_n such that $\psi_c(e) = c$ and it has error a_n on the internal vertices,*
5. *A_n can be constructed in time polynomial in n.*

Proof. The pendant edge e of L_n connects external vertex u and internal vertex v. Attach the pendant edges of the $(n-1)/2$ trees F_2, F_4, …, F_{n-1} (Lemma 4.3.1) to vertex v, let the pendant edges of these trees be v_2v, v_4v, …, $v_{n-1}v$, respectively (see Figure 4.9). Similarly, attach the pendant edges of the $(n-1)/2$ trees L_2, L_4, …, L_{n-1} (Lemma 4.3.2) to v, let the pendant edges of these trees be w_2v, w_4v, …, $w_{n-1}v$, respectively. Therefore the degree of v in A_n is n.

Let $a_n = (f_2 + f_4 + \cdots + f_{n-1}) + (k_2 + k_4 + \cdots + k_{n-1})$. Since A_n contains a copy of the trees F_2, F_4, …, F_{n-1}, and a copy of the trees L_2, L_4, …, L_{n-1}, thus it is clear that every coloring of A_n has at least a_n errors on the internal vertices. Moreover, if a coloring ψ has error a_n on the internal vertices, then $\psi(v_iv) = i$ for $i = 2, 4, \ldots, n-1$, and the error of v is zero. This implies that $\psi(e) \leq n$ and not even, as required.

The coloring ψ_c required by Property 4 is the following. For every $i = 2, 4, \ldots, n-1$, coloring ψ_c colors the edges of the tree F_i in such a way that the pendant edge v_iv receives color i, and there is error f_i on the internal vertices of F_i, by Lemma 4.3.1, there is such a coloring. For every $i = 2, 4, \ldots, c-1$, the tree L_i is colored such that the pendant edge w_iv has color $i-1$, and there is error k_i on the internal vertices of L_i. Similarly, for $i = c+1, \ldots, n-1$, the tree L_i is colored such that the pendant edge w_iv has color $i+1$, and there is error k_i on the internal vertices of L_i. Coloring ψ_c assigns color c to edge e, thus every color 1, 2, …, n appears on exactly one edge incident to v. Therefore the error of v is zero, and there is error a_n on the internal vertices of A_n. ■

Lemma 4.3.4 (Satisfaction-testing gadget). *For odd integers $x_1 < x_2 < x_3$, there is a tree S_{x_1,x_2,x_3} and an integer s_{x_1,x_2,x_3} such that*

1. *S_{x_1,x_2,x_3} has one pendant edge e,*
2. *the internal vertices of S_{x_1,x_2,x_3} have error at least s_{x_1,x_2,x_3} in every coloring,*
3. *if a coloring ψ has error s_{x_1,x_2,x_3} on the internal vertices of S_{x_1,x_2,x_3}, then $\psi(e) \in \{x_1, x_2, x_3\}$*
4. *for $i = 1, 2, 3$, there is a coloring ψ_i of S_{x_1,x_2,x_3} such that $\psi_i(e) = x_i$ and it has error s_{x_1,x_2,x_3} on the internal vertices,*
5. *S_{x_1,x_2,x_3} can be constructed in time polynomial in x_3.*

Proof. The pendant edge e of S_{x_1,x_2,x_3} connects external vertex u and internal vertex v. Attach to vertex v the pendant edges of

- $x_1 - 1$ trees F_1, F_2, …, F_{x_1-1} (Lemma 4.3.1),
- $x_2 - x_1 - 1$ trees F_{x_1+1}, …, F_{x_2-1},
- $x_3 - x_2 - 1$ trees F_{x_2+1}, …, F_{x_3-1}, and
- 2 copies of the tree A_{x_3} (Lemma 4.3.3).

Vertex v has degree x_3 in S_{x_1,x_2,x_3}. Set $s_{x_1,x_2,x_3} = f_1 + f_2 + \cdots + f_{x_1-1} + f_{x_1+1} + \cdots + f_{x_2-1} + f_{x_2+1} + \cdots + f_{x_3-1} + 2a_{x_3}$, because of the way S_{x_1,x_2,x_3} is constructed, it is clear that every coloring of S_{x_1,x_2,x_3} has error at least s_{x_1,x_2,x_3} on the internal vertices. If ψ has error exactly s_{x_1,x_2,x_3} on the internal vertices, then v has zero error and $\psi(e) \leq d(v) = x_3$. Furthermore, it also follows that the colors 1, …, $x_1 - 1$, $x_1 + 1$, …, $x_2 - 1$, $x_2 + 1$, …, $x_3 - 1$ are used at v by the pendant edges of the attached trees F_1, …, F_{x_1-1}, F_{x_1+1}, …, F_{x_2-1}, F_{x_2+1}, …, F_{x_3-1}, respectively. Therefore edge e has one of the remaining colors x_1, x_2, x_3, proving Property 3.

The colorings ψ_1, ψ_2, ψ_3 required by Property 4 color the $(x_1 - 1) + (x_2 - x_1 - 1) + (x_3 - x_2 - 1)$ trees of type F_i in the same way: all three colorings color these trees such that there is error $f_1 + f_2 + \cdots + f_{x_1-1} + f_{x_1+1} + \cdots + f_{x_2-1} + f_{x_2+1} + \cdots + f_{x_3-1}$ on the internal vertices of the trees, and their pendant edges use the colors 1, …, $x_1 - 1$, $x_1 + 1$, …, $x_2 - 1$, $x_2 + 1$, …, $x_3 - 1$ at v. Coloring ψ_i assigns color x_i to the pendant edge e, hence two colors not greater than x_3 remains unused at v: only the colors $\{x_1, x_2, x_3\} \setminus x_i$ are not yet assigned. These two colors are odd and not greater than x_3, thus by Property 4 of Lemma 4.3.3, we can color the two copies of A_{x_3} attached to v such that their pendant edges have these two colors, introducing additional error $2a_{x_3}$. Since there is zero error on v, the error of this coloring is s_{x_1,x_2,x_3} on the internal vertices of S_{x_1,x_2,x_3}, as required by Property 4. ■

Lemma 4.3.5 (Variable-setting gadget). *For every $n \geq 0$, there is a partial 2-tree W_n and an integer w_n such that*

1. *W_n has an external vertex v, and two edges e_1 and e_2 incident to v,*
2. *every coloring of W_n has error at least w_n on the internal vertices of W_n,*
3. *if a coloring ψ of W_n has error w_n on the internal vertices, then either*
   - *$\psi(e_1) = n + 1$, $\psi(e_2) = n + 3$ or*
   - *$\psi(e_1) = n + 5$, $\psi(e_2) = n + 7$ holds,*

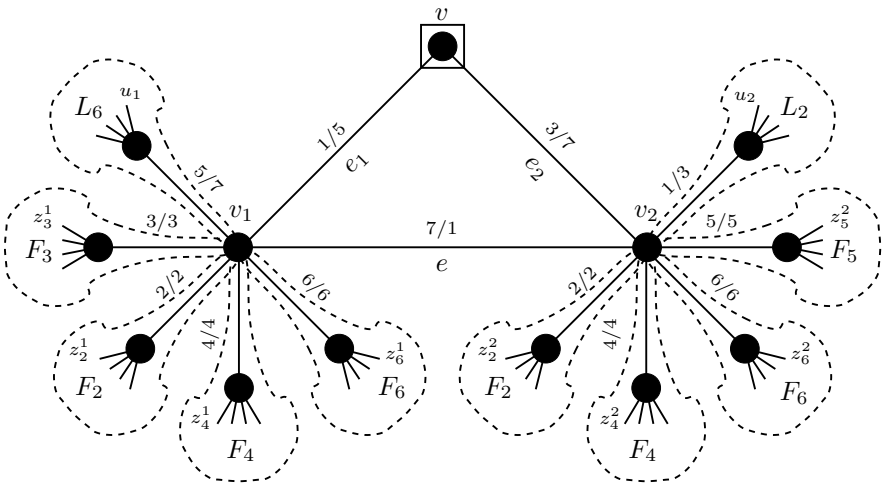


Figure 4.10: The variable-setting gadget W_0.

4. *there are colorings ψ_1 and ψ_2 of W_n with error w_n on the internal vertices such that*
   - $\psi_1(e_1) = n + 1$, $\psi_1(e_2) = n + 3$,
   - $\psi_2(e_1) = n + 5$, $\psi_2(e_2) = n + 7$, *and*
5. *W_n can be constructed in time polynomial in n.*

Proof. The graph W_n is constructed as follows (see Figure 4.10 for the case $n = 0$). The external vertex v is connected to vertex v_1 by edge e_1, and to v_2 by e_2. Vertices v_1 and v_2 are connected by an edge e. We attach several trees to vertices v_1 and v_2:

- Attach n trees F_1, F_2, ..., F_n to v_1, let the pendant edges of these trees be $z_1^1v_1$, $z_2^1v_1$, ..., $z_n^1v_1$, respectively.
- Similarly, attach a copy of these n trees to v_2, let the pendant edges be $z_1^2v_2$, $z_2^2v_2$, ..., $z_n^2v_2$.
- Attach to v_1 the trees F_{n+2}, F_{n+3}, F_{n+4}, F_{n+6} with pendant edges $z_{n+2}^1v_1$, $z_{n+3}^1v_1$, $z_{n+4}^1v_1$, $z_{n+6}^1v_1$, respectively.
- Attach to v_1 a tree L_{n+6} with pendant edge u_1v_1.
- Attach to v_2 the trees F_{n+2}, F_{n+4}, F_{n+5}, F_{n+6} with pendant edges $z_{n+2}^2v_2$, $z_{n+4}^2v_2$, $z_{n+5}^2v_2$, $z_{n+6}^2v_2$, respectively.
- Attach to v_2 a tree L_{n+2} with pendant edge u_2v_2.

Notice that both v_1 and v_2 have degree $n + 7$. The graph W_n is a partial 2-tree: it is chordal, and it has clique number 3.

Set $w_n = 2(f_1+f_2+\cdots+f_n)+(f_{n+2}+f_{n+3}+f_{n+4}+f_{n+6}+k_{n+6})+(f_{n+2}+f_{n+4}+f_{n+5}+f_{n+6}+k_{n+2})$. It is clear that every coloring of W_n has error at least w_n on the internal vertices: the combined error in the attached trees is always at least w_n. Moreover, if the error of coloring ψ is w_n on the internal vertices, then there has to be zero error on v_1 and v_2. Furthermore, from Lemma 4.3.1 and Lemma 4.3.2, in this case we also have that

- $\psi(z_i^1 v_1) = \psi(z_i^2 v_2) = i$ for $i = 1, 2, \ldots, n$,
- $\psi(z_i^1 v_1) = \psi(z_i^2 v_2) = i$ for $i = n+2, n+4, n+6$,
- $\psi(z_{n+3}^1 v_1) = n+3$,
- $\psi(z_{n+5}^2 v_2) = n+5$,
- $\psi(u_1 v_1)$ is either $n+5$ or $n+7$, and
- $\psi(u_2 v_2)$ is either $n+1$ or $n+3$.

Since the degree of v_1 is $n+7$, and there is zero error on v_1, it follows that $\psi(e) \leq n+7$. Moreover, $\psi(e)$ is either $n+1$ or $n+7$: as shown above, every other color not greater than $n+7$ is already used on at least one of v_1 and v_2. Assume first that $\psi(e) = n+1$. In this case $u_2 v_2$ cannot have color $n+1$, therefore $\psi(u_2 v_2) = n+3$ follows. Now the only unused color not greater than $n+7$ at v_2 is $n+7$, hence $\psi(e_2) = n+7$. There remains two unused colors at v_1: color $n+5$ and color $n+7$. However, edge e_1 cannot have color $n+7$, since edge e_2 already has this color. Thus we have $\psi(e_1) = n+5$ and $\psi(e_2) = n+7$, as required by Property 4. Similarly, assume that $\psi(e) = n+7$, it follows that $\psi(u_1 v_1) = n+5$. The only unused color not greater than $n+7$ at v_1 is $n+1$, hence edge e_1 has to receive this color. Colors $n+3$ and $n+1$ are the only remaining colors at v_2, therefore e_2 has color $n+3$, since $n+1$ is already used by e_1. Thus we have $\psi(e_1) = n+1$ and $\psi(e_2) = n+3$, as required.

The two colorings ψ_1 and ψ_2 required by Property 4 are given as follows (see Figure 4.10 for the case $n = 0$). Consider the (partial) coloring ψ with

- $\psi(z_i^1 v_1) = \psi(z_i^2 v_2) = i$ for $i = 1, 2, \ldots, n$,
- $\psi(z_i^1 v_1) = \psi(z_i^2 v_2) = i$ for $i = n+2, n+4, n+6$,
- $\psi(z_{n+3}^1 v_1) = n+3$ and
- $\psi(z_{n+5}^2 v_2) = n+5$.

Both ψ_1 and ψ_2 assign the same colors as ψ, but we also have

- $\psi_1(e_1) = n+1$, $\psi_1(e_2) = n+3$, $\psi_1(e) = n+7$,
- $\psi_1(u_1 v_1) = n+5$,
- $\psi_1(u_2 v_2) = n+1$.
- $\psi_2(e_1) = n+5$, $\psi_2(e_2) = n+7$, $\psi_2(e) = n+1$,
- $\psi_2(u_1 v_1) = n+7$,
- $\psi_2(u_2 v_2) = n+3$.

In these colorings there are zero error on vertices v_1 and v_2. Furthermore, these colorings can be extended to the attached trees with error w_n: the colors assigned to the pendant edges of the attached trees are compatible with the "best" coloring of the attached trees (see Property 4 of Lemma 4.3.2 and Property 3 of Lemma 4.3.1). This proves Property 4 of the lemma. ■

Theorem 4.3.6. *Minimum sum edge coloring is* NP*-hard for partial* 2*-trees.*

Proof. The proof is by reduction from 3SAT: given a 3CNF formula φ, we construct a partial 2-tree G and determine an integer K such that $\Sigma'(G) \leq K$ if and only if φ is satisfiable. By Prop. 2.1.3, we can assume that every variable occurs exactly twice positively and exactly twice negated in ϕ, and every clause contains exactly three literals.

Let n be the number of variables in φ, the variables will be called x_0, x_1, ..., x_{n-1}. The number of clauses is therefore $m = 4n/3$. For every literal of φ, there is a corresponding color, as follows:

- color $8i+1$ corresponds to the first positive occurence of x_i,
- color $8i+3$ corresponds to the second positive occurence of x_i,
- color $8i+5$ corresponds to the first negated occurence of x_i, and
- color $8i+7$ corresponds to the second negated occurence of x_i.

Notice that these numbers are odd, and every odd number not greater than $8n$ corresponds to a literal.

Take a vertex v, we will attach several gadgets to v to obtain the graph G. Attach $4n$ trees F_2, F_4, ..., F_{8n} to v, let the pendant edges of the attached trees be u_2v, u_4v, ..., $u_{8n}v$, respectively. Attach n variable-setting gadgets W_0, W_8, W_{16}, ..., $W_{8(n-1)}$ to v, let the two edges of W_{8i} incident to v be called $w_{i,1}v$ and $w_{i,2}v$. For every clause C_j of φ, we attach a satisfaction-testing gadget to v in the following way: if colors $c_{j,1} < c_{j,2} < c_{j,3}$ correspond to the three literals in clause C_j, then attach a tree S_{c_1,c_2,c_3} to v, and let s_jv be its pendant edge. Finally, attach $m/2$ copies of the tree A_{8n-1} to v, let the pendant edges of these trees be t_1v, t_2v, ..., $t_{\frac{m}{2}}v$. This completes the description of the graph G. Since every gadget is a partial 2-tree (or even a tree), the graph G is a partial 2-tree as well: joining graphs at a single vertex does not increase the treewidth of the graphs.

Let $K^{(1)} = f_2 + f_4 + \cdots + f_{8n}$. In every coloring of G the error is at least $K^{(1)}$ on the internal vertices of the $4n$ attached F_i trees. Let $K^{(2)} = w_0 + w_8 + \cdots + w_{8(n-1)}$. In every coloring the error is at least $K^{(2)}$ on the internal vertices of the n variable-setting gadgets. Let $K^{(3)} = m/2 \cdot a_{8n-1}$. In every coloring the error is at least $K^{(3)}$ on the internal vertices of the $m/2$ copies of A_{8n-1}. Let $K^{(4)} = \sum_{j=1}^{m} s_{c_{j,1},c_{j,2},c_{j,3}}$ where $c_{j,k}$ is the color corresponding to the kth literal in clause C_j. In every coloring of G, the error on the internal vertices of the m satisfaction-testing gadget is at least $K^{(4)}$. Finally, set $K = K^{(1)} + K^{(2)} + K^{(3)} + K^{(4)}$. It is clear that every coloring of G has error at least K. We claim that G has a coloring with error exactly K if and only if φ is satisfiable.

Assume first that coloring ψ has error K. This is possible only if ψ has zero error on v, and there is exactly K error on the internal vertices of the attached gadgets. By Lemma 4.3.1, Lemma 4.3.3, Lemma 4.3.4, and Lemma 4.3.5 this implies that

- $\psi(u_iv) = i$ for $i = 2, 4, \ldots, 8n$,
- for every $i = 0, 1, \ldots, n-1$, either
  - $\psi(w_{i,1}v) = 8i+1$ and $\psi(w_{i,2}v) = 8i+3$, or
  - $\psi(w_{i,1}v) = 8i+5$ and $\psi(w_{i,2}v) = 8i+7$,
- $\psi(s_jv) \in \{c_{j,1}, c_{j,2}, c_{j,3}\}$ for every $j = 1$, ..., m, and
- $\psi(t_iv) \leq 8n-1$ and odd for every $i = 1$, 2, ..., $m/2$.

Consider the following variable assignment: set variable x_i to true if $\psi(w_{i,1}v) = 8i+5$, $\psi(w_{i,2}v) = 8i+7$, and set x_i to false if $\psi(w_{i,1}v) = 8i+1$ and $\psi(w_{i,2}v) = 8i+3$. We show that this is a satisfying assignment of φ, every clause C_j is satisfied. Assume that $\psi(s_jv) = c_{j,k}$ for some $k = 1, 2, 3$, and let the kth literal in clause C_j be an occurence of the variable x_i. In this case, the kth literal of clause C_j is true in the

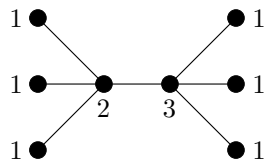


Figure 4.11: A tree with strength 3. The figure shows a minimum sum coloring with sum 11, while every 2-coloring has sum 12.

constructed variable assignment: otherwise color $c_{j,w}$ would appear also on edge $w_{i,1}v$ or $w_{i,2}v$. Therefore every clause contains at least one true literal, and the formula is satisfied by the variable assignment.

Now assume that φ has a satisfying variable assignment. Consider the following (partial) coloring ψ:

- $\psi(u_i v) = i$ for $i = 2, 4, \ldots, 8n$,
- for every $i = 0, 1, \ldots, n-1$,
  - if variable x_i is true, then $\psi(w_{i,1}v) = 8i + 5$ and $\psi(w_{i,2}v) = 8i + 7$,
  - if variable x_i false, then $\psi(w_{i,1}v) = 8i + 1$ and $\psi(w_{i,2}v) = 8i + 3$,

It is clear from the construction that for every $j = 1, 2, \ldots, m$, one of the colors $c_{j,1}$, $c_{j,2}$, $c_{j,3}$ is not already assigned: otherwise this would imply that clause C_j contains only false literals in the satisfying variable assignment, a contradiction. Therefore we can set $\psi(s_j v)$ to one of these three colors. So far coloring ψ assigns $4n$ even and $2n + m$ odd colors to the edges incident to v, therefore there remains exactly $m/2$ odd colors not greater than $8n$. Assign these colors to the edges $t_1 v$, $t_2 v$, $\ldots$, $t_{\frac{m}{2}} v$ in some order. Now every color not greater than $8n$ is used exactly once at v, hence there is zero error on vertex v in ψ. It is straightforward to verify that this coloring can be extended to the whole graph G such that the resulting coloring has error exactly K: in every gadget, the edges incident to v are colored in such a way that makes this extension possible. ■

4.4 Chromatic strength

In [KS89] it is noted that the number of colors required by a minimum sum coloring can be much greater than the chromatic number of the graph. In particular, for every $k \geq 2$, they show a tree where every minimum sum coloring uses at least k different colors (see Figure 4.11 for an example of the case $k = 3$). Let $s(G)$ be the *chromatic strength* of G, which is the smallest number of colors required in a minimum sum coloring of G. The *chromatic edge strength* $s'(G)$ is defined analogously. Clearly, $s(G) \geq \chi(G)$, but as the trees in [KS89] show, $s(G) - \chi(G)$ can be arbitrarily large. On the other hand, [MMS97] and independently [HMT00] prove an analog of Vizing's Theorem showing that $s'(G) \leq \Delta(G) + 1$ in every simple graph G. Hence we have

$$\Delta(G) \leq \chi'(G) \leq s'(G) \leq \Delta(G) + 1$$

if G is a simple graph. Harary and Plantholt conjectured (see [Wes95]) that the second inequality is in fact an equality, hence if a simple graph is k-edge-colorable, then it has a minimum sum edge coloring with k colors. However, this conjecture turned out to be false: for every odd integer $k \geq 5$, a graph with chromatic index k and edge strength $k+1$ was given in [MMS97]. Moreover, [HMT00] gives such a graph for $k = 4$. Thus we can conclude that the chromatic index and the chromatic edge strength are not always the same.

Here we study the computational complexity of determining the chromatic strength and chromatic edge strength of a simple graph. The complexity of the vertex strength is investigated in [Sal03]:

Theorem 4.4.1 ([Sal03]). *For every $k \geq 3$, it is* NP*-hard to decide whether $s(G) \leq k$ holds for a given graph G.*

Notice that it is not clear whether this problem belongs to NP. A minimum sum k-coloring is not a good certificate for $s(G) \leq k$, since we cannot verify that it is indeed a minimum sum coloring. On the other hand, the problem does not seem to belong to coNP either: a minimum sum coloring with more than k colors does not certify that $s(G) > k$, since it does not prove that this sum cannot be achieved using only k colors. The main result of this chapter is that we determine the exact complexity of the chromatic strength problem by showing that for every $k \geq 3$, it is Θ_2^p-complete to decide whether $s(G) \leq k$ holds. The class Θ_2^p contains those problems that can be solved in polynomial time with a logarithmic number of NP oracle calls (see Appendix A.3 and Section 4.4.3 for definitions). It is interesting to see a natural coloring problem that is complete for this less-known complexity class. In [Sal03] the complexity of the case $k = 2$ was left as an open question. We answer this question by showing that deciding $s(G) \leq 2$ is coNP-complete.

We obtain our Θ_2^p-completeness result for the chromatic strength by proving the stronger statement that even the more restricted chromatic edge strength problem is Θ_2^p-complete. The complexity of edge strength is also treated in [Sal03]. By observing that $s'(G) = \chi'(G)$ for every regular simple graph, they conclude that for regular graphs "Is $s'(G) \leq k$?" has the same complexity as "Is $\chi'(G) \leq k$?", and the latter problem is known to be NP-complete for every $k \geq 3$ [Hol81, LG83]. However, if we want to prove that edge strength is Θ_2^p-complete (that is, harder than the chromatic index problem), then necessarily we have to consider graphs where the edge strength and the chromatic index are not the same. Therefore we need substantially different (and more complicated) arguments than in [Sal03].

We prove the Θ_2^p-completeness of chromatic edge strength the following way. First we show that for every $k \geq 3$, there is a simple graph G_k with $\Delta(G_k) = \chi'(G_k) = k$ and $s'(G_k) = k + 1$. That is, we give counterexamples to the conjecture of Harary and Plantholt in all the remaining cases. Next, in Section 4.4.3 we introduce some new Θ_2^p-complete problems, which might be of independent interest as well. In particular, we show that it is Θ_2^p-complete to decide whether every minimum vertex cover of a given graph includes the distinguished vertex $\hat{v}$. Finally, we show that if the chromatic index and the chromatic edge strength are not always the same in graphs with maximum degree k, then it is Θ_2^p-complete to decide whether the edge strength is k in such a graph. Together with the existence of the counterexample graphs, this gives the required result.

In Section 4.4.1, we show that it is coNP-complete to decide whether $s(G) \leq 2$. In the rest of the chapter, we consider only the edge coloring version of the problem. The counterexamples to the conjecture of Harary and Plantholt are given in Section 4.4.2. In Section 4.4.3 we summarize the results on the complexity class Θ_2^p, and introduce the new Θ_2^p-complete problems. The reduction for the main hardness result is presented in Section 4.4.4. The construction of the key gadget of the reduction is given in Section 4.4.5.

4.4.1 Vertex strength of bipartite graphs

In this section we prove that for $k = 2$, it is coNP-complete to decide whether $s(G) \leq k$ holds. Notice that, unlike in the case $k \geq 3$, now it is easy to see that the problem is in coNP. First, the question makes sense only if the graph is bipartite, otherwise trivially $s(G) \geq 3$. In a bipartite graph the sum of the best 2-coloring is easy to determine: each connected component of the graph has exactly two 2-colorings, and taking the better coloring of each component gives the best 2-coloring of the graph. Therefore a minimum sum coloring with more than 2 colors certifies that $s(G) \leq 2$ *does not* hold: one can determine the sum of the best 2-coloring, and check that it is indeed larger than the sum of the given coloring. Thus the problem is in coNP.

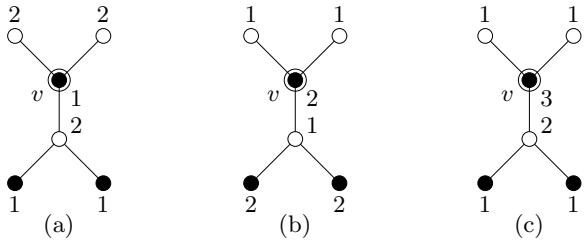


Figure 4.12: The new vertices attached to vertex v of H, and three minimum sum colorings that assign to vertex v (a) color 1, (b) color 2, (c) color 3.

The proof of coNP-hardness is by reduction from the precoloring extension problem. Precoloring extension (PrExt) is a generalization of vertex coloring (see Chapter 3): we are given a graph $G(V, E)$ with a subset $W \subseteq V$ of vertices having a preassigned color, the question is whether this precoloring can be extended to a proper k-coloring of the graph. We denote by 1-PrExt the special case where every color is used at most once in the precoloring. 1-PrExt is NP-complete in bipartite graphs [HT93], but polynomial-time solvable in interval graphs [BHT92] and more generally, in chordal graphs (Section 3.1). In our proof, we need the following result:

Theorem 4.4.2 ([BJW94]). 1-PrExt *is* NP*-complete for bipartite graphs, even if the number of colors is* 3*.*

Moreover, it can be assumed that the 3 precolored vertices are in the same bipartition class (see the proof in [BJW94]).

Theorem 4.4.3. *Given a graph G, it is* coNP*-complete to decide if $s(G) \leq 2$ holds.*

Proof. As we have noted above, the problem is in coNP. Hardness is proved by reduction from 1-PrExt for bipartite graphs. Given a bipartite graph $H(A, B; E)$ with three precolored vertices $v_1, v_2, v_3 \in A$, we construct a (bipartite) graph G such that $s(G) \leq 2$ if and only if the precoloring of H *cannot* be extended to the whole graph.

To construct the graph G, we attach 5 new vertices to every non-precolored vertex v of H (see Figure 4.12). Let the set V_v contain vertex v and the 5 vertices attached to it. In every coloring, the sum of the 6 vertices of V_v is at least 9. Moreover, as shown on Figure 4.12a-c, this minimum sum 9 can be achieved even if vertex v has color 1, 2, or 3.

The three precolored vertices v_1, v_2, v_3 are connected by the graph shown on Figure 4.13. Denote by V^* the set of these 15 vertices. The vertices in the set V^* have sum at least 20 in every coloring. Furthermore, it can be verified that the coloring shown on the figure is the unique minimum sum coloring of V^*. This completes the description of the graph G. It can be easily verified that if H is bipartite, then G is bipartite as well.

If the graph H has $n + 3$ vertices, then G has $6n + 15$ vertices, and the sum of every coloring is at least $9n + 20$. Moreover, G has a 2-coloring with sum $9n + 21$: color v_1, v_2, v_3, and their bipartition class with color 2, the other class receives color 1. Every set V_v has sum 9 in this coloring, while V^* has sum 21. This means that $s(G) > 2$ if and only if there is a coloring of G with sum $9n + 20$: otherwise the sum of G is $9n + 21$, which can be also achieved by a 2-coloring. If there is a coloring ψ with sum $9n + 20$, then it induces a coloring of H. For a such a coloring ψ, the sum of ψ has to be exactly 20 on the vertices of V^*. Therefore V^* is colored as shown on Figure 4.13, thus the coloring induced by ψ is a precoloring extension of H.

To see the other direction, assume that H has a precoloring extension with 3 colors. This coloring can be extended to a coloring of G having sum $9n + 20$. In V^*, the coloring can be extended to the coloring

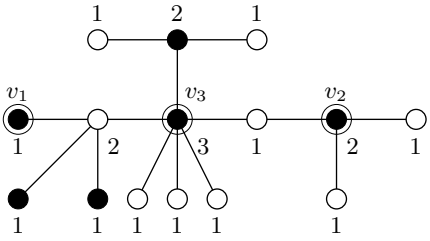


Figure 4.13: The graph that connects the precolored vertices v_1, v_2, v_3, and its unique minimum sum coloring with sum 20.

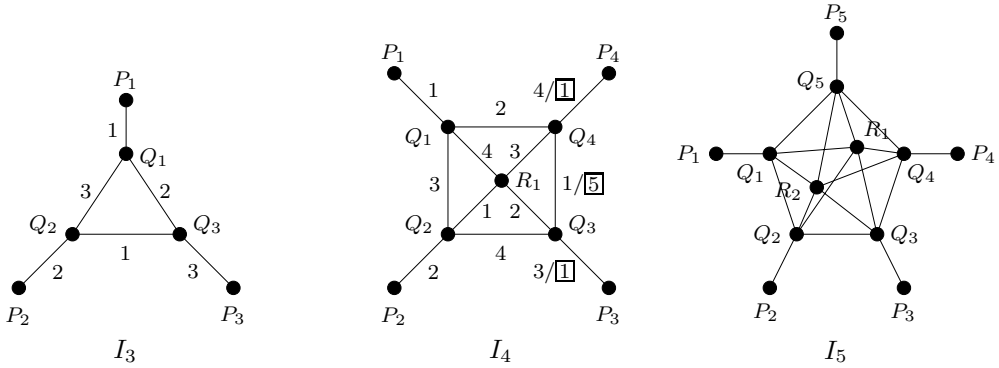


Figure 4.14: Izbicki's graphs for $k = 3$, 4, 5. In I_4, the sum of the coloring decreases if we use the colors shown in frames.

shown on Figure 4.13. In every V_v, depending on the color of v, the coloring can be extended to one of the three colorings shown on Figure 4.12. The set V^* has sum 20 in the resulting coloring, while every V_v has sum 9. Thus the sum of the coloring is $9n + 20$, and $s(G) = 3$ follows. ■

4.4.2 Graphs with $s'(G) > \chi'(G)$

The aim of this section is to show that for every $k \geq 3$, there is a simple graph G with $\Delta(G) = \chi'(G) = k$ and $s'(G) = k + 1$. This gives a counterexample to the conjecture of Harary and Plantholt [Wes95] for every possible value of k. Notice that for $k = 2$ there are no such graphs: if $\chi'(G) = 2$, then every connected component of G is a path or an even cycle, which can be edge colored optimally with 2 colors.

It turns out that for $k > 3$, the graphs constructed by Izbicki in [Izb64] (long before the conjecture) have the required properties. For every $k \geq 3$, the Izbicki graph $I_k(V_k, E_k)$ is defined as follows (see Figure 4.14):

$$\begin{aligned} V_k &= \{R_s, Q_t, P_t \mid 1 \leq s \leq k-3,\ 1 \leq t \leq k\}, \\ E_k &= \{(R_s, Q_t),\ (Q_t, Q_{t+1}),\ (Q_t, P_t) \mid 1 \leq s \leq k-3,\ 1 \leq t \leq k\}, \end{aligned}$$

where $Q_{k+1} = Q_1$. We note that these graphs were used in [LG83] to reduce the edge coloring problem of multigraphs to the edge coloring of simple graphs. Vertices R_s and Q_t have degree k, while vertices P_t have degree 1, therefore by the following lemma, the k edges (Q_t, P_t) have different color in every k-edge-coloring of I_k.

Lemma 4.4.4 (Izbicki [Izb64]). *Let G be a graph that contains only degree 1 and degree k vertices, denote by n the number of vertices in G having degree k. In every k-edge-coloring of G, if f_i $(1 \le i \le k)$ denotes the number of pendant edges with color i, then f_i has the same parity as n.*

Proof. If vertex v has degree k, then every color appears at v in every k-edge-coloring. Therefore with the above notations, color i appears at exactly $n + f_i$ vertices. This number must be even, thus the parities of n and f_i are the same. ■

Since I_k has $n = 2k - 3$ vertices with degree k, thus the Lemma implies that f_i is odd for every $1 \le i \le k$. There are k pendant edges in I_k, hence every f_i is 1, and the pendant edges have different colors. Therefore if I_k has a k-edge-coloring, then this coloring has error $\sum_i^k (i-1) = i(i-1)/2$, since the degree k vertices $R_s,\ Q_t$ have zero error. Moreover, I_k is k-edge-colorable, as shown by the following coloring ψ:

$$\begin{aligned} \psi(R_s, Q_t) &= [t+s+2]_k \ (1 \le s \le k-3,\ 1 \le t \le k), \\ \psi(Q_t, P_t) &= t \ (1 \le t \le k), \\ \psi(Q_t, Q_{t+1}) &= [t+2]_k \ (1 \le t \le k), \end{aligned}$$

where $[x]_k = x - k$ for $x > k$, and $[x]_k = x$ for $x \le k$.

Now consider the coloring ψ' that is the same as ψ except that

$$\begin{aligned} \psi'(Q_{k-1}, P_{k-1}) &= 1 \text{ instead of } k-1, \\ \psi'(Q_{k-1}, Q_k) &= k+1 \text{ instead of } 1, \\ \psi'(Q_k, P_k) &= 1 \text{ instead of } k. \end{aligned}$$

This modification increases the sum by $(1 + (k+1) + 1) - ((k-1) + 1 + k) = 3 - k$, which is negative if $k > 3$. Therefore I_k (for $k > 3$) has a $(k+1)$-edge-coloring with sum strictly smaller than the minimum sum that can be achieved by any k-edge-coloring, hence $s'(I_k) > k = \chi'(I_k) = \Delta(I_k)$.

Proposition 4.4.5. *For every $k > 3$, $\chi'(I_k) = k$ and $s'(I_k) = k + 1$.* ■

For $k = 3$, the graph I_3 does not provide a counterexample to the conjecture of Harary and Plantholt, as the minimum sum 12 can be achieved using only 3 colors (see Figure 4.14). However, the 3-edge-colorable graph shown on Figure 4.15 gives a counterexample for the case $k = 3$. This graph is the smallest counterexample for $k = 3$, and was found by an exhaustive computerized search. The search was performed using the program `nauty` of Brendan McKay [McK90], which is capable of enumerating every non-isomorphic graph with given number of vertices and given maximum degree. For each graph it was first checked whether it is 3-edge-colorable, and if so, then the sum of the best 3-edge-coloring and the best 4-edge-coloring was determined by a simple backtracking method. Checking all the 19430 non-isomorphic connected graphs on 12 vertices with maximum degree 3 took under a minute on a 800MHz computer.

Figure 4.15 shows a 4-edge-coloring of the graph with sum 29. Unfortunately, we cannot give a hand-verifiable proof that this sum cannot be achieved by a 3-edge-coloring. However, a very simple program can check all the $3^{15} \approx 14.3 \cdot 10^6$ possible 3-edge-colorings of the 15 edges, and can verify that the best 3-edge-coloring has sum 30.

Proposition 4.4.6. *For every $k \ge 3$, there is a simple graph G_k with $\Delta(G_k) = \chi'(G_k) = k$ and $s'(G_k) = k + 1$.*

4.4.3 The complexity class Θ_2^p

In the introduction of Section 4.4, we have argued that the problem of deciding whether $s'(G) \le k$ holds does not seem to belong to either NP or coNP. Thus determining the chromatic edge strength of a graph

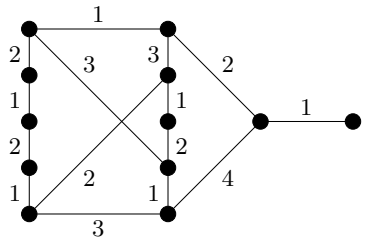


Figure 4.15: A graph with $\Delta(G) = \chi'(G) = 3$ but $s'(G) = 4$. The figure shows a minimum sum edge coloring with sum 29 using 4 colors, every 3-edge-coloring has sum at least 30.

seems to be a problem more difficult than those contained in NP. However, not much more difficult: using an NP-oracle, the value of $\Sigma'(G)$ can be determined with a polynomial number of oracle queries, and with an additional query it can be decided whether there is a k-edge-coloring with sum $\Sigma'(G)$ (for background on oracles, see Appendix A.3). In fact, a logarithmic number of oracle queries is sufficient: the value of $\Sigma'(G)$ can be determined using binary search. Therefore, as an upper bound, it can be said that this problem is in $\mathrm{P}^{\mathrm{NP}} = \Delta_2^p$. But exactly where does this problem lie between NP and Δ_2^p?

The class $\Theta_2^p = \mathrm{P}^{\mathrm{NP}[O(\log n)]}$ contains those languages that can be decided by a polynomial-time oracle Turing-machine that makes $O(\log n)$ sequential queries to an NP-oracle. There are several other natural characterizations of Θ_2^p in the literature: as shown in [Wag90, Hem89, PZ82], it is equivalent to $\mathrm{P}_{||}^{\mathrm{NP}}$ (polynomial-time computation with parallel access to an NP-oracle), $\mathrm{L}_{||}^{\mathrm{NP}}$ (log-space bounded computation with parallel access to an NP-oracle), and L^{NP} (log-space computation with an NP-oracle). The notation Θ_2^p comes from Wagner [Wag90], who defines this class as part of the polynomial hierarchy: Θ_{i+1}^p is the class of problems that can be decided in polynomial time by at most $O(\log n)$ queries to a Σ_i^p-oracle. Since $\Sigma_1^p = \mathrm{NP}$, thus Θ_2^p is equivalent to $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$. Clearly, $\Sigma_i^p, \Pi_i^p \subseteq \Theta_{i+1}^p \subseteq \Delta_{i+1}^p$. It is conjectured that these inclusions are proper. However, our present knowledge does not even rule out the possibility of P = PSPACE.

There are some more exotic characterizations of Θ_2^p. For example, Lange and Reinhardt [LR94] introduced the concept of empty alternation, and proved the surprising result that log-space and polynomial-time bounded computation with auxiliary Turing tape and empty alternation equals Θ_2^p. Holzer and McKenzie [HM00] gave similar characterizations of Θ_2^p using auxiliary stacks (see also [HM02]).

The class Θ_2^p turns out to be relevant in other ways as well. Mahaney [Mah82] has shown that if NP has a sparse Turing-complete set, then the polynomial hierarchy (PH) collapses to Δ_2^p. Kadin [Kad89] has strengthened this result by showing that if NP has sparse Turing-complete sets, then $\mathrm{PH} \subseteq \Theta_2^p$. Moreover, this theorem is optimal in the sense that the collapse to Θ_2^p relativizes, but there are relativized worlds with sparse NP-complete sets where PH does not collapse bellow Θ_2^p [Kad89].

If a complexity class has several natural complete problems, then this makes the class natural and worth studying. The abundance of complete problems is usually taken as a sign that the class captures some important aspect of computation. Wagner [Wag87] has shown that NP-hard optimization problems often give rise to Θ_2^p-complete decision problems. For example, it is Θ_2^p-complete to decide whether the size of the maximum independent set in G is odd, or to decide whether the maximum independent sets of graphs G_1 and G_2 have the same size.

Besides these somewhat technical problems, Θ_2^p has more natural complete problems. The following greedy algorithm is a well-known heuristic for the maximum independent set problem: take a vertex with minimum degree, put it into the independent set, delete it and its neighbors from the graph, and continue this while there are vertices in the graph. In general, this will not result in a maximum independent set, but in certain graphs, with a lucky sequence of choices, it is possible that the result is optimal. Hemaspaandra and Rothe [HR98] showed that it is Θ_2^p-complete to decide whether the greedy algorithm can find a

maximum independent set in the given graph G. More generally, for every rational number $r \geq 1$, they show that it is Θ_2^p-complete to decide whether the greedy algorithm can find an r-approximation of the optimum, that is, an independent set of size at least $1/r$ times the maximum.

Another example can be found in the study of electoral systems. The Condorcet Paradox states that even if every voter has a clear preference order of the candidates, it is not necessary that there is a "best" candidate who can beat every other candidate in pairwise comparisons (such a candidate is called a *Condorcet winner*). In 1876 Lewis Caroll proposed an electoral system that can be used to find a winner even if there is no such best candidate: let that candidate be the winner who can become a Condorcet winner with a minimal number of changes in the preferences of the voters. Hemaspaandra et al. [HHR97, HH00] showed that it is Θ_2^p-complete to decide whether candidate X is the winner in this system. It is quite fascinating to see that there is a Θ_2^p-complete problem that was posed more than 100 years before the definition of the class Θ_2^p.

In Section 4.4.4, we show that for every $k \geq 3$, deciding whether $s'(G) \leq k$ holds is Θ_2^p-complete. In order to prove this result, we introduce four new Θ_2^p-complete variants of the maximum independent set problem:

Maximum Independent Set with $\hat{v}$

Input: A graph $G(V, E)$ and a distinguished vertex $\hat{v} \in V$

Question: [Does at least one/Does every] maximum independent set in G [contains/avoids] $\hat{v}$?

The four problems are abbreviated MIS-$\exists\in$, MIS-$\exists\notin$, MIS-$\forall\in$, MIS-$\forall\notin$ (the symbol $\exists$ stands for "Does at least one," $\forall$ stands for "Does every," $\in$ stands for "contains," and $\notin$ stands for "avoids"). The rest of this section is devoted to the proof that these four problems are Θ_2^p-complete for 3-regular graphs. This result might be of independent interest as well. Wagner [Wag87] presents a general method of turning NP-complete problems into Θ_2^p-complete problems, our proof follows this path.

Theorem 4.4.7 (Wagner [Wag87]). *Let A be an* NP*-complete set and let χ_A be its characteristic function. For an arbitrary set B, if there is a polynomial-time computable function f such that for all $k \geq 1$ and for strings $w_1, w_2, \ldots, w_{2k} \in \Sigma^*$ with $\chi_A(w_1) \geq \chi_A(w_2) \geq \cdots \geq \chi_A(w_{2k})$ it holds that*

$$|\{w_i : w_i \in A\}| \text{ is odd} \iff f(w_1, \ldots, w_{2k}) \in B,$$

then B is Θ_2^p-hard.

That is, given a sequence of words, it is Θ_2^p-hard to decide whether an odd number of the words belong to the NP-complete set A, and if this problem is many-one reducible to B, then B is Θ_2^p-hard as well. Moreover, the problem remains Θ_2^p-hard when restricted to sequences satisfying the technical condition $\chi_A(w_1) \geq \chi_A(w_2) \geq \cdots \geq \chi_A(w_{2k})$. This condition means that we have to consider only sequences where $w_1, \ldots, w_\ell \in A$ and $w_{\ell+1}, \ldots, w_{2k} \notin A$ hold for some ℓ.

Theorem 4.4.8. *All four problems MIS-$\exists\in$, MIS-$\exists\notin$, MIS-$\forall\in$, MIS-$\forall\notin$ are Θ_2^p-complete for 3-regular graphs.*

Proof. To see that these problems belong to the class Θ_2^p, observe that by using binary search, a logarithmic number of adaptive NP-oracle calls are sufficient to determine $\alpha(G)$, the size of the maximum independent set in the graph. Having done that, a single NP or coNP question can answer whether there is an independent set, whether every independent set of size $\alpha(G)$ has the required property.

First we show the Θ_2^p-completeness of the MIS-$\forall\in$ problem using the method given by Theorem 4.4.7. By Cook's Theorem, one can compute in polynomial time a 3CNF formula φ_i for every string w_i such that $w_i \in A$ if and only if φ_i is satisfiable. It can be assumed that every variable in φ_i occurs at most twice positively and at most once negated, it is known that the satisfiability problem remains NP-complete with

these restrictions (cf. [Pap94]). Moreover, we show that φ_i can be modified in such a way that there is a clause $C_{i,1}$ in the modified formula φ_i' that contains only a single literal, and there is a variable assignment of φ_i' satisfying every clause except $C_{i,1}$. In order to obtain φ_i', replace every variable x_j with two variables x_j^1 and x_j^2, replace in the formula every literal x_j with x_j^1, and every literal $\overline{x}_j$ with $\overline{x}_j^2$. Add $n+1$ new variables $y_0, y_1, \ldots, y_n$, where n is the number of variables in the formula φ_i. For every $1 \leq j \leq n$, add a clause $(\overline{x}_j^1 \vee x_j^2 \vee \overline{y}_j)$. Create a cycle of implications by adding the $n+1$ clauses $(y_0 \vee \overline{y}_1)$, $(y_1 \vee \overline{y}_2)$, $\ldots$, $(y_n \vee \overline{y}_0)$, these clauses ensure that every variable y_j has the same value in a satisfying assignment. Finally, add a clause $C_{i,1} = (y_0)$. Notice that every variable occurs at most 3 times in φ_i'. If φ_i is satisfiable, then φ_i' is satisfiable as well: given a satisfying variable assignment of φ_i, setting $x_j^1 = x_j^2 = x_j$ and setting every y_j to true satisfies formula φ_i'. On the other hand, in every satisfying assignment of φ_i', the clause (y_0) implies that y_0 is true, and the cycle of implications ensures that every y_j is true. Therefore $(\overline{x}_j^1 \vee x_j^2)$ holds for every j, it is not possible that x_j^1 is true and x_j^2 is false at the same time. Thus setting $x_j = x_j^1$ gives a satisfying assignment of φ_i. Furthermore, even if φ_i' is not satisfiable, setting every x_j^1 to true and every x_j^2, y_j to false satisfies every clause except $C_{i,1}$. Therefore φ_i' has indeed the required properties.

Denote by n_i and m_i the number of variables and the number of clauses in φ_i'. For every formula φ_i', we apply the standard reduction from 3SAT to maximum independent set (cf. [GJ79]), denote by $G_i(V_i, E_i)$ the graph obtained. That is, G_i contains an edge $x_\ell \overline{x}_\ell$ for every variable x_ℓ, and a clique of size $|C_{i,p}|$ for every clause $C_{i,p}$. Furthermore, the qth vertex in the clique corresponding to the clause $C_{i,p}$ is connected to the negation of the qth literal in clause $C_{i,p}$. Let v_i be the vertex corresponding to the clause $C_{i,1}$ (it contains only one literal). If φ_i' is satisfiable, then G_i has a maximum independent set of size $n_i + m_i$: select the n_i vertices corresponding to the true literals in the satisfying assignment, and select one vertex from each of the m_i cliques $C_{i,p}$. This can be done for every clique: since at least one literal is true in each clause of the formula, at least one vertex of each clique is not adjacent to the n_i selected vertices. Furthermore, even if φ_i' is not satisfiable, G_i has a maximum independent set of size $n_i + m_i - 1$ that does not contain the vertex v_i (the assignment satisfying every clause other that $C_{i,1}$ yields such an independent set). Notice that the constructed graph G_i has maximum degree 3: this follows from the fact that each literal occurs at most twice in φ_i'.

Let us construct the graph G as follows (see Fig. 4.16). Graph G contains k triangles $c_i u_{2i-1} u_{2i}$ $(1 \leq i \leq k)$, a cycle of $2k$ vertices $a_1, b_1, a_2, b_2, \ldots, a_k, b_k$, and k additional vertices $q_2, q_4, \ldots, q_{2k}$. Add the $2k$ graphs G_1, G_2, $\ldots$, G_{2k} to G in the following way: for every odd i, connect the vertex v_i of G_i to u_i, for every even i, connect vertex v_i with q_i, and connect q_i with u_i. Moreover, for every $1 \leq i \leq k$, connect a_i with c_i. The resulting graph is shown on Figure 4.16 for $k = 4$. It is clear that constructing the graph G from the strings $w_1, \ldots, w_{2k}$ is a polynomial-time transformation. Notice that every vertex has degree at most 3 in the resulting graph. However, G is not 3-regular, we will handle this problem later.

Partition the vertices of the graph the following way:

- $C = \{a_1, b_1, \ldots, a_k, b_k\}$,
- $T_j = \{c_j u_{2j-1} u_{2j}\}$ for $1 \leq j \leq k$,
- $X_i = V_i$ for every odd $1 \leq i \leq 2k$,
- $X_i = V_i \cup \{q_i\}$ for every even $1 \leq i \leq 2k$,

where V_i is the vertex set of G_i. Clearly $\alpha(G) \leq \alpha(C) + \sum_{j=1}^{k} \alpha(T_j) + \sum_{i=1}^{2k} \alpha(X_i) = \sum_{i=1}^{2k} \alpha(X_i) + 2k$. Moreover, equality holds, since if we take a maximum independent set in every X_i, then the $2k$ vertices b_i, c_i $(1 \leq i \leq k)$ can be added to this set to obtain an independent set of the required size. This implies that every maximum independent set of G induces a maximum independent set for every class of the partition. In particular, every maximum independent set contains exactly one vertex from every triangle T_j, and contains the k vertices $a_1, \ldots, a_k$ or the k vertices $b_1, \ldots, b_k$ from C. Notice that $\alpha(X_i) = n_i + m_i$ for every even i, regardless of the satisfiability of φ_i': if φ_i' is satisfiable, then there is an independent set

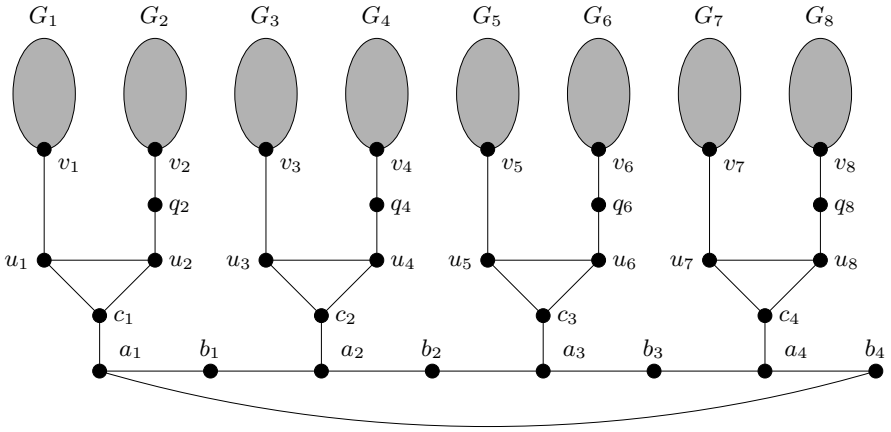


Figure 4.16: The graph G for $k = 4$ in the proof of Theorem 4.4.8.

of size $n_i + m_i$ in G_i, but every such set contains v_i, thus these sets cannot be extended with the vertex q_i. If φ_i' is not satisfiable, then there is an independent set of size $n_i + m_i - 1$ in G_i not containing the vertex v_i, therefore it can be extended with vertex q_i to an independent set of size $n_i + m_i$.

We claim that every maximum independent set of G contains $\hat{v} := b_1$ if and only if $|\{w_i : w_i \in A\}|$ is odd. Assume first that $|\{w_i : w_i \in A\}| = 2d - 1$ is odd. By the assumption $\chi_A(w_1) \geq \chi_A(w_2) \geq \cdots \geq \chi_A(w_{2k})$ this means that $w_{2d-1} \in A$ but $w_{2d} \notin A$. Therefore φ'_{2d-1} is satisfiable, and graph G_{2d-1} has an independent set of size $n_{2d-1} + m_{2d-1}$, but every such set contains vertex v_{2d-1}. Moreover, φ_{2d} is not satisfiable, and as it is noted in the previous paragraph, this implies that every maximum independent set of X_{2d} contains the vertex q_{2d}. Since every maximum independent set S of G induces a maximum independent set for X_{2d-1} and X_{2d}, hence $v_{2d-1}, q_{2d} \in S$ and $u_{2d-1}, u_{2d} \notin S$. As noted above, set S contains exactly one vertex from the triangle T_d, this must be c_d, implying that $a_d \notin S$. Therefore $b_1, b_2, \ldots, b_k \in S$ follows, that is, every maximum independent set of G contains b_1.

Now suppose that $|\{x_i : x_i \in A\}| = 2d$ is even. We show that there is a maximum independent set S of G not containing b_1. Let the set S contain the vertices $a_1, a_2, \ldots, a_k$, and a maximum independent set of every subset X_i. If one can choose from every triangle T_j a vertex that does not conflict with S, then S can be extended to a maximum independent set not containing b_1. The assumptions $|\{w_i : w_i \in A\}|$ even and $\chi_A(w_1) \geq \chi_A(w_2) \geq \cdots \geq \chi_A(w_{2k})$ imply that for every $1 \leq j \leq k$, either $w_{2j-1}, w_{2j} \notin A$ or $w_{2j-1}, w_{2j} \in A$. In the first case, φ'_{2j-1} is not satisfiable, G_{2j-1} has a maximum independent set not containing v_{2j-1}, therefore it can be assumed that $v_{2j-1} \notin S$, and S can be extended with u_{2j-1}. On the other hand, if $w_{2j} \in A$, then there is a maximum independent set of X_{2j} that contains vertex v_{2j}, hence $v_{2j} \in S$, $q_{2j} \notin S$, and S can be extended with u_{2j}.

Notice that this reduction can be used to prove the Θ_2^p-completeness of MIS-$\forall\notin$ as well: every maximum independent set contains b_1 if and only if every maximum independent set excludes a_1. Moreover, since MIS-$\exists\notin$ is the complement of MIS-$\forall\in$, and MIS-$\exists\in$ is the complement of MIS-$\forall\notin$, it follows that these problems are Θ_2^p-complete as well, because Θ_2^p is closed under taking complements.

The theorem requires that we prove the Θ_2^p-completeness of the problems for 3-regular graphs, but G has vertices with degree less than 3. If a vertex v of $G(V, E)$ has degree less than 3, then attach to v one or two gadgets to make the degree of v exactly 3, Figure 4.17 shows the attached gadget. Assume that g such gadgets are attached, denote by $G'(V', E')$ the 3-regular graph obtained. A gadget can contain at most 2 independent vertices, thus $\alpha(G') \leq \alpha(G) + 2g$. On the other hand, given an independent set of G, adding to this set the vertices a, b of every gadget yields an independent set of G', therefore $\alpha(G') = \alpha(G) + 2g$. Moreover, every maximum independent set of G can be extended to a maximum independent set of G',

and if S is a maximum independent set of G', then $S \cap V$ induces a maximum independent set for G. Therefore every maximum independent set of G contains b_1 if and only if every maximum independent set of G' contains b_1, hence the theorem is proved for 3-regular graphs as well. ■

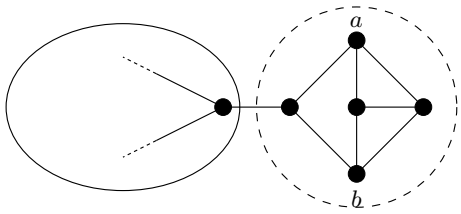


Figure 4.17: Attaching gadgets to the graph to make it three regular.

Replacing maximum independent set with minimum vertex cover in the problem definition results in four new problems MVC-$\exists\in$, MVC-$\exists\notin$, MVC-$\forall\in$, MVC-$\forall\notin$. From the well-known fact that every maximum independent set is the complement of a minimum vertex cover, it follows that these four problems are equivalent to the four problems MIS-$\exists\notin$, MIS-$\exists\in$, MIS-$\forall\notin$, MIS-$\forall\in$, respectively. For example, there is a minimum vertex cover containing vertex $\hat{v}$ if and only if there is a maximum independent set not containing $\hat{v}$.

Corollary 4.4.9. *All four problems MVC-$\exists\in$, MVC-$\exists\notin$, MVC-$\forall\in$, MVC-$\forall\notin$ are Θ_2^p-complete for 3-regular graphs.* ■

4.4.4 The reduction

For every $k \geq 3$, we prove that it is Θ_2^p-complete to decide whether $s'(G) \leq k$ holds for a given graph G. The proof is very similar to the proof of Theorem 4.2.3, where we have shown that minimum sum edge coloring is APX-hard. However, now the reduction is from the MVC-$\exists\in$ problem defined in Section 4.4.3, and we have to use a special gadget for the distinguished vertex $\hat{v}$. Given a 3-regular graph G, we construct a quasigraph G' such that $\epsilon(G') = \tau(G) + c_k$, where $\tau(G)$ is the size of the minimum vertex cover in G, and c_k is a constant depending only on k. Moreover, the minimum error $\tau(G) + c_k$ can be achieved by a k-edge-coloring of G' if and only if there is a minimum vertex cover of G containing distinguished vertex $\hat{v}$. This means that $s'(G) \leq k$ if and only if the answer to the MVC-$\exists\in$ problem is yes. The constructed graph G' is a quasigraph, but we want to prove that determining the edge strength is Θ_2^p-hard for simple graphs. However, this is not a problem, as the transformation of Proposition 4.1.1 gives us a simple graph G'' with the same edge strength as G'.

Quasigraph G' is constructed by associating vertex gadgets and edge gadgets to the vertices and edges of G. The *vertex gadget* shown on Figure 4.5 has 3 pendant edges that correspond to the 3 edges incident to the vertex in G. The coloring of the pendant edges will determine whether we select the vertex into the vertex cover or not. If the vertex is in the vertex cover, then all 3 pendant edges are colored with color 2, otherwise the pendant edges have color 1. The gadget has the following properties:

- There is a coloring ψ with zero error on the internal vertices of the vertex gadget that colors all three pendant edges with color 1. Moreover, every coloring with zero error on the internal vertices colors the pendant edges with color 1.

- There is a coloring ψ^* that colors all three pendant edges with color 2 and has only 1 error on the internal vertices.

Figure 4.5 shows two edge colorings of the vertex gadget. The first coloring has zero error on the internal vertices and assigns color 1 to the pendant edges. The error of the second coloring is 1 (there is error only at vertex v), and assigns color 2 to the pendant edges. Moreover, the first coloring is the unique coloring with zero error on the internal vertices: the reader can easily verify this by observing that the edges incident to degree 1 vertices have to be colored with color 1, and the implications of this uniquely determines the coloring of the rest of the graph. These observations prove that the gadget has the required properties.

The *edge gadget* shown on Figure 4.6 has two pendant edges f and g. If a coloring has zero error on the internal vertices of the gadget, then clearly edges f and g have color 1 or 2. There are 4 different ways of coloring f and g with these two colors. In 3 out of 4 of these combinations, when at least one of f and g is colored with color 2, the coloring can be extended to the whole gadget with zero error (Figure 4.6 shows these 3 different colorings). On the other hand, if both f and g have color 1, then there is at least one error on the internal vertices of the gadget. The reader can verify this by following the implications of coloring f and g with color 1, and requiring that every internal vertex has zero error.

For the distinguished vertex $\hat{v}$, a more complicated gadget is required than the vertex gadget shown on Figure 4.5. Like the vertex gadget, the *special vertex gadget* has a low-error coloring that assigns color 1 to the three pendant edges, and there is a coloring with error greater by 1 that assigns color 2 to the pendant edges. Furthermore, the low-error coloring can be achieved only with $\Delta + 1$ colors, while the other coloring uses only Δ colors. The following lemma states formally the properties of this gadget:

Lemma 4.4.10. *(Special vertex gadget) For every $k \geq 3$, if there is a simple graph H_k with $\chi'(H_k) = \Delta(H_k) = k$ and $s'(H_k) = k + 1$, then there is a quasigraph D_k satisfying the following requirements:*

1. *D_k has three pendant edges. Denote by V_0 the internal vertices of D.*
2. *Every edge coloring ψ with $\epsilon_\psi(V_0) = \epsilon(V_0)$ uses at least $k + 1$ colors and assigns color 1 to the three pendant edges.*
3. *There is a $(k + 1)$-edge-coloring ψ with $\epsilon_\psi(V_0) = \epsilon(V_0)$ that assigns color 1 to the three pendant edges.*
4. *There is a k-edge-coloring ψ^* with $\epsilon_{\psi^*}(V_0) = \epsilon(V_0) + 1$ that assigns color 2 to the three pendant edges.*

The proof of Lemma 4.4.10 is deferred to Section 4.4.5. We note here that the vertex gadget of Figure 4.5 satisfies these properties with $\epsilon(V_0) = 0$, except for Property 2.

Theorem 4.4.11. *For every $k \geq 3$, if there is a graph H_k with $\chi'(H_k) = \Delta(H_k) = k$ and $s'(H_k) = k+1$, then it is Θ_2^p-complete to decide whether $s'(G) \leq k$.*

Proof. The proof is by reduction from the MVC-$\exists\in$ problem, which was proved Θ_2^p-complete in Section 4.4.3 (Corollary 4.4.9). Given a 3-regular graph G with a distinguished vertex $\hat{v}$, we construct a quasigraph G' with maximum degree k such that $s'(G) = k$ if and only if there is a minimum vertex cover of G containing $\hat{v}$.

The quasigraph $G'(V', E')$ is constructed as follows. At first, let us forget about the distinguished vertex $\hat{v}$, consider it as an ordinary vertex like all the others. Later we will show what modifications have to be done for $\hat{v}$. A vertex gadget S_v corresponds to every vertex v of G, and an edge gadget S_e corresponds to every edge e of G. Direct the edges of G arbitrarily. If the ith edge incident to $v \in V$ $(i = 1, 2, 3)$ is the head of $e \in E$, then identify edge e_i of S_v with edge f of S_e. If the ith edge incident to $v \in V$ is the tail of $e \in E$, then identify edge e_i of S_v with edge g of S_e. Thus every vertex of G' is an internal vertex of a vertex gadget S_v or an edge gadget S_e. Denote by V_v the internal vertices of gadget S_v and by V_e the internal vertices of S_e, clearly these sets form a partition of V'.

We claim that G' has an edge coloring with error t if and only if G has a vertex cover with size t. Assume first that $D \subseteq V$ is a vertex cover of G. If $v \in D$, then color gadget S_v using coloring ψ^*: every pendant edge has color 2 and there is 1 error on the internal vertices. If $v \notin D$, then we use coloring ψ of the vertex gadget: every pendant edge of S_v has color 1 and there is no error on the internal vertices. Now consider a gadget S_e for some $e \in E$. The two pendant edges f and g are already colored with color 1 or 2. However, at least one of them is colored with 2, since at least one end vertex of e is in D. Therefore, using one of the three colorings shown on Figure 4.6, we can extend the coloring to every edge of S_e with zero error on the internal vertices of the gadget. Therefore errors appear only on the internal vertices of S_v for $v \in D$, and the total error of the coloring is $|D|$.

On the other hand, consider an edge coloring of G' with error t. Let $\widehat{V} \subseteq V$ be the set of those $v \in V$ for which there is error in V_v. Similarly, let $\widehat{E} \subseteq E$ be the set of those $e \in E$ for which there is error in V_e. Clearly, the coloring has error at least $|\widehat{V}| + |\widehat{E}| \leq t$. Let $\overline{V}$ be a set of $|\widehat{E}|$ vertices in G that cover every edge in $\widehat{E}$. The set of vertices $\widehat{V} \cup \overline{V}$ has size at most $|\widehat{V}| + |\widehat{E}| \leq t$. We show that this set is a vertex cover of G. It is clear that every edge $e \in \widehat{E}$ is covered, since there is a $v \in \overline{V}$ covering e. Now consider an edge $e \notin \widehat{E}$, this means that V_e is colored with zero error, thus, as we have observed, at least one pendant edge of S_e is colored with color 2. If this edge is the pendant edge of the vertex gadget S_v, then there is at least one error in V_v and v is in $\widehat{V}$. However, if the pendant edge of S_e and S_v is identified in the construction, this means that e is incident to v, thus $v \in \widehat{V}$ covers e.

We have shown that $\epsilon(G') = \tau(G)$. Now we modify slightly G' to take into account the distinguished vertex $\hat{v}$. The gadget corresponding to vertex $\hat{v}$ is not the vertex gadget of Figure 4.5, but the special vertex gadget defined in Lemma 4.4.10. By modifying appropriately the argument presented above, one can show that $\epsilon(G') = \tau(G) + \epsilon(V_0)$, where $\epsilon(V_0)$ is the minimum error on the internal vertices of the special gadget. Moreover, if G has a minimum vertex cover D containing $\hat{v}$, then G' has a minimum sum edge coloring using only k colors, since in this case we can use coloring ψ^* on the special gadget. On the other hand, if there is a minimum sum edge coloring using k colors, then by Property 2 of Lemma 4.4.10, the error is more than $\epsilon(V_0)$ on the internal vertices of the special gadget. This means that vertex $\hat{v}$ is contained in the set $\widehat{V}$ defined above, hence the constructed minimum vertex cover contains $\hat{v}$. Therefore $s'(G') = k$ if and only if G has a minimum vertex cover containing $\hat{v}$, what we had to prove. ■

In Section 4.4.2 we have seen that for every $k \geq 3$, there is a simple graph with maximum degree and chromatic index k that has edge strength $k+1$. Combining this with Theorem 4.4.11 gives

Corollary 4.4.12. *For every $k \geq 3$, it is Θ_2^p-complete to decide whether $s'(G) \leq k$.* ■

Determining the chromatic edge strength is the special case of determining the chromatic strength: edge strength is simply the strength of the line graph. Therefore Corollary 4.4.12 implies hardness for the chromatic strength as well:

Corollary 4.4.13. *For every $k \geq 3$, it is Θ_2^p-complete to decide whether $s(G) \leq k$.* ■

In the introduction, we have noted that if G is a simple graph, then $s'(G)$ is either $\Delta(G)$ or $\Delta(G)+1$ [MMS97, HMT00], and consequently, $s'(G)$ is either $\chi'(G)$ or $\chi'(G)+1$. In Theorem 4.4.11, we construct a graph with maximum degree and chromatic index k. Therefore comparing $s'(G)$ to $\Delta(G)$ or to $\chi'(G)$ is also hard:

Corollary 4.4.14. *For every $k \geq 3$, it is Θ_2^p-complete to decide for graphs with maximum degree k whether $s'(G) = \Delta(G)$ holds.* ■

Corollary 4.4.15. *For every $k \geq 3$, it is Θ_2^p-complete to decide for graphs with maximum degree k whether $s'(G) = \chi'(G)$ holds.* ■

Hajiabolhassan et al. [HMT00] asks an open question to characterize those graphs where $s'(G) \neq \chi'(G)$. Corollary 4.4.15 implies that we cannot hope for a nontrivial (NP or coNP) characterization of such graphs.

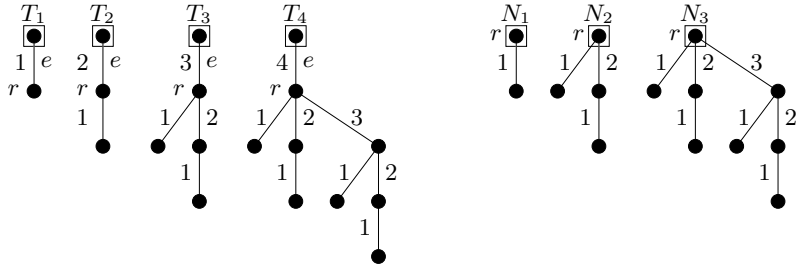


Figure 4.18: The trees T_i and N_i.

4.4.5 Special vertex gadget

The aim of this section is to construct the special vertex gadget defined in Lemma 4.4.10. However, some preparations are required before the proof. We recursively construct two families of trees T_i and N_i ($i \geq 1$). Every T_i has a pendant edge e, and every N_i has a root r. The trees T_1 and N_1 consist of a single edge. The tree N_i is constructed by attaching the pendant edges of a $T_1, T_2, \ldots, T_i$ tree to a common root r. The tree T_{i+1} is the same as N_i, with a pendant edge connected to the root r. The construction is demonstrated on Figure 4.18.

The properties of these trees are summarized in the following lemma:

Lemma 4.4.16. *(a) There is an edge coloring of the tree T_i that has no error on the internal vertices of T_i, and assigns color i to the pendant edge e. Furthermore, every coloring that assigns color j to e has error at least $|j-i|$ on the internal vertices.*
(b) There is a zero error edge coloring of the tree N_i that assigns the colors $1, 2, \ldots, i$ to the edges incident to r. Furthermore, if color $j \leq i$ is missing at r in a coloring, then this coloring has error at least $i-j+1$ on the internal vertices of N_i.

Proof. The proof is by induction on i. Both statements are trivial for $i = 1$. Now assume that $i > 1$ and both (a) and (b) hold for every $1 \leq j < i$. First we prove statement (a). Since $T_i - e$ is an N_{i-1} tree, thus it has a zero error coloring by the induction hypotheses. Extending this coloring by assigning color i to edge e does not create errors on the internal vertices of T_i, proving the first part of statement (a). Consider now an edge coloring that assigns color j to e. This coloring colors $T_i - e = N_{i-1}$ in such a way that color j is missing from vertex r. If $j < i$, then by the induction hypothesis, there is error at least $(i-1)-j+1 = |j-i|$ on the internal vertices of N_{i-1}, and we are ready. On the other hand, if $j > i$, then in the coloring of T_i the degree i internal vertex r has error at least $j-i$.

Next we prove statement (b). Let $e_1, e_2, \ldots, e_i$ be the edges incident to r in N_i, edge e_j is the pendant edge of the T_j tree attached to r. A zero error edge coloring of N_i can be obtained by coloring every attached tree T_j in such a way that the internal vertices have zero error and edge e_j has color j. Clearly, there is no error on r on any other vertex of N_i in this coloring.

Suppose that a color $j \leq i$ is missing from r in a coloring ψ of N_i. Define the following sequence of edges: $e_{s_1} = e_j$ and $e_{s_{k+1}} = e_{\psi(e_{s_k})}$ until an edge with $\psi(e_{s_{k'}}) > i$ is found (it can be verified that this sequence is finite). Since e_{s_k} is the pendant edge of a tree T_{s_k}, by statement (a), there is error at least $|s_k - \psi(e_{s_k})|$ on the internal vertices of T_{s_k}. Therefore the internal vertices of N_i has error at least

$$\begin{aligned} &|\psi(e_{s_1}) - s_1| + |\psi(e_{s_2}) - s_2| + \cdots + |\psi(e_{s_{k'-1}}) - s_{k'-1}| + |\psi(e_{s_{k'}}) - s_{k'}| \\ &\geq (\psi(e_{s_1}) - s_1) + (\psi(e_{s_2}) - s_2) + \cdots + (\psi(e_{s_{k'-1}}) - s_{k'-1}) + (\psi(e_{s_{k'}}) - s_{k'}) \\ &= \psi(e_{s_{k'}}) - s_1 \geq i + 1 - j \end{aligned}$$

since by definition, $\psi(e_{s_k}) = s_{k+1}$ for $1 \leq k < k'$, and $\psi(e_{s_{k'}}) > i$. ■

The colorings defined by Lemma 4.4.16 will be called the *standard colorings* of these gadgets.

Denote by $\Sigma'_\Delta(G)$ the minimum sum that a $\Delta(G)$-edge-coloring of G can have, clearly $\Sigma'_\Delta(G) \geq \Sigma'(G)$. Denote by $\epsilon_\Delta(G)$ the error of the best $\Delta(G)$-edge-coloring, that is, $\epsilon_\Delta(G) = 2\Sigma'_\Delta(G) - \ell(G)$.

In the following lemma, we determine how the error on the vertices changes if we attach to vertex v of G_1 an edge of G_2.

Lemma 4.4.17. *Let $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ be two graphs such that $V_1 \cap V_2 = \{v\}$ and edge e is the only edge in G_2 incident to v. Let d be the degree of v in G_1. Let $G(V_1 \cup V_2, E_1 \cup E_2)$ be the graph obtained by joining G_1 and G_2 at vertex v. If ψ_1 is an edge coloring of G_1, ψ_2 is an edge coloring of G_2, and these colorings assign distinct colors to the edges incident to v, then they can be combined to obtain an edge coloring ψ of G such that*

$$\epsilon_\psi(u) = \begin{cases} \epsilon_{\psi_1}(u) & \text{if } u \in V_1 \setminus \{v\} \\ \epsilon_{\psi_2}(u) & \text{if } u \in V_2 \setminus \{v\} \\ \epsilon_{\psi_1}(u) + \psi_2(e) - (d+1) & \text{if } u = v. \end{cases}$$

Conversely, if ψ is an edge coloring of G, then it induces an edge coloring ψ_1 of G_1 such that

$$\epsilon_{\psi_1}(u) = \begin{cases} \epsilon_\psi(u) & \text{if } u \in V_1 \setminus \{v\} \\ \epsilon_\psi(v) - \psi(e) + d + 1 & \text{if } u = v. \end{cases}$$

Proof. The first statement clearly holds for every vertex $u \neq v$, since combining the two colorings can change the error only on v, the only common vertex of the two graphs. Let $E_v \subseteq E_1$ be the edges incident to v in G_1. The error of v in coloring ψ is

$$\begin{aligned} \epsilon_\psi(v) = \Sigma'_\psi(v) - \sum_{i=1}^{d+1} i &= \sum_{f \in E_v} \psi(f) + \psi(e) - \sum_{i=1}^{d} i - (d+1) \\ &= \left(\sum_{f \in E_v} \psi_1(f) - \sum_{i=1}^{d} i \right) + \psi_2(e) - (d+1) = \epsilon_{\psi_1}(v) + \psi_2(e) - (d+1). \end{aligned}$$

The second statement can be proved by a similar calculation. ■

In particular, if we attach a tree $T_{d(v)}$ to a vertex v, then the error changes as follows:

Lemma 4.4.18. *Let v be an arbitrary vertex of simple graph $G(V, E)$, attach to v the pendant edge e of the tree $T_{d(v)}$. Denote by G' the graph obtained.*

(a) The error $\epsilon(G')$ is either $\epsilon(G) - 1$ or $\epsilon(G) + 1$, and it is $\epsilon(G) - 1$ if and only if there is a minimum sum edge coloring ψ of G such that some color $c \leq d(v)$ is missing from v.

(b) If $d(v) < \Delta(G)$, then $\epsilon_\Delta(G')$ is either $\epsilon_\Delta(G) - 1$ or $\epsilon_\Delta(G) + 1$, and it is $\epsilon_\Delta(G) - 1$ if and only if there is a $\Delta(G)$-edge-coloring with error $\epsilon_\Delta(G)$ where some color $c' \leq d(v)$ is missing from v.

Proof. Let ψ be a minimum sum edge coloring of G, and let $c \leq d(v) + 1$ be the smallest color not present at v in ψ. Take the standard coloring of $T_{d(v)}$ that has zero error on the internal vertices and assigns color $d(v)$ to edge e (Lemma 4.4.16a). Exchange in $T_{d(v)}$ the colors $d(v)$ and c on the alternating path starting at edge e, this results in a coloring of $T_{d(v)}$ that has $|d(v) - c|$ error the internal vertices and assigns color c to edge e. This coloring can be combined with ψ to obtain a coloring ψ' of G'. We use Lemma 4.4.17 to calculate the error of ψ'. The total error on the internal vertices of $T_{d(v)}$ is $|d(v) - c|$, and the error on the vertices of G is the same as in ψ, except on v, where the error is increased by $c - (d(v) + 1)$. Therefore the error of ψ' is $\epsilon_{\psi'}(G') = \epsilon_\psi(G) + c - (d(v) + 1) + |d(v) - c|$. If $c \leq d(v)$, then this equals $\epsilon_\psi(G) - 1$, thus

$\epsilon(G') \leq \epsilon(G) - 1$. If $c = d(v) + 1$, then $\epsilon_{\psi'}(G) = \epsilon_{\psi}(G) + 1$, and $\epsilon(G') \leq \epsilon(G) + 1$ follows. Therefore we have obtained that $\epsilon(G') \leq \epsilon(G) + 1$, and if G has a minimum sum edge coloring where a color $c \leq d(v)$ is missing from v, then $\epsilon(G') \leq \epsilon(G) - 1$.

To finish the proof of statement (a), we have to show that $\epsilon(G') \geq \epsilon(G) - 1$, and if every minimum sum edge coloring of G uses at v every color not greater than $d(v)$, then $\epsilon(G') \geq \epsilon(G) + 1$. Assume that a minimum sum coloring ψ' of G' is given with $\psi'(e) = c$. By Lemma 4.4.16 there is error at least $|d(v) - c|$ on the internal vertices of the tree $T_d(v)$, so there is error at most $\epsilon(G') - |d(v) - c|$ on the vertices V. Coloring ψ' induces a coloring ψ of G, and by the second part of Lemma 4.4.17,

$$\epsilon_{\psi}(G) = \epsilon_{\psi'}(V) - c + d(v) + 1 \leq \epsilon(G') - |d(v) - c| - c + d(v) + 1 \leq \epsilon(G') + 1,$$

hence $\epsilon(G') \geq \epsilon_{\psi}(G) - 1 \geq \epsilon(G) - 1$. Moreover, equality is only possible if $c \leq d(v)$ and ψ is a minimum sum edge coloring of G, or in other words, if there is a minimum sum edge coloring of G such that color $c \leq d(v)$ is missing from v. Finally, if every minimum sum coloring of G uses every color not greater than $d(v)$ on v, then either $c > d(v)$ or ψ is not a minimum sum coloring of G. In either case, $\epsilon_{\psi'}(G') \geq \epsilon(G)+1$ follows, completing the proof of statement (a) (recall that if ψ is not a minimum sum coloring of G, then $\epsilon_{\psi}(G) \geq \epsilon(G) + 2$, since the error of every coloring has the same parity). The proof of statement (b) is exactly the same. Notice that if $d(v) < \Delta(G)$, then $\Delta(G') = \Delta(G)$. ■

The following gadget will be used in the construction of the special vertex gadget.

Lemma 4.4.19. *For every $k \geq 1$, there is a quasigraph H_k satisfying the following properties (V_0 denotes the internal vertices of H_k):*

1. *H has two pendant edges f, g.*
2. *There is a $(k+1)$-edge-coloring ψ_{k+1} with $\psi_{k+1}(f) = k+1$, $\psi_{k+1}(g) = 1$ and $\epsilon_{\psi_{k+1}}(V_0) = 0$.*
3. *For every $i \leq k$, there is a $(k+1)$-edge-coloring ψ_i with $\psi_i(f) = i$, $\psi_i(g) = 2$, and $\epsilon_{\psi_i}(V_0) = k - i$.*
4. *For every coloring ψ, if $\psi(f) = i \leq k$, then $\epsilon_{\psi}(V_0) \geq k - i$.*
5. *For every coloring ψ, if $\psi(f) = k+1$ and $\epsilon_{\psi}(V_0) = 0$, then $\psi(g) = 1$.*

Proof. For $k = 1, 2, 3$, the graph H_k is shown on Figure 4.19. It can be verified directly that they satisfy the requirements of the lemma. Henceforth it is assumed that $k \geq 4$.

The graph H_k is constructed as follows. Take a path of 6 vertices $v_1, v_2, v_3, v_4, v_5, v_6$, let $f = v_1v_2$ and $g = v_5v_6$. Identify the root of a tree N_{k-1} with vertex v_2. Attach a half-loop to v_3, and attach to v_3 the pendant edges of $k-2$ trees T_2, T_3, ..., T_{k-1}. Attach a half-loop to v_4 as well, and attach to v_4 the pendant edges of $k-3$ trees T_3, T_4, ..., T_{k-1}. The resulting graph H_k is demonstrated on Figure 4.19.

The coloring ψ_{k+1} is defined as $\psi_{k+1}(v_1v_2) = k+1$, $\psi_{k+1}(v_2v_3) = k$, $\psi_{k+1}(v_3v_4) = 1$, $\psi_{k+1}(v_4v_5) = 2$, $\psi_{k+1}(v_5v_6) = 1, \psi_{k+1}(v_3v_3) = k+1$, $\psi_{k+1}(v_4v_4) = k$, and it gives the standard coloring to the attached trees. It can be verified that ψ_{k+1} is a proper edge coloring and there is zero error on the internal vertices, which gives Property 2. Similarly, the coloring ψ_k required by Property 3 for $i = k$ is defined as $\psi_k(v_1v_2) = k$, $\psi_k(v_2v_3) = k+1$, $\psi_k(v_3v_4) = k$, $\psi_k(v_4v_5) = 1$, $\psi_k(v_5v_6) = 2, \psi_k(v_3v_3) = 1$, $\psi_k(v_4v_4) = 2$.

To obtain the coloring ψ_i for some $i < k$ (Property 3), take the coloring ψ_k defined above, and exchange the colors k and i on the alternating path starting from edge f. This introduces error $k - i$ to only one vertex, namely to the vertex at the other end of the alternating path. Notice that this color exchange cannot affect edge g, since edge v_2v_3 has color $k+1$. Therefore we obtain a coloring satisfying Property 3.

To see that Property 4 holds, observe that a coloring ψ of H_k induces a coloring of the tree N_{k-1}, and color $\psi(f)$ is missing from the root of N_{k-1}. Therefore in this coloring of N_{k-1}, there is error at least $k - 1 - \psi(f) + 1 = k - i$ on the internal vertices (Lemma 4.4.16b), and this means that ψ has error at least $k - i$ on the internal vertices of H_k, as required.

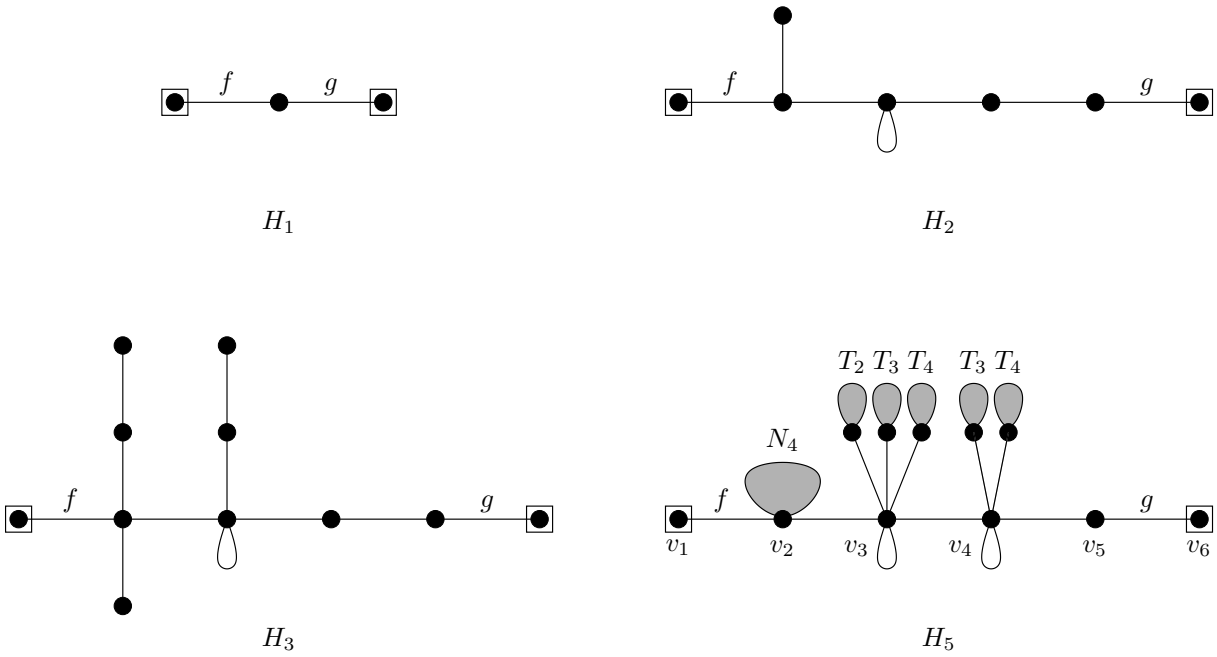


Figure 4.19: The graph H_1, H_2, H_3, and H_5.

To verify Property 5, assume that $\psi(f) = k+1$ and $e_\psi(V_0) = 0$, that is, there is zero error on each internal vertex of the gadget. The color of edge v_2v_3 cannot be less than k, since in that case the tree N_{k-1} could not be colored with zero error on its internal vertices. Vertex v_2 has degree $k+1$, hence the assumption that there is no error on v_2 implies that $\psi(v_2v_3) \leq k+1$. Since color $k+1$ is used by f on v_2, therefore we can conclude that $\psi(v_2v_3) = k$. For $2 \leq i \leq k-1$, edge v_3v_4 cannot have color i, since that would imply that the tree T_i attached to vertex v_3 cannot be colored with zero error. Since vertex v_4 has degree k, and color k is already used at v_3 by edge v_2v_3, it follows that $\psi(v_3v_4) = 1$. This implies in turn that $\psi(v_4v_5) \neq 1$. However, there is zero error on vertex v_5, therefore there must be an edge with color 1 at v_5. Thus edge g has color 1, as required. ■

Now we are ready to construct the special vertex gadget:

Proof of Lemma 4.4.10. By assumption, there exists a graph G with $\Delta(G) = k$ and $s'(G) = k+1$ (or equivalently, $\epsilon(G) < \epsilon_\Delta(G)$). If more than one graph satisfies this condition, then select a graph G such that

$$\epsilon_\Delta(G) - \epsilon(G) > 0 \text{ is minimal,} \qquad (*)$$

and among these graphs,

$$\epsilon(G) \text{ is minimal.} \qquad (**)$$

For every every vertex v of G, we define two sets $\Lambda(v), \Lambda_\Delta(v) \subseteq \{1, 2, \ldots, d(v)\}$. Set $\Lambda(v)$ contains j $(1 \leq j \leq d(v))$ if there is an edge coloring of G with error $\epsilon(G)$ such that j is missing from v. If $\Lambda(v) = \emptyset$, then this means that every minimum sum edge coloring has zero error on v. Similarly, $\Lambda_\Delta(v)$ contains j $(1 \leq j \leq d(v))$, if there is a $\Delta(G)$-edge-coloring with error $\epsilon_\Delta(G)$ such that j is missing from v.

First we show that at least one of $\Lambda(v)$ and $\Lambda_\Delta(v)$ is empty for every vertex v. Otherwise attach the pendant edge of a tree $T_{d(v)}$ to v, let G' be the obtained graph. Since there are colors $j \in \Lambda(v)$, $j' \in \Lambda_\Delta(v)$

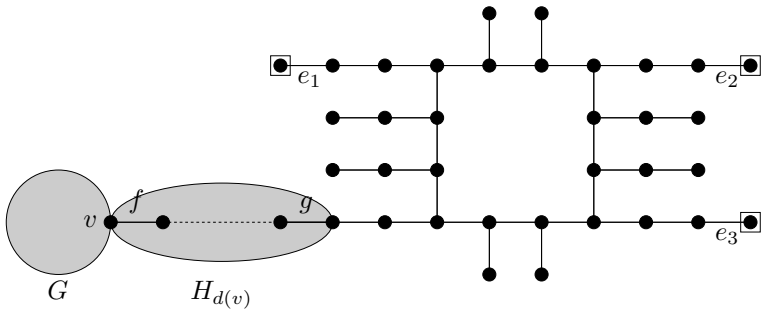


Figure 4.20: The structure of the special vertex gadget.

not greater than $d(v)$, by (a) and (b) of Lemma 4.4.18, we have $\epsilon(G') = \epsilon(G) - 1$ and $\epsilon_\Delta(G') = \epsilon(G) - 1$, which contradicts the minimality of G with respect to (**).

Since $\epsilon_\Delta(G) > 0$, there is at least one vertex v with $\Lambda_\Delta(v) \neq \emptyset$, $\Lambda(v) = \emptyset$. Call such a vertex a *join* vertex (later we will join another gadget to G at such a vertex, hence the name). Notice that $d(v) < \Delta(G)$, since $\Lambda_\Delta(v) \neq \emptyset$ means that there is a $\Delta(G)$-edge-coloring that uses a color greater than $d(v)$ at v.

The error has the same parity in every edge coloring, and $\epsilon_\Delta(G) > \epsilon(G)$ by assumption, thus it follows that $\epsilon_\Delta(G) \geq \epsilon(G) + 2$. We claim that $\epsilon_\Delta(G) = \epsilon(G) + 2$ holds for a minimal graph G. Assume that on the contrary, $\epsilon_\Delta(G) > \epsilon(G) + 2$, and let v be a join vertex in G. Attach to v a tree $T_{d(v)}$ and let G' be the graph obtained. Since $\Lambda_\Delta(v) \neq \emptyset$, there is a $\Delta(G)$-edge-coloring of G with error $\epsilon_\Delta(G)$ such that some color $c \leq d(v)$ is missing from v, thus by Lemma 4.4.18b, $\epsilon_\Delta(G') = \epsilon_\Delta(G) - 1$. Moreover, since $\Lambda(v) = \emptyset$, every color not greater than $d(v)$ is used at v in every minimum sum edge coloring of G, hence $\epsilon(G') = \epsilon(G) + 1$, by Lemma 4.4.18a. From the assumption $\epsilon_\Delta(G) > \epsilon(G) + 2$, it follows that $\epsilon(G') > \epsilon_\Delta(G')$ holds, thus G is not minimal with respect to (*), a contradiction.

Now we are ready to construct the graph D_k. As shown on Figure 4.20, the graph D_k consists of three parts: the minimal graph G defined above, a graph H_i from Lemma 4.4.19, and the variable gadget shown on Figure 4.5. Let vertex v be a join vertex of G. Set $d = d(v)$, and connect to v the pendant edge f of graph H_d. Finally, as shown on the figure, a graph with 34 new vertices is connected to the pendant edge g of H_d. The edges $e_1,\ e_2,\ e_3$ are the pendant edges of D_k.

Denote by V_0 the internal vertices of D_k and let V_G be the vertices of G (including v).

Claim 4.4.20. *If V_0 is the set of internal vertices of D_k, then $\epsilon(V_0) = \epsilon(G)$. Moreover, if $\epsilon_\psi(V_0) = \epsilon(G)$, then coloring ψ uses $\Delta(G) + 1$ colors, $\psi(f) = d + 1$ and $\psi(e_i) = 1$ for $i = 1,\ 2,\ 3$.*

Proof. Color G with error $\epsilon(G)$ such that colors $1, 2, \ldots, i$ appear at vertex v (such a coloring exists, since v is a join vertex and $\Lambda(v) = \emptyset$). Color the edges in H_d using coloring ψ_{d+1} of Lemma 4.4.19, it assigns color $d + 1$ to f, and it does not introduce additional error on v or on the internal vertices of H_i. Since this coloring assigns color 1 to edge g, it can be extended (in a unique way) to the rest of graph D_k without increasing the error on V_0 (similarly as in the case of the vertex gadget of Section 4.4.4). Therefore $\epsilon(V_0) \leq \epsilon(G)$. Notice that this coloring assigns color 1 to the edges $e_1,\ e_2,\ e_3$.

To show that $\epsilon(V_0) \geq \epsilon(G)$, let ψ be an edge coloring with $\epsilon_\psi(V_0) \leq \epsilon(G)$. First we show that $\psi(f) > d$. If not, then by Property 4 of Lemma 4.4.19, ψ has error at least $d - \psi(f)$ on the internal vertices of H_d, hence there can be error at most $\epsilon_\psi(V_0) - (d - \psi(f)) \leq \epsilon(G) - (d - \psi(f))$ on V_G. By the second part of Lemma 4.4.17, this implies that ψ induces a coloring ψ' of G with error at most $\epsilon_{\psi'}(G) \leq \epsilon_\psi(V_G) - \psi(f) + d + 1 \leq \epsilon(G) - (d - \psi(f)) - \psi(f) + d + 1 = \epsilon(G) + 1$. Furthermore, ψ' is not a minimum sum edge coloring of G, since color $\psi(f) \leq d$ is missing from v, and $\Lambda(v) = \emptyset$. Therefore $\epsilon_{\psi'}(G) > \epsilon(G)$, but this also means that $\epsilon_{\psi'}(G) \geq \epsilon(G) + 2$, since the parity of the error is the same in every edge coloring. However, this contradicts $\epsilon_{\psi'}(G) \leq \epsilon(G) + 1$.

Therefore it can be assumed that $\psi(f) > d$ for any coloring with $\epsilon_\psi(V_0) \leq \epsilon(G)$. Now, again by Lemma 4.4.17, ψ induces a coloring ψ' of G with error $\epsilon_{\psi'}(G) = \epsilon_\psi(V_G) - \psi(f) + (d+1) \leq \epsilon_\psi(V_G) \leq \epsilon_\psi(V_0) \leq \epsilon(G)$. Since $\epsilon_{\psi'}(G) \geq \epsilon(G)$, there have to be inequalities throughout. In particular, $\psi(f) = d+1$ and $\epsilon_\psi(V_0) = \epsilon(G)$, thus $\epsilon(V_0)$ cannot be strictly smaller than $\epsilon(G)$. Furthermore, every coloring ψ with $\epsilon_\psi(V_0) = \epsilon(G)$ induces a coloring ψ' of G with error $\epsilon(G)$. We know that error $\epsilon(G)$ can be achieved only by using $\Delta(G)+1$ colors. Therefore $\Delta(G)+1$ colors are required to achieve error $\epsilon(V_0) = \epsilon(G)$ on V_0. Moreover, we have seen that in such a coloring ψ, there is color $d+1$ on f and the error on $V_0 \setminus V$ is zero. By Property 5 of Lemma 4.4.19, this implies that $\psi(g) = 1$ and it follows that the pendant edges $e_1,\ e_2,\ e_3$ also have color 1, as required. ■

Property 2 of the Lemma follows immediately from Claim 4.4.20. Moreover, in the proof we have constructed a coloring ψ with $\epsilon_\psi(V_0) = \epsilon(V_0)$ and $\psi(e_i) = 1$ for $i = 1, 2, 3$, which implies Property 3.

To show that Property 4 holds, color G using $\Delta(G)$ colors with error $\epsilon_\Delta(G) = \epsilon(G)+2$ such that color $c \in \Lambda_\Delta(v)$ is missing from vertex v, denote by ψ_Δ this coloring. Color H_d such that edge f has color d, edge g has color 2, and there is error $d-c$ on the internal vertices of H_d (the coloring ψ_c from Property 3 of Lemma 4.4.19). This coloring can be extended to a coloring of D_k without introducing further errors on V_0 (see the second coloring on Figure 4.5), which gives a coloring ψ^* that assigns color 2 to the three pendant edges $e_1,\ e_2,\ e_3$. We use the first part of Lemma 4.4.17 to determine $\epsilon_{\psi^*}(V_0)$. There is error $d-c$ on $V_0 \setminus V$, and $\epsilon_{\psi^*}(u) = \epsilon_{\psi_\Delta}(u)$ for every $u \in V \setminus \{v\}$. By Lemma 4.4.17, $\epsilon_{\psi^*}(v) = \epsilon_{\psi_\Delta}(v) + \psi^*(f) - (d(v)+1) = \epsilon_{\psi_\Delta}(G) + c - d - 1$. Therefore $\epsilon_{\psi^*}(V_0) = \epsilon_{\psi_\Delta}(G) + (c-d-1) + (d-c) = \epsilon_\Delta(G) - 1 = \epsilon(G) + 1 = \epsilon(V_0) + 1$, as required. ■

CHAPTER 5

Minimum sum multicoloring

There are not more than five primary colors (blue, yellow, red, white, and black),
yet in combination they produce more hues than can ever been seen.
Sun Tzu (circa 500 B.C.), *The Art Of War*

In Chapter 4 we have seen that minimum sum coloring can be used to model the scheduling of dependent jobs if the goal is to minimize the sum of completion times. However, we have assumed that every job requires a unit amount of time, exactly one time slot has to be assigned to each job. In this chapter we generalize minimum sum coloring in such a way that it can model different time requirements.

In the *minimum sum multicoloring* problem a graph $G(V, E)$ is given with a *demand* $x(v)$ for each vertex. In the scheduling application, this demand will correspond to the number of time slots the job requires. The colors are the positive integers, they correspond to the time slots. We have to assign a set of $x(v)$ colors to each vertex v such that neighbors receive disjoint sets of colors. Given a coloring, the completion time of a job will be the largest color assigned to the vertex of the job. Thus it is not important what is the color set assigned to a vertex, only the largest color of this color set counts. Therefore to minimize the sum of completion times, we have to minimize the sum of these largest colors. Formally, the problem is defined as follows:

Minimum Sum Multicoloring

Input: A graph $G(V, E)$ and a *demand function* x: $V \to \mathbb{N}$.

Output: A *multicoloring* Ψ: $V \to 2^{\mathbb{N}}$ such that

- $|\Psi(v)| = x(v)$ for every $v \in V$, and
- $\Psi(u) \cap \Psi(v) = \emptyset$ if u and v are neighbors in G.

Goal: Let the *finish time* of vertex v in coloring Ψ be the highest color assigned to it, $f_\Psi(v) = \max\{i : i \in \Psi(v)\}$. The goal is to minimize $\sum_{v \in V} f_\Psi(v)$, the *sum* of the coloring Ψ.

There are two variants of the problem. In the *preemptive* version the color set $\Psi(v)$ can be arbitrary. In the *non-preemptive* version we require that $\Psi(v)$ contains a consecutive set of colors. This latter variant can be used to model non-preemptive scheduling problems, where it is not allowed to interrupt a job

once it is started. Such non-preemptive problems can arise when we schedule jobs on machines and it is physically impossible (or very expensive) to stop a job and continue it later. On the other hand, in a computer or network setting it is quite common to preempt a task and continue it later, usually this can be done with only a small overhead. The preemptive and non-preemptive versions of a scheduling problem can be very different. In a preemptive problem we have to assign a set of time slots to each job, while in the non-preemptive problem a single number (the starting time or the completion time) is sufficient. Therefore in the non-preemptive problems the search space is usually smaller, which can make exact solution easier. On the other hand, in a preemptive problem we have more tools to modify a solution, which can be useful when designing approximation algorithms. In this chapter we consider only the preemptive version of minimum sum multicoloring. If a set $\Psi(v)$ is assigned to v, then the number of *preemptions* in $\Psi(v)$ is the number of different i's such that $i \leq f_\Psi(v)$, $i \notin \Psi(v)$, but $i-1 \in \Psi(v)$. If $\Psi(v)$ is a continuous interval, then it has zero preemptions; if it is the union of two intervals, then it has one preemption, etc.

Minimum sum multicoloring was introduced in [BNHK$^+$99, BNHK$^+$00]. The problem is hard, and it is hard even to approximate: the best known polynomial-time algorithm has approximation ratio $O(n/\log^2 n)$ [BNHK$^+$00]. The same paper gives a 16-approximation algorithm for perfect graphs and a 7.184-approximation for interval graphs. However, very recently a 5.436-approximation was found for perfect graphs, which improves both results. For bipartite graphs [BNHK$^+$00] gives a 1.5-approximation algorithm. For trees a polynomial time approximation scheme (PTAS) is given by Halldórsson et al. [HKP$^+$03], which was later generalized for partial k-trees and planar graphs [HK02].

In [HKP$^+$03] a PTAS is given for the minimum sum multicoloring of trees, but no hardness result was proved for the problem. It was asked as an open question what is the complexity of minimum sum multicoloring on trees, and if it is NP-hard, then whether it is possible to solve the problem in polynomial time for paths. We answer the first question in Section 5.2 by showing that the problem is NP-hard for binary trees. Kovács [Kov04] gave a partial answer to the second question by showing that minimum sum multicoloring can be solved in $O(n^3p)$ time for a path of length n if the maximum demand is p. This algorithm is only pseudopolynomial, since the maximum demand can be exponentially large. As of this writing, it is still an open question whether there is a polynomial time algorithm for the problem on paths. In Section 5.1 we prove a structural result that might help resolving this question. We show that for paths there is always an optimum solution where every color set is the union of $O(\log p)$ continuous intervals. For bipartite graphs and perfect graphs, we prove the weaker bounds $O(n)$ and $O(n^2)$, respectively, on the number of preemptions required per vertex in an optimum solution.

In Section 5.3 and Section 5.4 we investigate the edge coloring version of minimum sum multicoloring, which can be formulated as follows:

Minimum Sum Edge Multicoloring

Input: A graph $G(V, E)$ and a *demand* function x: $E \to \mathbb{N}$.

Output: A *multicoloring* Ψ: $E \to 2^{\mathbb{N}}$ such that

- $|\Psi(e)| = x(e)$ for every edge e, and
- $\Psi(e_1) \cap \Psi(e_2) = \emptyset$ if e_1 and e_2 are adjacent in G.

Goal: The *finish time* of edge e in coloring Ψ is the highest color assigned to it, $f_\Psi(e) = \max\{c : c \in \Psi(e)\}$. The goal is to minimize $f_\Psi(G) = \sum_{e \in E} f_\Psi(e)$, the *sum* of the coloring Ψ.

An application of edge coloring is to model dedicated scheduling of biprocessor tasks. The vertices correspond to the processors and each edge $e = uv$ corresponds to a job that requires $x(e)$ time units of simultaneous work on the two preassigned processors u and v. The colors correspond to the available time slots: by assigning $x(e)$ colors to edge e, we select the $x(e)$ time units when the job corresponding to e is

executed. A processor cannot work on two jobs at the same time, this corresponds to the requirement that a color can appear at most once on the edges incident to a vertex. The finish time of edge e corresponds to the time slot when job e is completed, therefore minimizing the sum of the finish times is the same as minimizing the sum of completion times of the jobs. Such biprocessor tasks arise when we want to schedule file transfers between processors [CGJL85] or the mutual diagnostic testing of processors [HvdVV94]. Note that it is allowed that a job is interrupted and continued later: the set of colors assigned to an edge does not have to be consecutive, hence our problem models preemptive scheduling.

Of particular interest is the case where the graph to be colored is bipartite. A possible application of the bipartite problem is the following. One bipartition class corresponds to a set of clients, the other class corresponds to a set of servers. An edge e between two vertices means that the given client has to access the given server for $x(e)$ units of time. A client can access only one server at the same time, and a server cannot accept connections from more than one client simultaneously. Clearly, bipartite edge multicoloring models this situation.

Minimum sum edge multicoloring is a generalization of minimum sum edge coloring, hence any complexity result for the latter problem also applies to the multicoloring version as well. Furthermore, in Section 5.3 we show that although minimum sum edge coloring is polynomial-time solvable for trees [GK00, Sal03], the multicoloring version is NP-hard, even if every demand is 1 or 2. On the other hand, we also show that the problem is polynomial time solvable for trees if every demand is the same. This is a consequence of the following scaling property of minimum sum edge multicoloring in trees (proved in Section 5.4.2): if the demand of every edge is multiplied by the same integer q, then the sum of the optimum solution increases by a factor of q.

In Section 5.4 we give a polynomial time approximation scheme (PTAS) for minimum sum edge multicoloring of trees. Before that, the 2-approximation algorithm of [BNHK$^+$00] for arbitrary graphs was the only known approximation result for minimum sum edge multicoloring.

The material of Section 5.2 is taken from [Mar02]; Section 5.3 and Section 5.4 appeared as [Mar04g, Mar04h].

5.1 Number of preemptions

The main result of this section is the following. Consider an instance of minimum sum multicoloring on a path. Let p be the maximum demand in the graph, $p = \max_{v \in V} x(v)$. We show that there is a minimum sum multicoloring with at most $O(\log p)$ preemptions at every vertex. Such a structural result might be useful when designing algorithms for the problem.

Besides possible algorithmic applications, there is another reason for trying to bound the number of preemptions required by the optimum solutions. If $x(v)$ appears in binary form in the input, then p can be exponentially large. Now it is not at all trivial that the decision version of the problem (is there a coloring with sum at most C?) is in NP, since it is not obvious how an optimum solution could be described in polynomial length. Our result shows that for paths the problem is indeed in NP: the optimum solution can be compactly represented by describing $O(\log p)$ intervals for each vertex. Moreover, we also show that if the graph is perfect and has n vertices, then there is always an optimum solution with at most n^2 preemptions at every vertex, regardless of the size of p. Thus the optimum solution can be given as $n \cdot n^2$ intervals, hence the problem is in NP for perfect graphs. The proof uses polyhedral arguments. If the graph is bipartite, then simple observations show that there is an optimum solution with at most n preemptions.

Notation and basic observations are given in Section 5.1.1. Section 5.1.2 defines several operations, which will be used to transform colorings. Section 5.1.3 proves a key structural property of optimum colorings. In Section 5.1.4, this observation is used to bound the number of preemptions required in an optimum coloring. Section 5.1.5 investigates more general classes of graphs and bounds the number of preemptions required in bipartite and perfect graphs.

5.1.1 Preliminaries

We slightly extend the problem by allowing $x(v) = 0$. It is clear that this does not change the problem, but it will be useful for technical reasons. If $x(v) = 0$ then let $f_\Psi(v) = 0$ in every coloring Ψ. Note that by using this definition the trivial inequality $f_\Psi(v) \geq x(v)$ holds even if $x(v) = 0$.

We introduce some notational conventions. Let $f_\Psi(V') = \sum_{v \in V'} f_\Psi(v)$ for any $V' \subseteq V$ and coloring Ψ. Similarly, $x(V') = \sum_{v \in V'} x(v)$. If $x(v) = 1$, then we will write $\Psi(v) = a$ instead of $\Psi(v) = \{a\}$. We denote the sum of the optimal coloring of G with demand function x by $\mathrm{OPT}(G, x)$, or by $\mathrm{OPT}(G)$ if the function $x(v)$ is clear from the context. The set $[a, b]$ is the set of integers $\{a, a+1, \ldots, b\}$. A node v is *compact* in coloring Ψ if $f_\Psi(v) = x(v)$.

We define a secondary objective function, which is to maximize $\sum_{v \in V} f_\Psi^2(v)$. Call a coloring Ψ *square-optimal* if its sum is minimum, and among such colorings, $\sum_{v \in V} f_\Psi^2(v)$ is maximum. Note that in an optimum solution the secondary objective function can be bounded by $n \cdot OPT^2(G)$.

In this section we will always assume that G is a path. Let 1, 2 , ..., n be the nodes of G along the path in left to right order. For convenience, we set $x(0) = x(n+1) = f_\Psi(0) = f_\Psi(n+1) = 0$ in every coloring Ψ. Node v is a *local maximum* in Ψ if $f_\Psi(v) > f_\Psi(v-1)$ and $f_\Psi(v) > f_\Psi(v+1)$, and it is a *local minimum* if $f_\Psi(v) < f_\Psi(v-1)$ and $f_\Psi(v) < f_\Psi(v+1)$. Node v is an *upward step* in Ψ if $f_\Psi(v-1) < f_\Psi(v) < f_\Psi(v+1)$ and it is a *downward step* if $f_\Psi(v-1) > f_\Psi(v) > f_\Psi(v+1)$.

In [HK02] it is proved that every optimum coloring uses at most $O(p \cdot \chi(G) \cdot \log n)$ (where p is the maximum demand) colors, for bipartite graphs, this is $O(p \cdot \log n)$. It is easy to see that in an optimum coloring of a path $f_\Psi(i) \leq 3p$ holds for every i: the neighbors $\Psi(i-1)$ and $\Psi(i+1)$ forbid the use of at most $2p$ colors and $x(i) \leq p$.

Proposition 5.1.1. *If Ψ is an optimum coloring of a path, then $f_\Psi(i) \leq 3p$.* ■

A coloring Ψ is a *conflicting coloring* if $|\Psi(v)| = x(v)$ is satisfied for every node v, but $\Psi(u) \cap \Psi(v) = \emptyset$ does not hold for some neighbors u and v. A conflicting coloring contains one or more *conflicts*, where a conflict is a triple (u, v, c) such that u and v are neighboring nodes and $c \in \Psi(u)$, $c \in \Psi(v)$. In a path, a conflict is of the form $(v, v+1, c)$.

5.1.2 Operations

In this section we define several operations that transform a coloring to another one. These transformations will be used in Section 5.1.3 and 5.1.4 to show certain properties of optimum colorings. Before presenting the transformations, let us have a note about the figures. In the figures to follow, the horizontal axis corresponds to the path, and the vertical axis corresponds to the set of colors. The black parts of the columns show which colors are assigned to the given node. For example, on Figure 5.1, node v has two intervals of consecutive colors between color a and b.

The *parity shift operation from u to v in the range a to b* transforms a coloring Ψ to a coloring Φ as follows. Let $\Phi(w) = \Psi(w)$ for all $w < u$ and $w > v$ (see Figure 5.1). Otherwise let $x'(w) = |\Psi(w) \cap [a, b]|$ and

$$\Phi(w) = \begin{cases} (\Psi(w) \cap [1, a-1]) \cup (\Psi(w) \cap [b+1, \infty]) \cup [a, a + x'(w) - 1] & \text{if } w \equiv u \pmod 2 \\ (\Psi(w) \cap [1, a-1]) \cup (\Psi(w) \cap [b+1, \infty]) \cup [b - x'(w) + 1, b] & \text{if } w \not\equiv u \pmod 2 \end{cases}$$

Clearly, $|\Phi(w)| = x(w)$, but Φ is not, in general, a proper coloring, there might be conflicts between u and $u-1$, and between v and $v+1$. However, if $f_\Psi(u-1) < a < b$ and $f_\Psi(v+1) < a < b$, then Φ is a proper coloring. The operation can increase the finish time for only those vertices w that has parity different from u. But if the finish time is greater than b for all such w between u and v, then their finish time is not increased, hence the sum of Φ is not greater than the sum of Ψ.

The second operation is the *change of colors a and b from u to v*, resulting in a new coloring Φ. It does not change the colors assigned to nodes $w < u$ and $w > v$, and for every node $u \leq w \leq v$ and

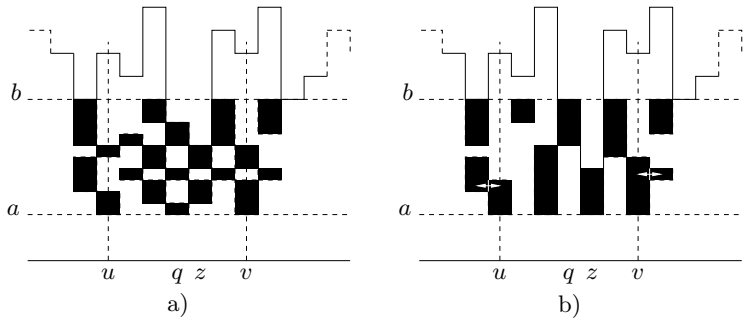


Figure 5.1: Parity shift from u to v in the range a to b. a) Before the operation. b) After the operation; there are conflicts between $u-1$ and u, and between v and $v+1$. The finish time of q increased while the finish time of z decreased.

every color $c \neq a, b$, we have $c \in \Phi(w)$ if and only if $c \in \Psi(w)$. Furthermore, for such a node w, color a (resp. b) $\in \Phi(w)$ if and only if b (resp. a) $\in \Psi(w)$ (see Figure 5.2). There are four possible conflicts that might arise in Φ: $(u-1, u, a)$, $(u-1, u, b)$, $(v, v+1, a)$, $(v, v+1, b)$. At most two of them can appear simultaneously in Φ: if the conflicts $(u-1, u, a)$ and $(u-1, u, b)$ both appear in Φ, then they were already present in Ψ, and similarly for $(v, v+1, a), (v, v+1, b)$. Thus we will distinguish between color changes with conflict on the left/on the right/on both sides. Note that if any of the following conditions hold, then there is no conflict on the left (assume $a < b$):

- $f_\Psi(u-1) < a$
- $a \in \Psi(u)$ and $b \in \Psi(u)$
- $a \notin \Psi(u)$ and $b \notin \Psi(u)$
- $a \in \Psi(u-1)$ and $b \in \Psi(u-1)$
- $a \notin \Psi(u-1)$ and $b \notin \Psi(u-1)$
- $a \notin \Psi(u-1)$ and $a \notin \Psi(u)$
- $b \notin \Psi(u-1)$ and $b \notin \Psi(u)$

If u is a local maximum and there is a conflict $(u-1, u, a)$ in Φ after the color change, then we can resolve this conflict by setting $\Phi'(u) = \Phi(u) - \{a\} \cup \{f_\Phi(u) + 1\}$. We call this operation a "fix," and use the term color change with a fix on the left/on the right/on both sides. A fix increases the sum of the coloring by one. The left (resp. right) fix can be applied only if u (resp. v) is a local maximum.

Assume $a < b$. If $f_\Psi(w) \geq b$ for all $u \leq w \leq v$, then the color change does not increase the finish time of any of the nodes. Furthermore, if it also holds that $f_\Psi(w) = b$ for some $u \leq w \leq v$ and $a \notin \Psi(w)$, then $f_\Phi(w)$ is strictly smaller than $f_\Psi(w)$. In this case, if only one fix required (i.e., either on the left or in the right, but not on both sides), then the finish time is increased only for one node. For example, if only a left fix is needed, then the operation increases the finish time of only u. Therefore if the finish time of w was decreased by more than 1, then the total finish time is decreased. On the other hand, if the finish time of w was decreased only by 1, then the total sum is not decreased, but, as an easy calculation shows, the secondary objective function is strictly greater in Φ' (this follows from $f_\Psi(u) \geq f_\Psi(w)$).

A *shift down of i from u to v* is a series of color changes from u to v that makes the node i compact, but possibly creates some conflicts between $u-1$ and u or between v and $v+1$. The color changes are done as follows. Let $c < f_\Psi(i)$ be the smallest color not in $\Psi(i)$ and do a color change of c and $f_\Psi(i)$ from

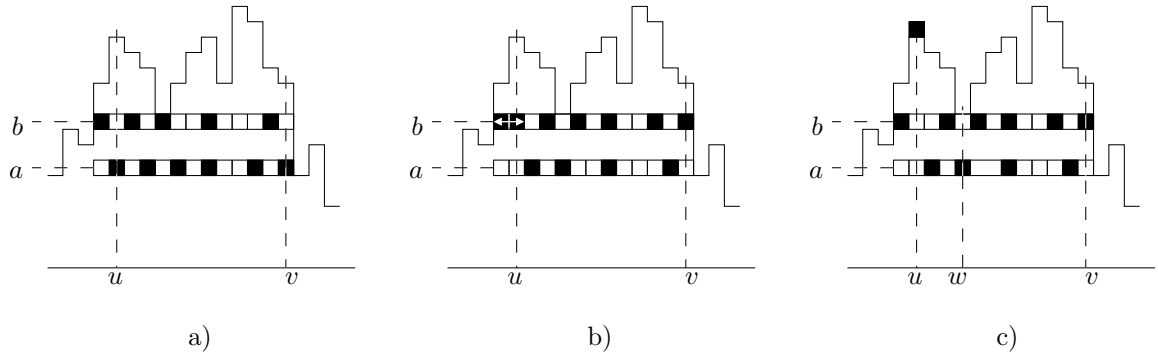


Figure 5.2: Change of colors a and b from u to v. a) The original coloring. b) After the change, the finish time node w decreased but there is a conflict on the left: color b appears at $u-1$ and u. There is no conflict on the right, since the finish time of $v+1$ is less than a. c) The fix at node u resolves the conflict, but increases the finish time of u by 1.

a to b. Repeat this m times, until $\Psi(i)$ is compact. Since every color change decreases the finish time of i, hence $m \leq f_\Psi(i) - x(i)$. Moreover, the color c chosen for the color change is strictly increasing, but always less than $x(i)$, thus we also have the bound $m \leq x(i)$.

The operation decreases the finish time of i by $f_\Psi(i) - x(i)$, but the coloring might contain conflicts. However, if u and v are local maximums, and after each color change we perform zero, one or two fixes, as required, then the resulting coloring will be proper. Therefore the total number of fixes is at most $2m$.

Let $\Psi(i) \cap [t+1, \infty] = c'_1 < c'_2 < \cdots < c'_k$ be the colors above t assigned to i. The *partial shift down of i above t from u to v* is a series of color changes where we chose (while it is possible) a color c such that $t < c < f_\Psi(i)$ and $c \notin \Psi(i)$, and do a color change of c and $f_\Psi(i)$ from a to b. After this operation, the set assigned to i is $(\Psi(i) \cap [1, t]) \cup [t+1, t+k]$, which decreases the finish time of i by $f_\Psi(i) - t - k$. It is easy to see that this requires no more than $\min\{k, f_\Psi(i) - t - k\}$ color changes.

The multicoloring instance can be *restricted from u to v*, that is, we can consider only the subgraph from node u to node v. A coloring Ψ can be similarly restricted, the restricted coloring is a proper coloring of the restricted instance, though the restriction of an optimum coloring is not necessarily an optimum coloring of the restricted instance. If Ψ_1 is a coloring of the restriction of the instance from u to v, and Ψ_2 is a coloring of the restriction from $v+1$ to w, then we can *concatenate* the two colorings to a coloring Ψ of the restriction from u to w. Coloring Ψ is not necessarily a proper coloring, there might be conflicts between v and $v+1$. However, if we have a coloring from u to v and another coloring from v to w, and they assign the same set to v, then their concatenation is a proper coloring from u to w.

Given a coloring Ψ, we define the instance *truncated at t* by the new demand function $x'(i) = |\Psi(i) \cap [t+1, \infty]|$. Let $c \in \Psi'(i)$ if and only if $c + t \in \Psi(i)$. Then Ψ' is clearly a proper coloring of the truncated instance. Furthermore, if Ψ is an optimum coloring, then Ψ' is an optimum coloring of the restricted instance: it is easy to see that a better solution to the restricted instance would imply a better solution to the original problem.

Using the operations defined above, we introduce two more complex transformations. In the following, what will be important is how these operations change the set of colors assigned to node n and how they change the sum of the coloring, not their exact definitions. Therefore we state the existence of these two transformations in two lemmas, and present the actual operations only in the proofs. If x is an arbitrary expression, then we denote $\max\{x, 0\}$ by $(x)^+$.

The operation described in the following lemma makes the coloring of node n compact:

Lemma 5.1.2. *For every coloring Ψ there is a coloring Φ such that $\Phi(n) = [1, x(n)]$ and $f_\Phi(V) \leq f_\Psi(V) + (f_\Psi(n) - f_\Psi(n-1))^+$.*

Proof. We consider three cases: (1) $f_\Psi(n) < f_\Psi(n-1)$, (2) $f_\Psi(n-2) < f_\Psi(n-1) < f_\Psi(n)$, (3) $f_\Psi(n) > f_\Psi(n-1)$ and $f_\Psi(n-1) < f_\Psi(n-2)$.

In the first case, let u be the first local maximum to the left of n. It exists because $f_\Psi(0) = x(0) = 0$. We do a shift down of n from u to n. This consists of a series of at most $f_\Psi(n) - x(n)$ color changes, each of them requires at most one fix, since there is no conflict on the right. The finish time of n is decreased by $f_\Psi(n) - x(n)$, thus the total sum is not increased.

In the second case, let $s = |\Psi(n-1) \cap [1, x(n)]| \leq f_\Psi(n) - x(n)$. The new coloring Φ is the same as Ψ except $\Phi(n) = [1, x(n)]$ and $\Phi(n-1) = (\Psi(n-1) \cap [x(n)+1, \infty]) \cup [f_\Psi(n)+1, f_\Psi(n)+s]$. It is easily verified that this is a proper coloring. The finish time of n is decreased by $f_\Psi(n) - x(n)$ and the finish time of $n-1$ is increased to $f_\Psi(n) + s$, that is, by at most $f_\Psi(n) + f_\Psi(n) - x(n) - f_\Psi(n-1)$. Thus the total increase is at most $f_\Psi(n) - f_\Psi(n-1)$.

In the third case, we let u be the first local maximum to the left of $n-1$ and do a shift down of n from u to n. This requires at most $f_\Psi(n) - x(n)$ left fixes, and increases the finish time of $n-1$ to $f_\Psi(n)$, that is, by at most $f_\Psi(n) - f_\Psi(n-1)$. Since the finish time of n is decreased by $f_\Psi(n) - x(n)$, the total increase of the sum is at most $f_\Psi(n) - f_\Psi(n-1)$, what we had to prove. ■

The second operation also makes node n compact. However, here we assume that the color set $[1, \alpha]$ is already assigned to n.

Lemma 5.1.3. *Assume that Ψ is a coloring and for some α, we have $[1, \alpha] \subseteq \Psi(n)$ and $|\Psi(n) \cap [\alpha+1, \infty]| = \beta$. Then there is a coloring Φ such that $\Phi(n) = [1, x(n)]$ and $f_\Phi(V) \leq f_\Psi(V) + \beta + (f_\Psi(n) - f_\Psi(n-1))^+ + x(n) - f_\Psi(n)$.*

Proof. We consider again the previous three cases: (1) $f_\Psi(n) < f_\Psi(n-1)$, (2) $f_\Psi(n-2) < f_\Psi(n-1) < f_\Psi(n)$, (3) $f_\Psi(n) > f_\Psi(n-1)$ and $f_\Psi(n-1) < f_\Psi(n-2)$.

In the first case, let u be the first local maximum to the left of n. We do a partial shift down of n from u to n above α. This consists of a series of at most β color changes, each of them requires at most one fix, since there cannot be conflict on the right. The finish time of n is decreased by $f_\Psi(n) - x(n)$, thus the total increase of the sum is at most $\beta + x(n) - f_\Psi(n)$.

In the second case, let $s = |\Psi(n-1) \cap [\alpha+1, \alpha+\beta]| \leq \beta$. The new coloring Φ is the same as Ψ except that $\Phi(n) = [1, x(n)]$ and $\Phi(n-1) = (\Psi(n-1) \cap [x(n)+1, \infty]) \cup [f_\Psi(n)+1, f_\Psi(n)+s]$. It is easily verified that this is a proper coloring (note that $\alpha + \beta = x(n)$ holds). The finish time of n is decreased by $f_\Psi(n) - x(n)$ and the finish time of $n-1$ is increased to $f_\Psi(n)+s$, that is, by at most $\beta + f_\Psi(n) - f_\Psi(n-1)$, thus we are ready with this case.

In the third case, we let u be the first local maximum to the left of $n-1$ and do a partial shift down of n from u to n above α. This requires at most β left fixes, and increases the finish time of $n-1$ by at most $f_\Psi(n) - f_\Psi(n-1)$. Since the finish time of n is decreased by $f_\Psi(n) - x(n)$, the total increase of the sum is at most $\beta + (f_\Psi(n) - f_\Psi(n-1)) + x(n) - f_\Psi(n)$, what we had to prove. ■

We have seen in Prop. 5.1.1 that the finish time of a node can be bounded by a constant multiple of the maximum demand. The following lemma shows that if i is not a local maximum then its finish time can be bounded in terms of its own demand.

Lemma 5.1.4. *If Ψ is an optimum solution and i is not a local maximum, then $f_\Psi(i) \leq 4x(i)$.*

Proof. Assume first that i is a local minimum, let u (resp. v) be the first local maximum to the left (resp. right) of i. If $\Psi(i)$ is not compact, then we perform a shift down of i from u to v. This requires at most $x(i)$ color changes. Therefore the fixes increase the sum by at most $2x(i)$. However, the finish time of i is decreased by $f_\Psi(i) - x(i)$. Assuming that Ψ is an optimum coloring, it follows that $f_\Psi(i) \leq 3x(i)$.

Now assume without loss of generality that i is an upward step. Denote by v the first local maximum to the right of i.

First we first modify the coloring Ψ to achieve that $f_\Psi(i) - f_\Psi(i-1) \leq x(i)$. We do a partial shift down of i above $f_\Psi(i-1)$ from i to v. This series of color changes does not require any left fixes. Since every color change decreases the finish time of i by at least one, this shift operation does not increase the total sum. After this operation, $[f_{\Psi'}(i-1)+1, f_{\Psi'}(i)] \subseteq \Psi'(i)$ in the resulting coloring Ψ', which obviously implies $f'_\Psi(i) - f'_\Psi(i-1) \leq x(i)$. Thus in the following we can assume that $f_\Psi(i) - f_\Psi(i-1) \leq x(i)$.

Now consider the restriction of the instance from 1 to i. We can apply Lemma 5.1.3 to node i with $\alpha = 0$ and $\beta = x(i)$. Therefore there exists a coloring Φ_1 with $\Phi_1(i) = [1, x(i)]$ and $f_{\Phi_1}([1,i]) \leq f_\Psi([1,i]) + \beta + (f_\Psi(i) - f_\Psi(i-1))^+ + x(i) - f_\Psi(i) \leq f_\Psi([1,i-1]) + 3x(i)$.

Next we restrict the instance and the coloring Ψ from i to n, and do a shift down of i from i to v. This requires at most $x(i)$ fixes on the right and no fixes on the left. The finish time of i is decreased by $f_\Psi(i) - x(i)$, thus for the resulting coloring Φ_2, we have $f_{\Phi_2}([i,n]) \leq f_\Psi([i,n]) + x(i) + x(i) - f_\Psi(i)$. We can concatenate Φ_1 and Φ_2 to a coloring Φ of the original instance since they assign the same set to i. We can bound the sum of Φ by

$$\begin{aligned} f_\Phi([1,n]) &= f_{\Phi_1}([1,i]) + f_{\Phi_2}([i,n]) - f_{\Phi_2}(i) \leq \\ &\leq (f_\Psi([1,i-1]) + 3x(i)) + (f_\Psi([i,n]) + 2x(i) - f_\Psi(i)) - x(i) = \\ &= f_\Psi([1,n]) + 4x(i) - f_\Psi(i) \end{aligned}$$

By assumption, Ψ is an optimum coloring, thus the sum of Φ is not smaller than the sum of Ψ. By the inequalities shown above, this implies $4x(i) - f_\Psi(i) \geq 0$. ■

5.1.3 Bounding the reduced sequence

Let $\ell_0^{(i)} = i$ be the first element of the *left sequence* of the node i in a coloring Ψ, and for $k \geq 0$ let $\ell_{k+1}^{(i)}$ be the largest $j < \ell_k^{(i)}$ such that $f_\Psi(j) \leq f_\Psi(\ell_k^{(i)})$ (see Figure 5.3). The last element of the sequence is $\ell_{L(i)}^{(i)} = 0$. The right sequence is similarly defined with $r_0^{(i)} = i$ and $r_{k+1}^{(i)}$ is the smallest $j > r_k^{(i)}$ such that $f_\Psi(j) \leq f_\Psi(r_k^{(i)})$, the last element of the sequence is $r_{R(i)}^{(i)} = n+1$. The *reduced left (right) sequence* is the sequence truncated at the first compact node, its length is $R'(i)$ (resp. $L'(i)$). The first compact node in the sequence is included in the reduced sequence only if its finish time is greater than zero. In particular, if $f_\Psi(i) = x(i) = 0$, then $L'(i) = R'(i) = 0$.

In this section, our aim is to bound the length of the reduced left and right sequence of every node: in Section 5.1.4, Lemma 5.1.13 shows the connection between the length of these sequences and the number of preemptions required in an optimum coloring. The following series of lemmas will prove that $L'(i)$ is at most logarithmic in p. The bound for $R'(i)$ follows by symmetry.

Lemma 5.1.5. *If Ψ is a square-optimal solution, and nodes u and v are local minimums such that at least one of them is not compact and for all $u < w < v$, we have $f_\Psi(w) \geq \max\{f_\Psi(u), f_\Psi(v)\}$, then u and v have opposite parity.*

Proof. Let q be the first local maximum to the left of u and z the first local maximum to the right of v (see Figure 5.4). We do a shift down of u from q to u, with doing the required fixes on the left (but not on the right). This requires at most $f_\Psi(u) - x(u)$ fixes and decreases the finish time of u by that amount, thus it does not increase the total sum. Similarly, we do a shift down of v from v to z (with doing fixes on the right), this does not increase the total sum either. There might be conflicts between u and $u+1$ or between v and $v-1$, we resolve this by applying a parity shift from u to v in the range from 1 to $\max\{f_\Psi(u), f_\Psi(v)\}$. This does not change u and v, since they were already compact after the shift down, thus this results in a proper coloring. The parity shift does not increase the finish time of any of the nodes $u < w < v$, since, by assumption, they have finish time not smaller than $\max\{f_\Psi(u), f_\Psi(v)\}$.

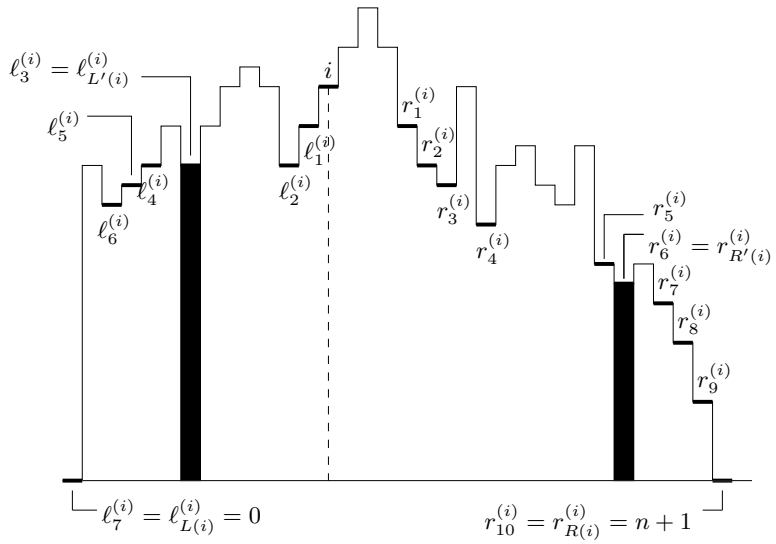


Figure 5.3: The left and right sequence of the node i.

We have shown that the operations described above do not increase the total sum. They cannot decrease it, that would contradict the optimality of Ψ. Assume without loss of generality that u was not compact, then the finish time of u was decreased by $f_\Psi(u) - x(u) > 0$, and the finish time of q was increased by exactly the same amount. However, this would increase the secondary objective function, which is a contradiction, since Ψ is square-optimal. ■

Corollary 5.1.6. *If Ψ is a square-optimal solution, and for some i and $1 \leq j < L'(i) - 1$, the nodes $\ell_j^{(i)}$ and $\ell_{j+1}^{(i)}$ are both local minimums, then they have opposite parity.*

Proof. By the definition of the left sequence, for every node $\ell_{j+1}^{(i)} < v < \ell_j^{(i)}$, we have $f_\Psi(v) > f_\Psi(\ell_j^{(i)}) \geq f_\Psi(\ell_{j+1}^{(i)})$. From the definition of the reduced left sequence, the nodes $\ell_j^{(i)}$ and $\ell_{j+1}^{(i)}$ are not compact, thus the corollary follows from Lemma 5.1.5. ■

Lemma 5.1.7. *If Ψ is a square-optimal coloring, then for every i and $3 \leq j < L'(i) - 3$, at least one of $\ell_{j+1}^{(i)}, \ell_{j+2}^{(i)}, \ell_{j+3}^{(i)}$ and at least one of $\ell_{j-1}^{(i)}$, $\ell_{j-2}^{(i)}, \ell_{j-3}^{(i)}$ has the same parity as $\ell_j^{(i)}$.*

Proof. Assume that, on the contrary, they all have parity different from $\ell_j^{(i)}$. Then $\ell_{j+1}^{(i)}$ is a local minimum, otherwise $\ell_{j+2}^{(i)}$ would have parity different from $\ell_{j+1}^{(i)}$, and one of them would have the same parity as $\ell_j^{(i)}$. Similarly, $\ell_{j+2}^{(i)}, \ell_{j-2}^{(i)}, \ell_{j-3}^{(i)}$ are also local minimums. By Corollary 5.1.6, $\ell_{j+1}^{(i)}$ and $\ell_{j+2}^{(i)}$ have different parity, therefore one of them has the same parity as $\ell_j^{(i)}$. By a similar argument, $\ell_{j-2}^{(i)}$ and $\ell_{j-3}^{(i)}$ are both local minimums, thus we are ready. ■

Lemma 5.1.8. *If Ψ is an optimum coloring and for some i and $3 \leq j < L'(i) - 3$, $\ell_j^{(i)}$ is an upward step, then $f_\Psi(\ell_{j-3}^{(i)}) \geq \frac{9}{8} f_\Psi(\ell_{j+4}^{(i)})$.*

Proof. By Lemma 5.1.7, one of $\ell_{j+1}^{(i)}, \ell_{j+2}^{(i)}, \ell_{j+3}^{(i)}$ has the same parity as $\ell_j^{(i)}$, call it v_1. Similarly, call v_3 the node $\ell_{j-1}^{(i)}, \ell_{j-2}^{(i)}$ or $\ell_{j-3}^{(i)}$ which also has this parity. Let $v_2 = \ell_j^{(i)}$, and denote by v_0 the next element in the left sequence after v_1. Call q the first local maximum to the right of v_3. Let $d_i = f_\Psi(v_i) - f_\Psi(v_{i-1})$ for

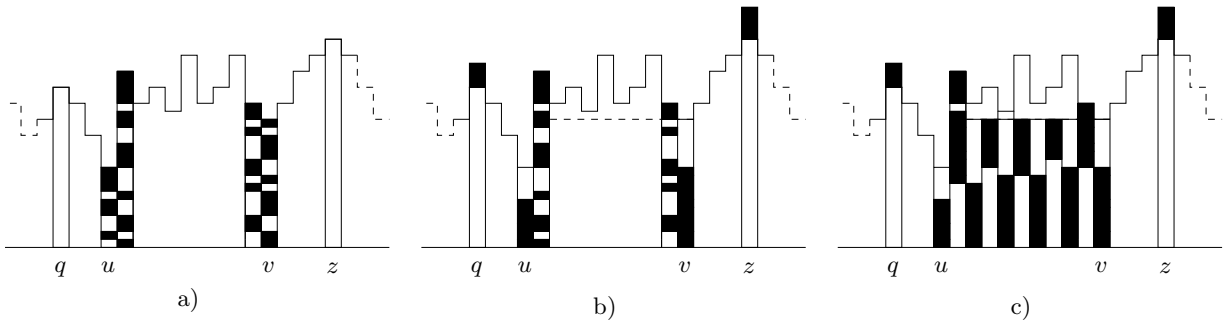


Figure 5.4: Transformation of the coloring in the proof of Lemma 5.1.5. a) Nodes u and v are local minimums, q and z are local maximums. b) After the shift down, u and v are compact, the fixes increased the finish time of q and z, and there might be conflicts between u and $u+1$ or between v and $v-1$. c) The parity shift resolves the conflicts without increasing the finish times.

$1 \leq i \leq 3$, Note that $d_1 + d_2 + d_3 = f_\Psi(v_3) - f_\Psi(v_0) \leq f_\Psi(\ell^{(i)}_{j-3}) - f_\Psi(\ell^{(i)}_{j+4})$. We will construct a coloring Φ and show that if it has sum not smaller than Ψ, then $f_\Psi(\ell^{(i)}_{j-3}) \geq \frac{9}{8} f_\Psi(\ell^{(i)}_{j+4})$, which proves the lemma, since Ψ is an optimum solution.

Before going into the details, we will give some intuition for the proof. We will make the three nodes v_1, v_2, v_3 compact. This decreases the finish time on these vertices, but it possibly increases the sum somewhere else. However, it turns out that the decrease at v_1 pays for any possible increase to the left of v_1, and the decrease at v_3 pays for the increase to the right of v_3. The total increase between v_1 and v_3 is roughly $d_1 + d_2 + d_3$, thus the optimality of Ψ implies that the decrease of $f_\Psi(v_2) - x(v_2)$ at v_2 cannot pay for this. If $d_1 + d_2 + d_3$ is "large," then this means that the finish times in the left sequence decrease rapidly, what we have to show. Therefore $d_1 + d_2 + d_3$ is small, and $f_\Psi(v_2) - x(v_2)$ is even smaller. Since $f_\Psi(v_2 - 1) < f_\Psi(v_2)$, thus $x(v_2 - 1) \leq f_\Psi(v_2) - x(v_2)$ is small. By Lemma 5.1.4, this implies that $f_\Psi(v_2 - 1)$ is small, hence the left sequence decreases rapidly.

First consider the restriction of Ψ from 1 to v_1. By Lemma 5.1.2, there is a coloring Ψ_1 with $\Psi_1(v_1) = [1, x(v_1)]$ and $f_{\Psi_1}([1, v_1]) \leq f_\Psi([1, v_1]) + (f_\Psi(v_1) - f_\Psi(v_1 - 1))^+ \leq f_\Psi([1, v_1]) + d_1$. Concatenate Ψ_1 with the restriction of Ψ from $v_1 + 1$ to v_2. The resulting coloring might have conflicts between v_1 and $v_1 + 1$, but we can solve this, without increasing the finish time of any of the nodes, by performing a parity shift from v_1 to v_2 in the range of 1 to $f_\Psi(v_1)$. Call Ψ_2 the resulting proper coloring of $[1, v_2]$. Clearly $f_{\Psi_2}([1, v_2]) \leq f_\Psi([1, v_2]) + d_1$.

Because of the parity shift (v_1 and v_2 have the same parity), $[1, \alpha_2] \subseteq \Psi_2(v_2)$ for some α_2 and $\Psi_2(v_2)$ does not contain any other colors up to $f_\Psi(v_1)$. Thus $\Psi_2(v_2)$ contains $\beta_2 \leq f_\Psi(v_2) - f_\Psi(v_1) = d_2$ colors above α_2. Applying Lemma 5.1.3 on v_2 with α_2, we get a coloring Ψ'_2 with $\Psi'_2(v_2) = [1, x(v_2)]$ and $f_{\Psi'_2}([1, v_2]) \leq f_{\Psi_2}([1, v_2]) + \beta_2 + (f_\Psi(v_2) - f_\Psi(v_2 - 1))^+ + x(v_2) - f_\Psi(v_2) \leq f_{\Psi_2}([1, v_2]) + 2d_2 + x(v_2) - f_\Psi(v_2)$.

Concatenate Ψ'_2 with the restriction of Ψ from $v_2 + 1$ to v_3. To resolve possible conflicts between v_2 and $v_2 + 1$, we do a parity shift from v_2 to v_3 in the range 1 to $f_\Psi(v_2)$ to obtain a proper coloring Ψ_3. Since v_2 and v_3 have the same parity, after the parity shift we have that $[1, \alpha_3] \subseteq \Psi_3(v_3)$ for some α_3 and $\Psi_3(v_3)$ does not contain any other colors up to $f_\Psi(v_2)$. Thus $\Psi_3(v_3)$ contains $\beta_3 \leq f_\Psi(v_3) - f_\Psi(v_2) = d_3$ colors above α_3. We use Lemma 5.1.3 once more to obtain a coloring Ψ'_3 with $\Psi'_3(v_3) = [1, x(v_3)]$ and $f_{\Psi'_3}([1, n]) \leq f_{\Psi_3}([1, v_3]) + \beta_3 + (f_\Psi(v_3) - f_\Psi(v_3 - 1))^+ + x(v_3) - f_\Psi(v_3) \leq f_{\Psi_3}([1, v_3]) + 2d_3 + x(v_3) - f_\Psi(v_3)$.

Finally, consider the restriction of Ψ from v_3 to n, and do a shift down of v_3 from v_3 to q which results in a coloring Ψ_4 with $\Psi_4(v_3) = [1, x(v_3)]$. This operation requires at most $f_\Psi(v_3) - x(v_3)$ fixes on the right (at q), thus $f_{\Psi_4}([v_3 + 1, n]) \leq f_\Psi([v_3 + 1, n]) + f_\Psi(v_3) - x(v_3)$. Concatenate Ψ'_3 and Ψ_4 to obtain a coloring Φ (note that $\Psi'_3(v_3) = \Psi_4(v_3) = [1, x(v_3)]$, thus Φ is a proper coloring).

Now we bound the sum of Φ using the inequalities obtained above:

$$\begin{aligned}
f_\Phi([1,n]) &= f_{\Psi'_3}([1,v_3]) + f_{\Psi_4}([v_3+1,n]) \leq \\
&\leq (f_{\Psi_3}([1,v_3]) + 2d_3 + x(v_3) - f_\Psi(v_3)) + (f_\Psi([v_3+1,n]) + f_\Psi(v_3) - x(v_3)) = \\
&= f_{\Psi'_2}([1,v_2]) + f_\Psi([v_2+1,v_3]) + 2d_3 + f_\Psi([v_3+1,n]) \leq \\
&\leq f_{\Psi_2}([1,v_2]) + 2d_2 + x(v_2) - f_\Psi(v_2) + 2d_3 + f_\Psi([v_2+1,n]) \leq \\
&\leq f_\Psi([1,v_2]) + d_1 + 2d_2 + x(v_2) - f_\Psi(v_2) + 2d_3 + f_\Psi([v_2+1,n]) \leq \\
&\leq f_\Psi([1,n] + 2(d_1+d_2+d_3) + x(v_2) - f_\Psi(v_2)
\end{aligned}$$

By assumption, Ψ is an optimum coloring, thus $2(d_1+d_2+d_3) \geq f_\Psi(v_2) - x(v_2)$. Since v_2 is an upward step, $f_\Psi(v_2) - x(v_2) \geq x(v_2-1)$, and by Lemma 5.1.4, $f_\Psi(v_2-1) \leq 4x(v_2-1)$, hence $f_\Psi(v_2) - x(v_2) \geq \frac{1}{4} f_\Psi(v_2-1) \geq \frac{1}{4} f_\Psi(v_1)$. Therefore we have $f_\Psi(v_3) - f_\Psi(v_1) \geq d_1+d_2+d_3 \geq \frac{1}{2}(f_\Psi(v_2) - x(v_2)) \geq \frac{1}{8} f_\Psi(v_1)$, which proves the lemma. ■

Corollary 5.1.9. *If Ψ is a square-optimal coloring, then for every i, the reduced left sequence of i, $\ell^{(i)}_1, \ldots, \ell^{(i)}_{L'(i)}$ contains at most $O(\log p)$ upward steps.*

Proof. Let m be the number of left steps in the reduced left sequence of i and denote by $\ell^{(i)}_{s_1}, \ell^{(i)}_{s_2}, \ldots, \ell^{(i)}_{s_m}$ the subsequence that contains all the upward steps of the reduced left sequence. We show that $f_\Psi(\ell^{(i)}_{s_j}) \geq \frac{9}{8} f_\Psi(\ell^{(i)}_{s_{j+7}})$ holds for every j. By applying Lemma 5.1.8 for $\ell^{(i)}_{s_{j+3}}$, we get that $f_\Psi(\ell^{(i)}_{s_{j+3}-3}) \geq \frac{9}{8} f_\Psi(\ell^{(i)}_{s_{j+3}+4})$. Clearly, $s_{j+3} - 3 \geq s_j$ and $s_{j+3} + 4 \leq s_j + 7$, thus the proposed inequality indeed holds. Now, by Proposition 5.1.1, we have $3p \geq f_\Psi(i) \geq f_\Psi(\ell^{(i)}_{s_1}) \geq (\frac{9}{8})^{\lfloor m/7 \rfloor}$, and the corollary follows. ■

Corollary 5.1.9 bounds the number of steps in the reduced left sequence, and by symmetry, a similar bound follows for the reduced right sequence. It remains to give a bound on the number of local minimums. Using the following lemma, we can bound the number of local minimums in the reduced left sequence using the bound on the number of steps in the reduced *right* sequence.

Lemma 5.1.10. *If Ψ is a square-optimal coloring, and for some i and $1 \leq j < L'(i) - 1$, $v = \ell^{(i)}_j$ and $u = \ell^{(i)}_{j+1}$ are both local minimums, $f_\Psi(\ell^{(i)}_j) > f_\Psi(\ell^{(i)}_{j+1})$ and they have different parity, then there is a $1 \leq k \leq R'(i)$ such that $f_\Psi(\ell^{(i)}_j) > f_\Psi(r^{(i)}_k) \geq f_\Psi(\ell^{(i)}_{j+1})$ and $r^{(i)}_k$ is a downward step.*

Proof. Let $c < f_\Psi(v)$ be such that $c \notin \Psi(v)$ (note that v is not compact), and let z (resp. w) be the first local maximum to the left of v (resp. u), respectively, clearly $w < u < z < v$ (see Figure 5.5). Let $q = r^{(i)}_k$ be the first node in the right sequence of i having finish time strictly smaller than $f_\Psi(v)$, we will show that this k satisfies the requirements of the Lemma. If there is a $v < m < q$ that is compact (Figure 5.5a), then, by the minimality of k, $f_\Psi(m) \geq f_\Psi(v)$. Now we can do a color change of c and $f_\Psi(v)$ from z to m, this requires no right fix, since both c and $f_\Psi(y)$ are contained in $\Psi(m)$, thus it does not increase the sum. Therefore the color change either decreases the sum or increases the secondary objective function, which is a contradiction. This proves that $k \leq R'(i)$, we have to show that q is a downward step and $f_\Psi(q) \geq f_\Psi(u)$.

If q is a local minimum, then it has the same parity as u or as v. In either case, Lemma 5.1.5 gives a contradiction, by the way we chose q, the conditions hold. Therefore q is a downward step.

Now assume that $f_\Psi(q) < f_\Psi(u)$. If $f_\Psi(u) \notin \Psi(v)$, then the color change of $f_\Psi(u)$ and $f_\Psi(v)$ from z to $q-1$ does not require a right fix and decreases the finish time of v, a contradiction (Figure 5.5b). If $f_\Psi(u)$ is contained in both of $\Psi(u)$ and $\Psi(v)$, then by the fact that u and v have different parity, it follows that there is a $u < s < v-1$ such that $f_\Psi(u)$ is contained neither in $\Psi(s)$ nor $\Psi(s+1)$ (Figure 5.5c). Let $c' < f_\Psi(u)$ be such that $c' \notin \Psi(u)$ and do a color change of c' and $f_\Psi(u)$ from w to s, this decreases the finish time of u but does not require a fix on the right, a contradiction. Thus we proved that $f_\Psi(q) \geq f_\Psi(u)$ and we have seen that q is a downward step, the Lemma follows. ■

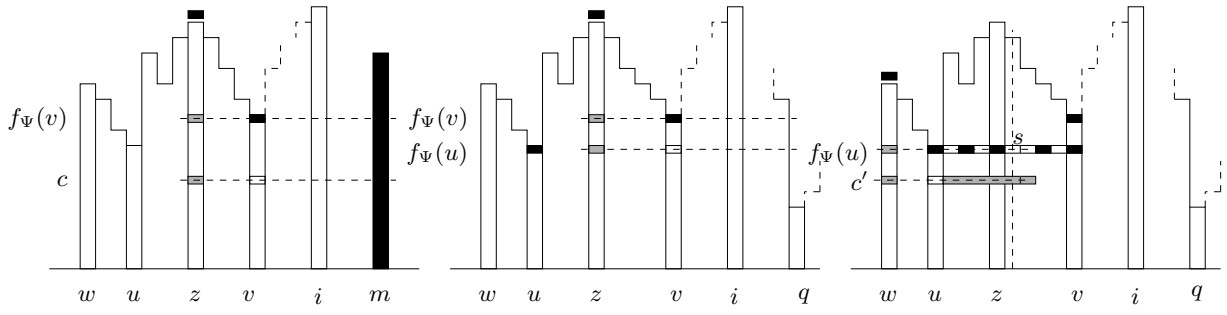


Figure 5.5: Lemma 5.1.10. a) If there is a compact $v < m < q$, then change the colors $f_\Psi(v)$ and c. b) If $f_\Psi(u) \notin \Psi(v)$, then change the colors $f_\Psi(u)$ and $f_\Psi(v)$. c) If $f_\Psi(u) \in \Psi(v)$, then change the colors $f_\Psi(u)$ and c'.

Corollary 5.1.11. *If Ψ is a square-optimal coloring, then for every i, the reduced left sequence of i, $\ell_j^{(i)}, \ldots, \ell_{L'(i)}^{(i)}$ contains at most $O(\log p)$ local minimums.*

Proof. A local minimum $\ell_j^{(i)}$ can be one of three types: (1) $\ell_{j-1}^{(i)}$ is a upward step, (2) $\ell_{j-1}^{(i)}$ is a local minimum and $f_\Psi(\ell_{j-1}^{(i)}) > f_\Psi(\ell_j^{(i)})$, (3) $f_\Psi(\ell_{j-1}^{(i)}) = f_\Psi(\ell_j^{(i)})$. By Corollary 5.1.9, there are $O(\log p)$ upward steps in the reduced sequence, thus there are at most $O(\log p)$ local minimums of type 1. By Corollary 5.1.6, if $\ell_j^{(i)}$ and $\ell_{j-1}^{(i)}$ are both local minimums, then they have different parity. Therefore by Lemma 5.1.10, for every node $\ell_j^{(i)}$ of type 2, there is a node v in the reduced right sequence of i which is a right step and $f_\Psi(\ell_j^{(i)}) \leq f_\Psi(v) < f_\Psi(\ell_{j-1}^{(i)})$ holds. Since there are $O(\log p)$ right steps in the reduced right sequence and clearly the inequality above cannot hold for two different nodes of type 2 with the same node v, this bounds the number of type 2 nodes by $O(\log p)$. Finally, note that there cannot be three local minimums in the reduced left sequence with the same finish time, two of them would have the same parity, and by Lemma 5.1.5, this gives a contradiction. Therefore there cannot be two nodes of type 3 after each other in the left sequence, thus the number of type 3 nodes can be bounded by the number of type 1 and type 2 nodes, which proves the corollary. ■

By combining Corollary 5.1.9 and Corollary 5.1.11, we have

Corollary 5.1.12. *If Ψ is a square-optimal coloring, then $L'(i)$ is at most $O(\log p)$.* ■

5.1.4 Optimum coloring

To obtain the main result of this section, it has to be shown that if a node has short left and right sequences, then the coloring can be rearranged in such a way that the color set assigned to the node has a small number of preemptions:

Lemma 5.1.13. *Given a coloring Ψ, there is a coloring Φ with $f_\Psi(i) = f_\Phi(i)$ for every i and $\Phi(i)$ has at most $L'(i) + R'(i)$ preemptions.*

Proof. We prove the lemma by induction on $k = \max(L'(i) + R'(i))$. If $L'(i) = R'(i) = 0$, then node i is compact ($x(i) = 0$), thus the statement is true. Now assume that it is true for k, we prove that it is also true for $k + 1$.

Let $b_0 = 0 < b_1 < b_2 < \cdots < b_{l-1} < b_l = n+1$ be the compact nodes. For each $0 \leq j < l$, we construct a coloring Φ_j of the restriction of the instance from b_j to b_{j+1}, such that for every $b_j \leq i \leq b_{j+1}$, $f_{\Phi_j}(i) = f_\Psi(i)$ and $\Phi_j(i)$ contains at most $L'(i) + R'(i)$ preemptions. Concatenating these colorings Φ_j to a coloring Φ clearly satisfies the properties required by the lemma. (The colorings can be concatenated without conflict since the b_j nodes are compact in all of them.)

Let m be the smallest non-zero finish time $f_\Psi(i)$ in $b_j \leq i \leq b_{j+1}$ and consider the restriction of the instance from b_j to b_{j+1}, truncated at m. The truncation of Ψ is a solution of the truncated instance. If we can prove that in the truncation of Ψ, the sum $L'(i) + R'(i)$ is strictly smaller than in Ψ, then, by the induction hypotheses, there is an optimum coloring Φ'_j of the truncated instance where $\Phi'_j(i)$ has less than $L'(i) + R'(i)$ preemptions. We do a parity shift of Ψ from b_j to b_{j+1} in the range 1 to m, and add the first m colors to Φ'_j to obtain a coloring Φ_j. This coloring Φ_j has the required properties, since $\Phi_j(i)$ has at most one more preemption than $\Phi'_j(i)$.

It is easy to see that the reduced left (resp. right) sequence of node i in the truncated instance is the prefix of the reduced left (resp. right) sequence in Ψ. Let $b_j \leq i' \leq b_{j+1}$ such that its finish time is m, and assume without loss generality that it is to the left of i, that is, $i' < i$. Clearly i' belongs to the left sequence of i, since there are no nodes with finish time smaller than m between i' and i. Furthermore, i' belongs to the reduced left sequence: there are no compact nodes between i and i', and because $m > 0$, i' belongs to the reduced sequence even if it is the first compact node in the left sequence. But in the truncated instance, i' has zero finish time, thus it cannot be in the reduced left sequence of i, which implies that $L'(i)$ is strictly smaller in the truncated instance, what we had to prove. ■

Combining Corollary 5.1.9 and Lemma 5.1.13, we get

Theorem 5.1.14. *For every instance of the preemptive sum multicoloring problem on paths, there is a solution Ψ such that for every node i, $\Psi(i)$ is the union of at most $O(\log p)$ intervals.* ■

Theorem 5.1.14 is best possible in the sense that there are instances where in every optimum solution there is a node with $\Theta(\log p)$ preemptions. Let $x(i) = 4^i$. We show that the only optimum solution is $\Psi(1) = [1, 4]$ and $\Psi(i+1) = [1, x(i) + x(i+1)] \setminus \Psi(i)$, that is, the first $x(i+1)$ colors not used by $\Psi(i)$. Clearly, $f_\Psi(i+1) = x(i+1) + x(i)$. If there is a solution Φ with sum smaller that Ψ, then $f_\Phi(i) < f_\Psi(i)$ for some i. In this case it follows that $f_\Phi(i-1) \geq x(i-1) + x(i)$, therefore $f_\Phi(i-1) + f_\Phi(i) \geq x(i-1) + x(i) + x(i) > f_\Psi(i) + f_\Psi(i-1)$. Summing this for all i where $f_\Phi(i) < f_\Psi(i)$, we get that the sum of Ψ is strictly smaller than the sum of Φ. It can be shown by induction that $\Psi(2i)$ has i preemptions, thus node n has $\frac{n}{2} = \Theta(\log p)$ preemptions.

Theorem 5.1.14 states that for every instance, there is a square-optimal coloring Ψ with the required property. There is a natural question that can be asked: is it true perhaps that *every* solution (or at least every square-optimal solution) has this property? However, the theorem cannot be strengthened this way: we present an example where a node in a square-optimal solution has $\Theta(p)$ preemptions.

Let $x(1) = 2C$, $x(2) = 10C$, $x(3) = 100C$, $x(4) = 11C$, $x(5) = 100C$, $x(6) = 10C$, $x(7) = 2C$, where C is a large constant. Consider the coloring $\Psi(1) = \Psi(7) = [1, 2C], \Psi(2) = \Psi(6) = [2C+1, 12C], \Psi(3) = \Psi(5) = [1, C] \cup [12C+1, 111C], \Psi(4) = [C+1, 2C] \cup [2C+1, 12C]$ (see Figure 5.6). We will show that Ψ is a square-optimal solution of the instance. In order to do this, we introduce a new definition, which will be helpful in proving the optimality of a coloring. Given a demand function x, the set $X \subseteq \mathbb{N}$ is *left critical* for the node i, if for every coloring Ψ, $|\Psi(i) \cap X| = |X| - \alpha$ implies $f_\Psi([1, i]) \geq \mathrm{OPT}([1, i], x) + \alpha$. That is, in the restriction of the instance from 1 to i, if i does not use α colors from X, then its sum is greater that the optimum by at least α. Similarly, X is *right critical* for i, if for every coloring Ψ, $|\Psi(i) \cap X| = |X| - \alpha$ implies $f_\Psi([i, n]) \geq \mathrm{OPT}([i, n], x) + \alpha$.

In the instance described above, clearly $[1, 2C]$ is a left critical set for node 1, since if α colors are missing from this set in a coloring Φ, then $f_\Phi(1) \geq 2C + \alpha$. Next we show that the set $[2C+1, 12C]$ is left critical for node 2. Assume that α colors are missing from this set: let $a = |\Phi(2) \cap [1, 2C]|$ and $b = |\Phi(2) \cap [12C+1, \infty]|$, clearly $\alpha = a + b = 10C - |\Phi(2) \cap [2C+1, 12C]|$. Since $[1, 2C]$ is a left critical

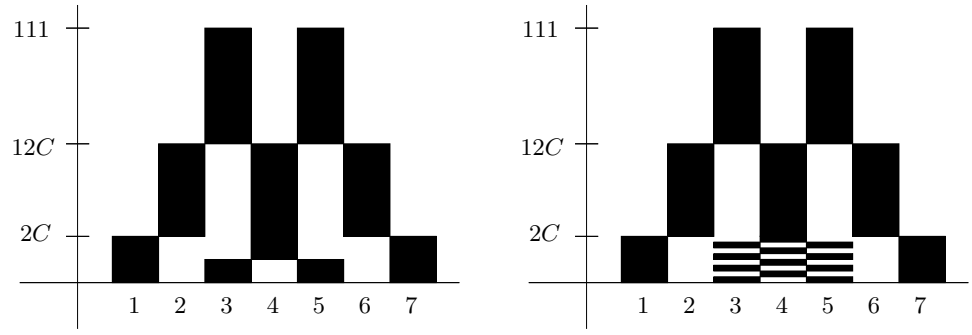


Figure 5.6: Two different square-optimal solutions of the instance $x(1) = 2C$, $x(2) = 10C$, $x(3) = 100C$, $x(4) = 11C$, $x(5) = 100C$, $x(6) = 10C$, $x(7) = 2C$. In the second solution, nodes 3, 4 and 5 have C preemptions. The vertical axis is distorted to save space.

set for node 1, if node 2 uses a colors from this set, then $f_\Phi(1) \geq \mathrm{OPT}(\{1\}, x) + a = 2C + a$. If node 2 uses b colors above $12C$, then its finish time is at least $12C + b$, thus $f_\Phi([1,2]) \geq 2C + a + 12C + b = \mathrm{OPT}([1,2], x) + a + b = \mathrm{OPT}([1,2], x) + \alpha$, what we had to show. It is easy to see that $\Phi^*(1) = [1, 2C]$, $\Phi^*(2) = [2C + 1, 12C]$, $\Phi^*(3) = [1, 2C] \cup [12C + 1, 110C]$ is an optimum coloring of the nodes 1 to 3. We show that $[1, 2C] \cup [12C + 1, 110C]$ is a left critical set of node 3. If α colors are missing in $\Phi(3)$ from this set, then $|\Phi(3) \cap [2C+1, 12C]| = a$ and $|\Phi(3) \cap [110C+1, \infty]| = b$ with $a+b = \alpha$. Now, because $[2C+1, 12C]$ is a left critical set for node 2, we have $f_\Phi([1,3]) \geq \mathrm{OPT}([1,2], x) + a + 110C + b = \mathrm{OPT}([1,3], x) + \alpha$. By a symmetric argument, it is also true that $[1, 2C] \cup [12C + 1, 110C]$ is right critical for node 5.

The coloring Ψ described above has sum $262C$. We show that it is an optimum solution. Assume there is a coloring Φ with smaller sum, we can assume that $\Phi(4)$ does not contain colors above $100C$, since otherwise the sum is at least $300C$. Let $|\Phi(4) \cap [1, 2C]| = a$, $|\Phi(4) \cap [2C + 1, 12C]| = b$ and $|\Phi(4) \cap [12C + 1, 100C]| = c$. Clearly $f_\Phi(4) \geq 12C + c$, $f_\Phi(4) \geq 13C - a$ and $a + c \geq C$. As it was show above, $[1, 2C] \cup [12C + 1, 110C]$ is a left critical set of node 3, and in $\Phi(3)$ at least $a + c$ colors are missing from this set, thus $f_\Phi([1,3]) \geq \mathrm{OPT}([1,3], x) + a + c = 124C + a + c$, and similarly, $f_\Phi([5,7]) \geq 124C + a + c$. Now we have $f_\Phi([1,7]) = f_\Phi([1,3]) + f_\Phi(4) + f_\Phi([5,7]) \geq 2(124C + a + c) + 13C - a \geq 263C - a$, thus $a > C$. However, we also have $f_\Phi([1,7]) \geq 2(124C + a + c) + 12C + c \geq 260C + 2a$ which is higher than $262C$ if $a > C$, therefore Φ cannot have sum strictly smaller than Ψ, which proves the optimality of Ψ. To see that Ψ is square-optimal, we show that in every other optimum solution Φ the finish times are the same as in Ψ. It was shown that Φ can have sum 262 only if $a = C$ which implies $f_\Phi(4) = 12C$. Clearly $f_\Phi([1,2]) \geq f_\Psi([1,2]) = 12C$ and $f_\Phi(3) \geq f_\Psi(3) = x(3) + x(4) = 111C$, thus Φ and Ψ can have the same sum if these inequalities are equalities, implying that the finish times are the same in the optimum colorings.

Now consider the coloring $\Psi'(1) = \Psi'(7) = [1, 2C], \Psi'(2) = \Psi'(6) = [2C + 1, 12C], \Psi'(3) = \Psi'(5) = \{1, 3, \ldots, 2C-1\} \cup [12C+1, 111C], \Psi'(4) = \{2, 4, \ldots, 2C\} \cup [2C+1, 12C]$. This is a square-optimal coloring (it has the same finish times as Ψ), but node 3 has $C = \Theta(p)$ preemptions.

5.1.5 Perfect graphs

What can be said about graphs more general than paths? It can be shown that the logarithmic bound of Theorem 5.1.14 does not hold even in the case of binary trees (use the gadgets defined in Section 5.2.1). However, we show that in bipartite graphs with n vertices, there is always an optimum coloring where every vertex has at most n preemptions, and the same holds for perfect graphs with the weaker bound n^2. Unlike in Theorem 5.1.14, these bounds are independent of p, the size of the maximum demand.

First we introduce a well-studied coloring problem, where the number of different colors used has to be minimized instead of the total sum. In Lemma 5.1.17, we will show why this problem is relevant to the study of preemptions in minimum sum coloring.

Minimum Makespan Multicoloring

Input: A graph $G(V, E)$ and a *color requirement (or length, demand)* function x: $V \to \mathbb{N}$.

Output: A *multicoloring* $\Psi : V \to 2^{\mathbb{N}}$ such that

- $\Psi(u) \cap \Psi(v) = \emptyset$ if u and v are neighbors in G, and
- $|\Psi(v)| = x(v)$.

Goal: The *finish time* of vertex v in coloring Ψ is the highest color assigned to it, $f_\Psi(v) = \max\{i : i \in \Psi(v)\}$. The goal is to minimize $\max_{v \in V} f_\Psi(v)$, the *makespan* of the coloring Ψ.

In the case of bipartite graphs, an easy argument shows that there is always a minimum makespan coloring without preemptions.

Lemma 5.1.15. *If G is bipartite, then for every demand function x, there is a minimum makespan coloring Ψ where every vertex is colored non-preemptively.*

Proof. Let $S = \max_{uv \in E}(x(u) + x(v))$ be the maximum demand appearing on the endpoints of an edge, clearly at least S colors are required for a proper coloring. We show that there is a non-preemptive coloring using S colors. Let V_1 and V_2 be the two bipartition classes of the graph. Let $\Psi(v) = [1, x(v)]$ if $x \in V_1$ and let $\Psi(v) = [S - x(v) + 1, S(v)]$ if $x \in V_2$. This is a proper coloring: if $e = uv$, then $\Psi(u) \cap \Psi(v) = \emptyset$ follows from $x(u) + x(v) \leq S$. ■

In order to bound the number of preemptions required in perfect graphs, the polyhedral properties of perfect graphs will be used. We only sketch the main idea, the reader is referred to [GLS88] for the background.

Lemma 5.1.16. *If G is a perfect graph with n vertices, then there is a minimum makespan coloring Ψ where every vertex has at most $n - 1$ preemptions.*

Proof. Denote by $S(G)$ the set of all independent sets in G. Let A be a $|S(G)|$ times n matrix whose rows are the characteristic vectors of the sets in $S(G)$. Let the components of the n dimensional vector $\mathbf{c}$ be the values of $x(v)$, and consider the following linear program:

$$\begin{aligned} \max \mathbf{c}'\mathbf{x} \\ A\mathbf{x} \leq \mathbf{1} \\ \mathbf{x} \geq 0 \end{aligned} \tag{5.1}$$

It can be easily verified that the integer solutions of the dual program corresponds to the solutions of the makespan problem: we select some independent sets, with possibly selecting a set multiple times, such that every vertex v is in at least $x(v)$ selected independent sets. If G is a perfect graph, then it is well known that this (5.1) is a Totally Dual Integral system. For every integer vector $\mathbf{c}$, the dual program has an optimum integral basic solution $\mathbf{y}$. Since A has n columns, the basic solution $\mathbf{y}$ has at most n nonzero components, thus there are n independent sets $S_1, \ldots, S_n$, and n nonnegative integers $y_1, \ldots, y_n$ such that taking every independent set S_i with multiplicity y_i covers every vertex v at least $x(v)$ times, and $\sum_{i=1}^n y_i$ is as small as possible. Now we construct a minimum makespan coloring where every vertex has at most $n-1$ preemptions. The first y_1 colors are used only on the vertices in S_1, the colors from $y_1 + 1$ to $y_1 + y_2$ are used only in S_2 and so on. Obviously, this is a proper coloring using $\sum_{i=1}^n y_i$ colors. ■

If no vertex has finish time between a and b, then we can reorder the colors between a and b without increasing the finish time of any of the vertices. In the following lemma, we perform several such reorderings, to obtain a minimum sum coloring with few preemptions.

Lemma 5.1.17. *Assume that G is a graph such that for every demand function $x(v)$, G has a minimum makespan coloring with at most m preemptions at every vertex. Then for every demand function $x(v)$, there is a minimum sum coloring with at most $(m+1)n$ preemptions at every vertex.*

Proof. Let Ψ be a minimum sum coloring of G, we transform it into a coloring with $(m+1)n$ preemptions at every vertex, without increasing the sum. More precisely, if the vertices are ordered such that $f_\Psi(v_1) \leq f_\Psi(v_2) \leq \cdots \leq f_\Psi(v_n)$, then we construct an optimum coloring where every vertex has at most $m+1$ preemptions between $f_\Psi(v_i)$ and $f_\Psi(v_{i+1})$.

Let $f_i = f_\Psi(v_i)$ and let $f_0 = 0$. For every $1 \leq i \leq n$, we perform the following transformation. Let $x'_i(v) = |\Psi(v) \cap [f_{i-1}+1, f_i]|$, and let Ψ'_i be a minimum makespan coloring of G with demand function x'_i. Obviously, Ψ'_i uses most $f_i - f_{i-1}$ colors, and it can be assumed that every vertex in Ψ'_i has at most m preemptions. Now we replace the range of colors $[f_{i-1}+1, f_i]$ in Ψ with those appearing in Ψ'_i. Formally, if $c \in [f_{i-1}+1, f_i]$, then let $c \in \Psi^*(v)$ if and only if $c - f_{i-1} \in \Psi'_i(v)$. It is obvious that in $\Psi^*(v)$, there are at most $m+1$ preemptions in the range $[f_{i-1}+1, f_i]$ (it is $m+1$, not only m, since f_i can be a preemption in $\Psi^*(v)$ even if $f_i - f_{i-1}$, the last color of $\Psi'(v)$ is not a preemption). Repeating this procedure for every $1 \leq i \leq m$ results in a coloring with at most $(m+1)n$ preemptions at every vertex. ■

Combining Lemma 5.1.17 with Lemma 5.1.15 and 5.1.16 gives:

Corollary 5.1.18. *If G is a perfect graph with n vertices, then there is an optimum minimum sum coloring where there are at most n^2 preemptions at every vertex. Moreover, if G is bipartite, then the same holds with at most n preemptions.* ■

Lemma 5.1.16 shows that in the case of perfect graphs, there is always an optimum solution that can be represented in size polynomial in the length of the input, even if p is not polynomial in the number of vertices: it is enough to give at most n^2 intervals for each of the n vertices. This can serve as a polynomial-size certificate proving that $\mathrm{OPT}(G) \leq k$, thus

Corollary 5.1.19. *The preemptive sum multicoloring for perfect graphs is in* NP*, even if p is not polynomial in n.* ■

5.2 Complexity of minimum sum multicoloring for trees

In this section we show that minimum sum multicoloring is NP-hard for binary trees. The proof is inspired by the proof of Theorem 2.2.1 where it is shown that list multicoloring is NP-hard for trees. We will give a similar reduction from the maximum independent set problem. However, when we were reducing to the list multicoloring problem, we could use the lists to exclude certain colors from the solution. In the minimum sum multicoloring problem there are no such lists, every color can be used at every vertex. Therefore we will construct "penalty gadgets" that can exclude colors by ensuring that if a forbidden color is used, then the gadget can be colored only with large penalty. We will also use the color copying trick used in Theorem 2.2.2 to make the reduction work for binary trees.

Let us introduce some notations. If $V' \subseteq V$ and Ψ is a coloring then let $f_\Psi(V') = \sum_{v \in V'} f_\Psi(v)$. Similarly, $x(V') = \sum_{v \in V'} x(v)$. The sum of the optimum coloring of (G, x) is denoted by $\mathrm{OPT}(G, x)$, or by $\mathrm{OPT}(G)$ if the function $x(v)$ is clear from the context. The notation $[a, b]$ stands for the set $\{a, a+1, \ldots, b\}$ if $a \leq b$, otherwise it is the empty set.

5.2.1 The penalty gadgets

The goal of the penalty gadgets is that by connecting such a gadget to a node v, we can force v not to use certain colors: if node v uses a forbidden color, then the gadget can be colored only with a "very large" penalty.

For offset t, demand size d and penalty C we define a tree $T_{t,d,C}$. The root r of this tree will be connected to some node v. When the root r of this tree uses the set $[t+1, t+d]$, then the tree can be colored optimally. On the other hand, if v uses even one color from $[t+1, t+d]$, then r cannot have the set $[t+1, t+d]$ and this constrains the gadget in such a way that it can be colored only with $f_\Psi(T_{t,d,C}) \geq OPT(T_{t,d,C}, x) + C$. When C is sufficiently large, then this will force v to use colors not in $[t+1, t+d]$.

Proposition 5.2.1. *For integers $d, C > t \geq 0$ there is a binary tree $T_{t,d,C}$ and a demand function $x(v)$ such that*

1. *The root r has demand $x(r) = d$.*
2. *$\Psi(r) = [t+1, t+d]$ in every optimum coloring Ψ.*
3. *If $\Psi(r) \neq [t+1, t+d]$ for a coloring Ψ, then $f_\Psi(T_{t,d,C}) \geq \mathrm{OPT}(T_{t,d,C}, x) + C$.*
4. *The demand x of every vertex is polynomially bounded by d and C.*

Furthermore, there is an algorithm which, given t, d and C, outputs the tree $T_{t,d,C}$, the demand function x and the value $\mathrm{OPT}(T_{t,d,C}, x)$ in time polynomial in d and C.

Proof. Let $k = \lceil \log_2(C+t) \rceil$ and $\widehat{C} = 2^k$. Obviously, $C + t \leq \widehat{C} < 2(C+t)$. The tree $T_{t,d,C}$ consists of a complete binary tree and some attached paths. The complete binary tree T_0 has $k+1$ levels, the root r is on level 1 and the leaves, $\ell_1, \ell_2, \ldots, \ell_{\widehat{C}}$, are on level $k+1$. Attach a path of $k+3$ nodes to every leaf: node ℓ_i ($1 \leq i \leq \widehat{C}$) is connected to path P_i: $a_{i,k+2}, a_{i,k+1}, \ldots, a_{i,2}, a_{i,1}, a_{i,0}$ (nodes ℓ_i and $a_{i,k+2}$ are neighbors). Figure 5.7 shows the construction for $t = 2$, $d = 4$, $C = 6$. Clearly, $T_{t,d,C}$ has $2\widehat{C} - 1 + (k+3)\widehat{C}$ nodes, which is polynomially bounded in C.

We say that a node is of type j if it is either on the jth level of T_0 or it is an $a_{i,j}$ for some $1 \leq i \leq \widehat{C}$.

The demand $x(v)$ will depend only on the type of node v. Let

$$\begin{aligned} g(0) &= t, \\ g(1) &= d, \\ g(n) &= (3d + t + C) \cdot 4^{n-2} \text{ for } n \geq 2. \end{aligned}$$

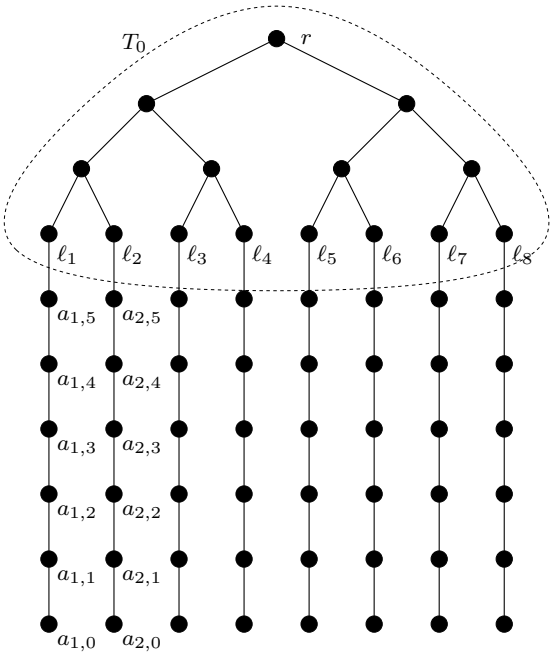
Obviously, $g(n)$ is monotone and it is easy to see that

$$g(i+1) \geq 3g(i) + t + C \geq g(i-1) + t + C$$

for all $i \geq 1$ (these inequalities will be used later).

For a node v of type i let $x(v) = g(i)$. This implies that $x(r) = g(1) = d$ for the root r. The maximum value of $x(v)$ is $g(k+2) = (3d + t + C) \cdot 4^k$, which is bounded by a polynomial of d and C.

We describe a proper multicoloring Ψ, which will turn out to be the unique optimum solution. The same color set is assigned to the nodes of the same type. Start with $\Psi(v) = [1, t]$ for every node v of type 0. Then color the different types in increasing order: assign to the nodes of type i the first $g(i)$ colors not used by the type $i-1$ nodes. This gives a proper coloring since the already colored neighbors of type i nodes are type $i-1$ nodes. Notice that the root r receives the set $[t+1, t+d]$, as required. It is easy to prove that the finish time of a node v of type i is $f_\Psi(v) = g(i) + g(i-1) = x(i) + x(i-1)$ since there will be exactly $g(i-1)$ "skipped" colors and the finish time of nodes of type i is greater then the finish time



Type	Demand	Set of colors assigned by Ψ						
1	4		$[3,6]$					
2	20	$[1,2]$		$[7,24]$				
3	80		$[3,6]$		$[25,100]$			
4	320	$[1,2]$		$[7,24]$		$[101,400]$		
5	1280		$[3,6]$		$[25,100]$			$[401,1600]$
4	320	$[1,2]$		$[7,24]$		$[101,400]$		
3	80		$[3,6]$		$[25,100]$			
2	20	$[1,2]$		$[7,24]$				
1	4		$[3,6]$					
0	2	$[1,2]$						

Figure 5.7: The tree $T_{t,d,C}$ for $t = 2,\ d = 4$ and $C = 6$. The nodes on the same level have the same type. On the right are the demands and also the colors assigned by the optimum coloring.

of the nodes of type $i-1$ because $g(i) > g(i-2)$. The following simple observation will be used later: if u is a type i node and v is its type $i+1$ neighbor, then in every coloring Φ, the equalities $\Phi(u) = \Psi(u)$ and $f_\Phi(v) = f_\Psi(v) = g(i+1) + g(i)$ imply $\Phi(v) = \Psi(v)$. This follows directly from the definition of Ψ: there is just one way of choosing the first $x(v) = g(i+1)$ colors not used by u.

The following three lemmas show that Ψ is an optimum coloring, and if a coloring Φ assigns to r a set different from $\Psi(r) = [t+1, t+d]$, then $f_\Phi(T_{t,d,C}) \geq f_\Psi(T_{t,d,C}) + C$.

Lemma 5.2.2. *(a) $f_\Phi(T_0) \geq f_\Psi(T_0) - t$ holds for every coloring Φ of $(T_{t,d,C}, x)$.*
(b) If $\Phi(r) = \Psi(r)$, then $f_\Phi(T_0) \geq f_\Psi(T_0)$.
(c) If there is a $v \in T_0 \setminus \{r\}$ such that $f_\Phi(v) < f_\Psi(v)$, then $f_\Phi(T_0) \geq f_\Psi(T_0) + C$.

Proof. Let $L = \{v \in T_0 : f_\Phi(v) < f_\Psi(v)\}$ and let $H = T_0 \setminus L$ be its complement in T_0. We note that L is an independent set. To see this, let v and u be neighbors of type i and $i+1$, respectively. The sum of their demand is $g(i) + g(i+1)$, thus at least one of them must have finish time not smaller than $g(i) + g(i+1)$. Clearly this makes it impossible to have $f_\Phi(v) < f_\Psi(v) = g(i)+g(i-1)$ and $f_\Phi(u) < f_\Psi(u) = g(i)+g(i+1)$ simultaneously.

Partition the vertices of T_0 as follows. Define a subset S_v for every node $v \in H$. Let $v \in S_v$ for every $v \in H$, and $u \in L$ is in S_v iff v is the parent of u. When the root r is in L then r forms a set itself, $S^* = \{r\}$. It is clear that this defines a partition, every vertex is in exactly one subset. Apart from S^*, every subset contains a node from H and zero, one or two nodes from L.

Assume that the set S_v contains no node from L. Then $f_\Phi(S_v) \geq f_\Psi(S_v)$ follows from the definition of H and L. Now consider a set S_v which has at least one node from L. It contains a type i node v from H and one or two type $i+1$ nodes (u_1, u_2) from L. Since v and u_z $(z = 1, 2)$ are neighbors and the sum of their demand is $g(i) + g(i+1)$, at least one of them must have finish time at least $g(i) + g(i+1)$.

Since u_z is in L, we have $f_\Phi(u_z) < f_\Psi(u_z) = g(i) + g(i+1)$, thus $f_\Phi(v) \geq g(i) + g(i+1)$. Therefore, $f_\Phi(v) - f_\Psi(v) \geq (g(i) + g(i+1)) - (g(i-1) + g(i)) = g(i+1) - g(i-1)$. Since $f_\Psi(u_z) = g(i+1) + g(i)$ and $x(u_z) = g(i+1)$, clearly $f_\Phi(u_z) - f_\Psi(u_z) \geq -g(i)$. Now

$$f_\Phi(S_v) - f_\Psi(S_v) \geq (g(i+1) - g(i-1)) - 2g(i) \geq g(i+1) - 3g(i) \geq C + t,$$

where the last inequality follows from $g(i+1) \geq 3g(i) + C + t$.

If r is in S^*, then $f_\Phi(S^*) = f_\Psi(S^*) + (f_\Phi(r) - f_\Psi(r))$ holds. Therefore $f_\Phi(T_0) \geq f_\Psi(T_0) + (f_\Phi(r) - f_\Psi(r)) \geq f_\Psi(T_0) - t$, since $f_\Phi(r) \geq d$. This proves statement (a), and (b) also follows because $\Phi(r) = \Psi(r)$ implies $f_\Phi(r) - f_\Psi(r) = 0$. Furthermore, if $f_\Phi(u) < f_\Psi(u)$ for some $u \in T_0 \setminus \{r\}$, then $f_\Phi(S_v) \geq f_\Psi(S_v) + C + t$ for the set S_v of the partition that contains u. This proves statement (c). ■

Lemma 5.2.3. *$f_\Phi(P_i) > f_\Psi(P_i)$ holds for every coloring $\Phi \neq \Psi$ of $T_{t,d,C}$ and for every $1 \leq i \leq \widehat{C}$.*

Proof. Assume that $f_\Phi(P_i) \leq f_\Psi(P_i)$, define $L = \{v \in P_i : f_\Phi(v) < f_\Psi(v)\}$ and $H = P_i \setminus L$. If $f_\Phi(P_i) \leq f_\Psi(P_i)$ and Φ is different from Ψ, then there is a $v \in P_i$ such that $f_\Phi(v) < f_\Psi(v)$, thus L is not empty. As in Lemma 5.2.2, it is easy to see that L is an independent set. The nodes of P_i are partitioned into $|H|$ classes: if $v \in H$ then v is in S_v, if $u \in L$ then u is in S_v, where v is the child of u. Notice that $a_{i,0} \in H$ since $f_\Psi(a_{i,0}) = x(a_{i,0}) = g(0) \leq f_\Phi(a_{i,0})$.

We prove that $f_\Phi(S_v) \geq f_\Psi(S_v)$ for every S_v. If $S_v = \{v\}$, then it is clear that $f_\Phi(S_v) \geq f_\Psi(S_v)$ holds. Assume that $S_v = \{u, v\}$, node $u \in L$ is type $j+1$, and $v \in H$ (its child) is type $j \geq 0$. The finish time of node v is at least $x(u) + x(v) = g(j+1) + g(j)$, therefore

$$f_\Phi(S_v) \geq x(u) + (x(u) + x(v)) = g(j+1) + (g(j+1) + g(j))$$

holds. On the other hand, if $j \geq 1$, then $f_\Psi(S_v) = (g(j+1)+g(j))+(g(j)+g(j-1))$, thus $f_\Phi(S_v) > f_\Psi(S_v)$ follows from $g(j+1) > g(j) + g(j-1)$. In the case $j = 0$, we have $f_\Psi(S_v) = t + (t+d) = g(j) + (g(j) + g(j+1)) < f_\Phi(S_v)$, since $f_\Phi(S_v) \geq g(j+1) + (g(j) + g(j+1)) = d + (t+d)$ (recall that $t < d$). Since H is not empty, there is at least one subset S_v in the partition with $f_\Phi(S_v) > f_\Psi(S_v)$, contradicting $f_\Phi(P_i) \leq f_\Psi(P_i)$. ■

Lemma 5.2.4. *If $\Phi(r) \neq \Psi(r) = [t+1, t+d]$, then $f_\Phi(T_{t,d,C}) \geq f_\Psi(T_{t,d,C}) + C$.*

Proof. Denote by $P^* = \bigcup_{i=1}^{\widehat{C}} P_i = T_{t,d,C} \setminus T_0$ the union of the paths. If there is a node $v \in T_0 \setminus \{r\}$ with $f_\Phi(v) < f_\Psi(v)$, then by part (c) of Lemma 5.2.2 $f_\Phi(T_0) \geq f_\Psi(T_0) + C$, and by Lemma 5.2.3 $f_\Phi(P^*) \geq f_\Psi(P^*)$ follows, which implies $f_\Phi(T_{t,d,C}) \geq f_\Psi(T_{t,d,C}) + C$, and we are done. Therefore it can be assumed that $f_\Phi(v) \geq f_\Psi(v)$ for every node $v \in T_0 \setminus \{r\}$. Furthermore, if there is a $v \in T_0$ with $f_\Phi(v) \geq f_\Psi(v) + C + t$, then $f_\Phi(T_0) \geq f_\Psi(T_0) + C$, thus $f_\Phi(P^*) \geq f_\Psi(P^*)$ implies $f_\Phi(T_{t,d,C}) \geq f_\Psi(T_{t,d,C}) + C$. In the following, it will be assumed that $f_\Psi(v) \leq f_\Phi(v) \leq f_\Psi(v) + C + t$ holds for every $v \in T_0 \setminus \{r\}$.

Call a vertex v *changed* in Φ if $\Phi(v) \neq \Psi(v)$. The goal is to show that if the root r is changed, then all the nodes $a_{1,k+2}, a_{2,k+2}, \ldots, a_{\widehat{C},k+2}$ are changed. Let v be a node of type i in T_0 and let u be one of its children, a node of type $i+1$. If v is changed, then there is a color $j \in \Phi(v)$ and $j \notin \Psi(v)$. We consider two cases. If $j \leq f_\Psi(u)$, then by the fact that $j \notin \Psi(v)$ and the way Ψ was defined $j \in \Psi(u)$ follows. Therefore u is also changed since $j \in \Phi(v)$ implies $j \notin \Phi(u)$. In the second case, where $j > f_\Psi(u) = g(i+1) + g(i)$ we have

$$\begin{aligned} f_\Phi(v) \geq j > g(i+1) + g(i) &= (g(i+1) - g(i-1)) + (g(i) + g(i-1)) \\ &= g(i+1) - g(i-1) + f_\Psi(v) \geq f_\Psi(v) + C + t, \end{aligned}$$

contradicting the assumption $f_\Phi(v) \leq f_\Psi(v) + C + t$.

Assume that $f_\Phi(T_{t,d,C}) < f_\Psi(T_{t,d,C}) + C$. By applying the previous result inductively, one finds that all the leaves ℓ_i and their children $a_{i,k+2}$ $(1 \le i \le \widehat{C})$ are changed. Lemma 5.2.3 ensures that Φ is not an optimum coloring of P_i, thus $f_\Phi(P_i) \ge f_\Psi(P_i) + 1$ and $f_\Phi(P^*) \ge f_\Psi(P^*) + \widehat{C} \ge f_\Psi(P^*) + C + t$. By Lemma 5.2.2, $f_\Phi(T_0) \ge f_\Psi(T_0) - t$, hence $f_\Phi(T_{t,d,C}) \ge f_\Psi(T_{t,d,C}) + C$. ■

To finish the proof of Prop. 5.2.1, we have to show that requirements 2 and 3 hold. If $\Phi(r) = \Psi(r)$, then by part (b) of Lemma 5.2.2 and by Lemma 5.2.3, $f_\Phi(T_{t,d,C}) \ge f_\Psi(T_{t,d,C})$. If $\Phi(r) \ne \Psi(r)$, then by Lemma 5.2.4, $f_\Phi(T_{t,d,C}) \ge f_\Psi(T_{t,d,C}) + C$. Therefore the coloring Ψ is an optimum coloring and the tree satisfies the requirements of the proposition.

Clearly, the tree $T_{t,d,C}$ and the demand function x can be constructed in polynomial time. The sum of the optimum solution can be also calculated, by adding the appropriate finish time of every node. ■

5.2.2 The reduction

We will reduce the maximum independent set problem to the minimum sum coloring problem in binary trees. In the decision version of the minimum sum coloring problem, the input is a graph G, a demand function $x(v)$, and an integer K. The question is whether there exists a multicoloring Ψ with sum less than K. The reduction is based on the proof of Theorem 2.2.2. The penalty gadgets $T_{t,d,C}$ of Section 5.2.1 are used to imitate the effect of the color lists.

More precisely, the penalty gadget is used in two different ways: as a lower penalty gadget and as an upper penalty gadget. The *lower penalty gadget* $T^L_{d,C}$ is a tree $T_{0,d,C}$. By connecting the root of such a tree to a node v, the node v is forced to use only colors greater than d: otherwise the gadget can be colored only with a penalty C. A tree is a tree of *type* T^L if it is the tree $T^L_{d,C}$ for some d and C.

The *upper penalty gadget* $T^U_{d,C}$ is a tree $T_{d,C,C}$. If this gadget is connected to a node v, then this forces v to use only colors not greater than d. If v uses only colors at most d, then its finish time is at most d, and the gadget can be colored optimally. If v uses a color greater than d but not greater than $d+C$, then the gadget can be colored only with penalty C. If v uses colors greater than $d+C$, then it has finish time at least $d+C$, which is a penalty of at least C compared to the case when v uses only colors at most d.

Theorem 5.2.5. *The minimum sum preemptive multicoloring problem is* NP*-complete on binary trees when the value of the demand function is polynomially bounded.*

Proof. Let a graph $G(V, E)$ and an integer k be given. Denote $n = |V|$, $m = |E|$, and let $C = 8mn$. Let integers $u_{i,1} < u_{i,2}$ denote the two end vertices of the ith edge in G.

We define a binary tree T, which consists of a core $\widehat{T}$ and some attached subtrees of type T^L and T^U. We start with a path of $2m-1$ nodes, $a_1, b_1, a_2, b_2, \ldots, a_{m-1}, b_{m-1}, a_m$. Define $x(a_i) = k$ $(1 \le i \le m)$ and $x(b_i) = C + n - k$ $(1 \le i \le m-1)$. For every $1 \le i \le m$ attach a path of 6 nodes to a_i. Let these nodes be $c_{i,1}, d_{i,1}, c_{i,2}, d_{i,2}, c_{i,3}, d_{i,3}$. Let $x(c_{i,j}) = 1$, $x(d_{i,j}) = C + n - 1$ $(j = 1, 2)$ and $x(c_{i,3}) = 1$, $x(d_{i,3}) = u_{i,2} - u_{i,1} - 1$. Clearly, $x(v) \ge 0$ for every node v. This completes the definition of $\widehat{T}$. Now attach trees of type T^L and T^U to $\widehat{T}$ as follows (see Figure 5.8):

- a $T^U_{C+n,2C}$ to every node b_i $(1 \le i \le m-1)$,
- a $T^U_{n,C}$ to the node a_1,
- a $T^U_{C+n,2C}$ to every node $d_{i,j}$ $(1 \le i \le m,\ j = 1, 2)$,
- a $T^U_{u_{i,2}+1,C}$ to every node $c_{i,1}$ $(1 \le i \le m)$,
- a $T^L_{u_{i,1}-1,C}$ to every node $c_{i,2}$ $(1 \le i \le m)$,
- a $T^L_{u_{i,1},C}$ and a $T^U_{u_{i,2}-1,C}$ to every node $d_{i,3}$ $(1 \le i \le m)$.

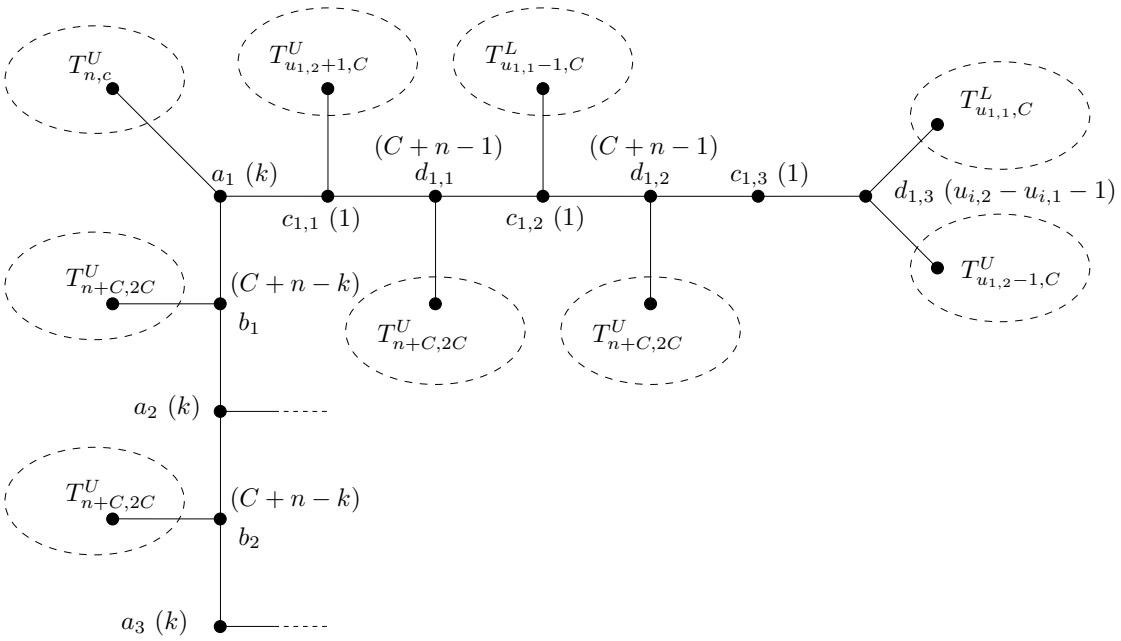


Figure 5.8: The tree T for $m = 3$. For the sake of clarity, the nodes $c_{i,j}, d_{i,j}$ for $i \geq 2$ and the subtrees connected to these nodes are omitted. The numbers in parentheses are the demand of the vertices.

It is clear that the size of the resulting tree T is polynomial in n, the number of vertices of G, because $\widehat{T}$ has $8m-1$ nodes and we attach $7m$ trees to it, each of size bounded by a polynomial in $C+n$.

As required by Prop. 5.2.1, the algorithm that constructs the trees of type T^U and T^L also outputs the minimum sum of these $7m$ trees, that is, the value of $\mathrm{OPT}(T \setminus \widehat{T})$. Let $K = \mathrm{OPT}(T \setminus \widehat{T}) + x(\widehat{T}) + C$.

The intuition behind the construction is that in a "well-behaved" solution, when the coloring of the T^L and T^U trees are optimal, for every i, the three nodes $c_{i,1}, c_{i,2}, c_{i,3}$ have the same color. The trees attached to these nodes ensure that this color must be either $u_{i,1}$ or $u_{i,2}$, one of the end nodes of the ith edge in G. This color cannot appear in a_i, this is the reason why the k colors assigned to the nodes a_i form an independent set, at least one end node of each edge is not in the set.

First we prove that if there is an independent set S of size k, then T can be colored with sum smaller than K. Let $\widehat{u}_i \in \{u_{i,1}, u_{i,2}\}$, $\widehat{u}_i \notin S$ be an end node of the ith edge. Assume that Ψ colors all the trees of type T^U and T^L optimally, i.e., $f_\Psi(T \setminus \widehat{T}) = \mathrm{OPT}(T \setminus \widehat{T})$ and let

- $\Psi(a_i) = S$ $(1 \leq i \leq m)$,
- $\Psi(b_i) = [1, C+n] \setminus S$ $(1 \leq i \leq m-1)$,
- $\Psi(c_{i,j}) = \{\widehat{u}_i\}$ $(1 \leq i \leq m, j = 1,2,3)$,
- $\Psi(d_{i,j}) = [1, C+n] \setminus \{\widehat{u}_i\}$ $(1 \leq i \leq m,\ j = 1,2)$,
- $\Psi(d_{i,3}) = [u_{i,1}+1, u_{i,2}-1]$ $(1 \leq i \leq m)$.

It is straightforward to verify that Ψ is a proper coloring of T. Notice that $f_\Psi(v) \leq x(v) + n$ holds for every node v of $\widehat{T}$, thus $f_\Psi(\widehat{T})$ can be bounded by $x(\widehat{T}) + |\widehat{T}|n$. Therefore $f_\Psi(T) = f_\Psi(T \setminus \widehat{T}) + f_\Psi(\widehat{T}) \leq \mathrm{OPT}(T \setminus \widehat{T}) + x(\widehat{T}) + |\widehat{T}|n = \mathrm{OPT}(T \setminus \widehat{T}) + x(\widehat{T}) + (8m-1)n < \mathrm{OPT}(T \setminus \widehat{T}) + x(\widehat{T}) + C = K$, what we had to show.

To prove the other direction, we will show that when there is a coloring Ψ with sum $f_\Psi(T) < K$, then there is a set of k independent vertices in G. Obviously $f_\Psi(T) = f_\Psi(\widehat{T}) + f_\Psi(T - \widehat{T}) \geq x(\widehat{T}) + \mathrm{OPT}(T \setminus \widehat{T})$. If there is even one node $v \in \widehat{T}$ such that $f_\Psi(v) \geq x(v) + C$, then $f_\Psi(\widehat{T}) \geq x(\widehat{T}) + C$ and $f_\Psi(T) \geq \mathrm{OPT}(T \setminus \widehat{T}) + x(\widehat{T}) + C = K$. Thus it can be assumed that $f_\Psi(v) < x(v) + C$ for every $v \in \widehat{T}$. Now consider a tree T_v of type T^L or T^U attached to some node $v \in \widehat{T}$. If $f_\Psi(T_v) \geq \mathrm{OPT}(T_v) + C$, then $f_\Psi(T) \geq x(\widehat{T}) + \mathrm{OPT}(T \setminus \widehat{T}) + C = K$. Thus it can be assumed that $f_\Psi(T_v) < \mathrm{OPT}(T_v) + C$. Therefore, by the definition of T_v, if it is a $T^L_{d,C}$ (resp. $T^U_{d,C}$) tree, then Ψ assigns to its root the set $[1, d]$ (resp. $[d+1, d+C]$). Obviously, it follows that the node v cannot use the colors in this set.

By the argument in the previous paragraph, $f_\Psi(a_1) < x(a_1) + C \leq n + C$ and $\Psi(a_1) \cap [n+1, n+C] = \emptyset$, which implies that $\Psi(a_1)$ contains only colors not greater than n. Similarly, $f_\Psi(b_1) < x(b_1) + C \leq n + 2C$ and $\Psi(b_1) \cap [n+C+1, n+3C] = \emptyset$, which implies that the $n-k+C$ colors in $\Psi(b_1)$ are not greater than $n+C$. This set of colors must be disjoint from the k colors in $\Psi(a_1)$, therefore we have $\Psi(b_1) = [1, n+C] \setminus \Psi(a_1)$. Furthermore, $f_\Psi(a_2) < x(a_2) + C \leq n + C$, hence it must use the k colors not used by b_1, therefore $\Psi(a_2) = \Psi(a_1)$. Continuing on this way, we get $\Psi(a_i) = \Psi(a_1) = S$ for all $2 \leq i \leq m$ and S contains k colors not greater than n.

Assume that the set S is not independent, that is, both end vertices of some edge of G is in this set, $u_{i,1}, u_{i,2} \in S$. From the assumption $f_\Psi(T) < K$ follows that $c_{i,1}$ cannot use either of these colors.

We have seen that $f_\Psi(c_{i,1}) < 1 + C$ and $\Psi(c_{i,1}) \cap [u_{i,2} + 1, u_{i,2} + C] = \emptyset$ follow from the assumption $f_\Psi(T) < K$, which implies that the color of $c_{i,1}$ is at most $u_{i,2} \leq n$. Moreover, since $f_\Psi(d_{i,1}) < 2C + n - 1$ and $\Psi(d_{i,1}) \cap [n + C + 1, n + 3C] = \emptyset$, thus node $d_{i,1}$ must use the first $C + n - 1$ colors missing from $c_{i,1}$, therefore we have $\Psi(d_{i,1}) = [1, C + n] \setminus \Psi(c_{i,1})$. Similarly as in the case of the nodes a_j and b_j, it follows that $\Psi(c_{i,1}) = \Psi(c_{i,2}) = \Psi(c_{i,3}) = \{u\}$. Furthermore, notice that $u \geq u_{i,1}$, since $c_{i,2}$ cannot use the colors below $u_{i,1}$: these colors are assigned to the root of the attached tree $T^L_{u_{i,1}-1,C}$. Similarly, u cannot be in $[u_{i,2} + 1, u_{i,2} + C]$ since $c_{i,1}$ cannot use these colors. Finally, observe that $d_{i,3}$ must have the colors $[u_{i,1} + 1, u_{i,2} - 1]$ which forbids $c_{i,3}$ from using a color between $u_{i,1}$ and $u_{i,2}$. Since u is a color not greater than C, thus it must be either $u_{i,1}$ or $u_{i,2}$.

If the demands are polynomially bounded, then the problem is obviously in NP: a proper coloring with the given sum is a polynomial size certificate, which finishes the proof of NP-completeness. ■

5.3 Complexity of minimum sum edge multicoloring for trees

In this section we prove that minimum sum edge multicoloring is NP-hard for trees, even if every demand is 1 or 2. First we give some definitions that will be useful tools for proving the optimality of certain colorings. Then three families of special trees are introduced, they will be used as gadgets in the NP-hardness proof.

Denote by E_v the set of edges incident to v. Let $\ell(v) = \min_\Psi f_\Psi(E_v)$ be the minimum sum taken on the edges incident to v in any proper coloring. If all the edges incident to v have demand 1, then clearly $\ell(v) = d(v)(d(v) + 1)/2$. Furthermore, it is easy to see that if one edge incident to v has demand 2 and all the the other edges have demand 1, then $\ell(v) = d(v)(d(v) + 1)/2 + 1$.

Let $G(A, B; E)$ be a bipartite graph. An obvious lower bound for $\mathrm{OPT}(G)$ is $\ell(A) = \sum_{v \in A} \ell(v)$. We call a coloring Ψ *A-good* if $f_\Psi(E) = \ell(A)$, which is equivalent to saying that $f_\Psi(E_v) = \ell(v)$ for every $v \in A$. Every A-good coloring is clearly an optimum coloring, and if there is an A-good coloring, then every optimum coloring is A-good.

The tree T_i is defined as follows (see Figure 5.9 for T_6). The root r has a single child v, and node v has $i - 1$ children v_1, v_2, ..., v_{i-1}. The node v_j has a single child v'_j, and node v'_j has $i - 1$ children $v'_{j,1}$, ..., $v'_{j,i-1}$. The edges $v_j v'_j$ have demand 2, the demand of all the other edges are 1. Let the nodes v, v'_1, ..., v'_{i-1} be in A (white nodes on the figure), the remaining nodes are in B. Consider the coloring $\Psi(rv) = i$, $\Psi(vv_j) = j$, $\Psi(v_j v'_j) = \{i, i+1\}$, $\Psi(v'_j v'_{j,k}) = k$. This is an A-good coloring, thus it is an optimum coloring and every optimum coloring is A-good. Therefore if Φ is an optimum coloring, then $\Phi(rv) = i$ because

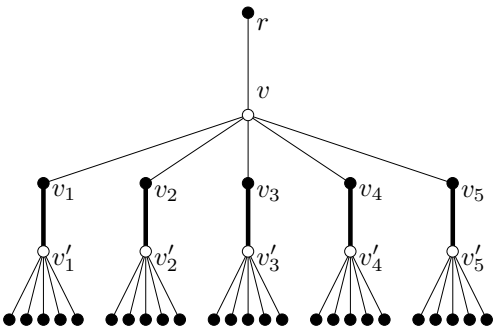


Figure 5.9: The tree T_6. The strong edges have demand 2.

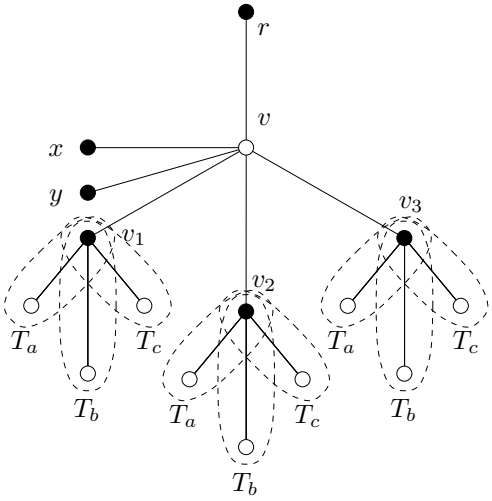


Figure 5.10: The tree $T_{a,b,c}$ with $c = 6$

$f_\Phi(E_{v'_j}) = \ell(E_{v'_j})$ implies $\Phi(v_j v'_j) = \{i, i+1\}$, and $f_\Phi(E_v) = \ell(v)$ implies that one edge in E_v is colored with color i, which can be only rv. Thus the color of rv is i in every optimum coloring.

In the tree $T_{a,b,c}$ (for $a < b < c$) the root r has a single child v, and node v has $c-1$ children x, y, v_1, $\ldots$, v_{c-3} (see Figure 5.10). Every node v_j is the root of a T_a, T_b and T_c tree, as defined in the previous paragraph. We show that in every A-good (optimum) coloring of $T_{a,b,c}$ the edge rv is colored with color a, b or c, and there are three A-good colorings assigning a, b, and c to edge rv, respectively. Color the trees T_a, T_b, T_c at the v_j nodes with an A-good coloring, assign the colors a, b, c to the edges rv, vx, vy in some order, and assign the colors $\{1, \ldots, c\} \setminus \{a, b, c\}$ to the edges $vv_1, \ldots, vv_{c-3}$ in some order. It can be easily verified that this is an A-good (therefore optimum) coloring and the edge rv can have any of the colors a, b, c. To see that in every A-good coloring edge rv can receive only these colors, observe that if the colorings of the T_a, T_b, T_c subtrees rooted at v_j are all A-good, then vv_j cannot be colored with colors a, b, c. In an A-good coloring, the edges incident to v can receive only colors not greater than c, thus rv, vx, vy receive the colors a, b, c. Therefore rv is colored with either a, b or c.

Finally, let the tree $\widehat{T}_i$ be a star: the root r has a child v, and node v has $i+1$ children x, v_1, $\ldots$, v_i. The edges rv, vx have demand 2, the other edges have demand 1. The node v is in A. It is easy to see that if Ψ is a A-good coloring, then $\Psi(rv)$ is either $\{i+1, i+2\}$ or $\{i+3, i+4\}$.

Theorem 5.3.1. *The minimum sum edge multicoloring problem is* NP*-hard for trees, even if every demand is* 1 *or* 2*.*

Proof. The reduction is from 3-occurrence 3SAT, it can be assumed that every variable occurs at most twice positively and at most twice negatively. Given a formula with n variables and m clauses, we construct a tree T and a demand function such that the tree has an A-good coloring (i.e., a coloring with sum at most $\ell(A)$) if and only if the formula is satisfiable.

Consider the variable x_k $(0 \leq k < n)$, which is the hth literal of the ith clause. Let $d_{i,h}$ be $4k+1$ if this is the first positive occurrence of x_k, $4k+2$ if this is the second positive occurrence, $4k+3$ if this is the first negated occurrence, and $4k+4$ if this is the second negated occurrence. Tree T has a node r which is the root of $n+m$ trees. To each variable x_j corresponds a tree $\widehat{T}_{4j}$, and to each clause i a tree $T_{d_{i,1},d_{i,2},d_{i,3}}$. This defines T and its demand function.

Assume that a coloring is A-good, then it is an A-good coloring of all the $n+m$ subtrees (since $r \notin A$). Therefore the root edge of $\widehat{T}_{4j}$ corresponding to variable x_j uses either the set $\{4j+1, 4j+2\}$ or the set $\{4j+3, 4j+4\}$. Assign to the variable x_j the value "false" in the first case and "true" in the second case. This will be a satisfying assignment: if the root edge of the tree corresponding to clause i uses a color from $\{4j+1, 4j+2, 4j+3, 4j+4\}$, then variable x_j satisfies clause i. More precisely, if it uses $4j+1$ or $4j+2$ (resp. $4j+3$ or $4j+4$), then x_j has the value "true" (resp. "false"), and because of the construction, x_j appears in clause i positively (resp. negatively).

To prove the other direction, given a satisfying assignment, we construct an A-good coloring of the tree. Take an A-good coloring of the subtree $\widehat{T}_{4j}$ corresponding to variable x_j such that its root edge uses the colors $\{4j+1, 4j+2\}$ (resp. $\{4j+3, 4j+4\}$) if x_j is "false" (resp. "true"). Since every clause is satisfied by some variable, we can choose an A-good coloring for each subtree corresponding to a clause such that it does not conflict with any of the trees corresponding to the variables. Clearly, this will be an A-good coloring of the tree.

We have reduced the known NP-complete problem to the minimum sum edge multicoloring problem. The reduction can be done in polynomial time, thus minimum sum edge multicoloring is NP-hard. ■

We note that in the proof, the proposed optimum coloring colors non-preemptively every edge with demand 2. Thus the reduction works even if we impose the additional constraint of non-preemptive coloring.

Corollary 5.3.2. *The non-preemptive version of minimum sum edge multicoloring is* NP*-hard for trees, even if every demand is* 1 *or* 2*.* ■

5.4 Approximating minimum sum multicoloring on the edges of trees

In this section we consider the minimum sum edge multicoloring problem restricted to trees. We have seen in Section 5.3 that, unlike the unit demand case, minimum sum edge multicoloring is NP-hard for trees if the demands are allowed to be at most 2. On the other hand, in this section we show that the problem is polynomial time solvable in trees if every demand is the same. This is a consequence of the following scaling property of minimum sum edge multicoloring in trees: if the demand of every edge is multiplied by the same integer q, then the sum of the optimum solution increases by a factor of q. It is easy to see that the sum changes by at most a factor of q, the interesting thing is that this factor is exactly q. The main result in this section is a polynomial time approximation scheme (PTAS) for minimum sum edge multicoloring in trees.

Recently, the vertex coloring version of minimum sum multicoloring was investigated by several papers [HKP$^+$03, Mar02, HK02, BNHK$^+$00, Kov04], but the edge coloring problem is only mentioned in

[BNHK+00] and [Kub96]. In [HK02, HKP+03] PTAS is given for the vertex coloring version of the problem in the case when the graph is a tree, partial k-tree, or a planar graph. One of their main tools is the decomposition of colors into layers of geometrically increasing size. This method will be used here as well. However, most of the other tools in [HK02, HKP+03] cannot be applied in our case, since those tools assume that the graph can be colored with a constant number of colors. In our case this is not necessarily true: if the maximum degree of the tree can be arbitrary, then the line graph of the tree can contain arbitrarily large cliques. On one hand, these large cliques make the tools developed for partial k-trees impossible or very difficult to apply. On the other hand, a large clique helps us in finding an approximate solution: since the sum of a large clique has to be very large in every coloring (it grows quadratically in the size of the clique), more errors can be tolerated in an approximate solution, and this gives us more elbow space in constructing a good approximation.

In Section 5.4.1 we introduce notations and give some preliminary results. The scaling property for trees is proved in Section 5.4.2. Section 5.4.3 gives a simple PTAS for the problem if the demand of every edge in the tree is bounded by a constant, while Section 5.4.4 deals with the case where the maximum degree of the tree is bounded by a constant. Finally, Section 5.4.5 combines the ideas presented in the preceding two sections to give a PTAS for the general case, where the demands and the degrees can be arbitrary.

In [Mar04f] the results of this section are generalized: the PTAS for trees is extended to partial k-trees and planar graphs. The approximation scheme for partial k-trees is very similar to the one presented here for trees, although some new ideas and techniques are also required. Using the layering technique of Baker [Bak94], the PTAS for partial k-trees can be used to obtain a PTAS for planar graphs. We state here these results without proof:

Theorem 5.4.1 ([Mar04f]). *For every $\epsilon > 0$ and every integer k, there is a linear time $(1+\epsilon)$-approximation algorithm for solving minimum sum edge multicoloring on partial k-trees.*

Theorem 5.4.2 ([Mar04f]). *For every $\epsilon > 0$, there is a linear time $(1+\epsilon)$-approximation algorithm for solving minimum sum edge multicoloring on planar graphs.*

5.4.1 Preliminaries

We extend the notion of finish time to a set E' of edges by defining $f_\Psi(E') = \sum_{e \in E'} f_\Psi(e)$. Given a graph G and a demand function $x(e)$ on the edges of G, the minimum sum that can be achieved is denoted by $\mathrm{OPT}(G, x)$, or by simply $\mathrm{OPT}(G)$, if the demand function is clear from the context.

For a minimization problem, algorithm A is an *α-approximation* algorithm if it always produces a solution with cost at most α times the optimum. A *polynomial time approximation scheme (PTAS)* is an algorithm that has a parameter ϵ such that for every $\epsilon > 0$ it produces an $(1+\epsilon)$-approximate solution, and the running time is polynomial in n for every fixed ϵ, e.g., $O(n^{1/\epsilon})$. A linear time PTAS runs in time $O(f(\epsilon)n)$ where f is an arbitrary function. When designing a PTAS, it can be assumed that ϵ is smaller than some fixed constant ϵ_0. In the following it is assumed that ϵ is sufficiently small and $1/\epsilon$ is an integer. Appendix A.2 contains some more background on approximation algorithms.

Henceforth the graph G is a rooted tree with root r. The root is assumed to be a node of degree one, the *root edge* is the edge incident to r. Every edge has an *upper node* (closer to r) and a *lower node* (farther from r). Edge f is a *child edge* of edge e if the upper node of f is the same as the lower node of e. In this case, edge e is the *parent edge* of edge f. A node is a *leaf node* if it has no children, and an edge is a *leaf edge* if its lower node is a leaf node. The subtree T_e consists of the edge e and the subtree rooted at the lower node of e.

A *top-down traversal* of the edges is an ordering of the edges in such a way that every edge appears later than its parent edge. Similarly, in a *bottom-up traversal* of the edges every edge appears before its parent. It is clear that such orderings exist and can be found in linear time.

Since the tree is a bipartite graph, every node has a *parity*, which is either 1 or 2, and neighboring nodes have different parity. Let the parity of an edge be the parity of its upper node. Thus if two edges have the same parity and they have a common node v, then v is the upper node of both edges.

If the tree has maximum degree Δ, then the edges can be colored with Δ colors such that adjacent edges have different colors. Fix such a coloring and let the *type* of an edge be its color in this coloring. In some of the algorithms, the leaf edges will be special, they are handled differently. Therefore in these cases we assign a type only to the non-leaf edges. Clearly, if every edge has at most D non-leaf child edges, then the non-leaf edges can be given a type from $1, 2, \ldots, D+1$ in such a way that adjacent edges have different type.

The following lemma bounds the number of colors required in a minimum sum multicoloring. In the following, the maximum demand in the instance is always denoted by p.

Lemma 5.4.3. *If G is a graph with maximum degree Δ and maximum demand p, then every optimum coloring of the minimum sum edge multicoloring problem uses at most $p(2\Delta-1)$ colors.*

Proof. Assume that an optimum coloring Ψ uses a color greater than $p(2\Delta-1)$ on the edge $e = uv$. Remove the colors from e. Since at most $\Delta - 1$ edges (other than e) are incident to u, with a demand of at most p each, at most $p(\Delta-1)$ colors are used on u. Similarly, there are at most $p(\Delta-1)$ colors used at v. Therefore there are at least p colors not greater than $p(2\Delta-1)$ that is used neither on u nor v. These p colors can be used to color the edge e. This will decrease the finish time of e, contradicting the optimality of Ψ. ■

If both the maximum degree of the tree and the maximum demand are bounded by a constant, then the problem can be solved in linear time. The idea is that there are only a constant number of possible color sets that can appear at each edge, hence using standard dynamic programming techniques, the optimum coloring can be found during a bottom-up traversal of the edges.

Theorem 5.4.4. *The minimum sum edge multicoloring problem for trees can be solved in $2^{O(p\Delta)} \cdot n$ time.*

Proof. We apply dynamic programming to solve the problem. Denote by $\mathcal{U}_k^n$ the set of all k element subsets of $\{1, 2, \ldots, n\}$. Let T_e be the subtree of the tree T whose root edge is e. Set $m := p(2\Delta-1)$. For every $e \in E(T)$ and $X \in \mathcal{U}^m_{x(e)}$, let $S(e, X)$ denote the value of the optimum sum in the subtree T_e with the further restriction that e receives the set X. Clearly, $\mathrm{OPT}(T, x) = \min_{X \in \mathcal{U}^m_{x(r)}} S(r, X)$, since by Lemma 5.4.3, the root edge r is colored by a set from $\mathcal{U}^m_{x(r)}$ in every optimum coloring.

We determine the values $S(e, X)$ following a bottom-up traversal of the edges. If T_e consists of only the edge e, then $S(e, X)$ is the highest color in X. Now assume that the child edges of e are $e_1, e_2, \ldots, e_k$, and for each $1 \le i \le k$, we have already computed a table containing the value of $S(e_i, Y)$ for every $Y \in \mathcal{U}^m_{x(e_i)}$. We would like to determine the value $S(e, X)$ for some set X. One way to do this is to choose k sets $X_i \in \mathcal{U}^m_{x(e_i)}$ $(1 \le i \le k)$ in every possible way. If the sets $X, X_1, X_2, \ldots, X_k$ are pairwise disjoint, then there is a coloring Ψ with $\Psi(e) = X$, $\Psi(e_i) = X_i$ and $f_\Psi(T_e) = \sum_{i=1}^k S(e_i, X_i) + \max_{c \in X} c$. Set $S(e, X)$ to the minimum of this sum for the best choice of the sets $X_1, \ldots, X_k$. It is easy to see that this is indeed the value given by the definition of $S(e, X)$ (by Lemma 5.4.3, every optimum coloring uses only the colors $1, \ldots, m$).

The method described above solves at most $|\mathcal{U}_p^m|$ subproblems $S(e, X)$ at each edge e. In each subproblem, $k \le \Delta - 1$ subsets X_i are chosen in every possible way, the number of combinations considered is at most $|\mathcal{U}_p^m|^{\Delta-1} = O(m^{p(\Delta-1)}) = 2^{O(p\Delta \log(p\Delta))}$. However, using dynamic programming once again, each subproblem $S(e, X)$ can be solved in $2^{O(p\Delta)}$ time. Denote by $T_{e,i}$ the union of the trees T_{e_1}, T_{e_2}, $\ldots$, T_{e_i} (the first i children of edge e). For every $Y \subseteq \{1, 2, \ldots, m\}$ and $1 \le i \le k$ denote by $P(i, Y)$ the minimum sum on $T_{e,i}$ with the restriction that exactly the colors in Y are used on the edges $e_1, \ldots, e_i$. Clearly, $P(1, Y) = S(e_1, Y)$. To calculate $P(i, Y)$ for some $i > 1$ notice that $P(i, Y)$ is the minimum of $S(i, X_i) + P(i-1, Y \setminus X_i)$, where the minimum is taken over all $x(e_i)$ size subsets X_i of Y. Finally, $S(e, X)$

can be easily determined by considering every set $Y \subseteq \{1, 2, \ldots, m\}$ disjoint from X, and selecting the one where $P(k, Y)$ is minimal.

At each edge we solve at most $2^m \cdot (\Delta - 1)$ subproblems $P(i, Y)$. To solve a subproblem $P(i, Y)$, we consider $|\mathcal{U}^m_{x(e_i)}| < 2^m$ different sets X_i. Therefore the total number of combinations considered at an edge is $2^{O(m)}$. The work to be done for each combination is polynomial in m, hence it is dominated by $2^{O(m)}$. The number of edges in the tree is $O(n)$, thus the total running time of the algorithm is $2^{O(m)} \cdot n = 2^{O(p\Delta)} \cdot n$. ■

On the other hand, we have seen in Theorem 5.3.1 that if only the demand is bounded, then the problem becomes NP-hard.

In the minimum sum multicoloring problem our goal is to minimize the sum of finish times, not to minimize the number of different colors used. Nevertheless, in Theorem 5.4.5 we show that the minimum number of colors required for coloring the edges of a tree can be determined by a simple formula. We also show that there is always an optimum solution where the color sets are relatively simple. This result will be used by the approximation algorithm presented in Section 5.4.4 to solve certain subproblems.

Theorem 5.4.5. *Let T be a tree and let $C = \max_{v \in V(T)} \sum_{e \ni v} x(e)$. Every coloring of T uses at least C colors, and one can find in linear time a coloring Ψ using C colors where each $\Psi(e)$ consists of at most two intervals of colors. Moreover, if each $x(e)$ is an integer multiple of some integer q, then we can find such a Ψ where the intervals in each $\Psi(e)$ are of the form $[qi_1 + 1, qi_2]$ for some integers i_1 and i_2.*

Proof. It is clear that at least C colors are required in every coloring: there is a vertex v such that the edges incident to v require C different colors. A coloring Ψ satisfying the requirements can be constructed by a simple greedy algorithm. Call a set of colors $S \subseteq [1, C]$ a *circular interval* if it is either an interval $[a, b]$ or the union of two intervals $[1, a] \cup [b, C]$. The algorithm presented below assigns a circular interval of colors to each edge, therefore each $\Psi(e)$ consists of one or two intervals.

Consider a top-down traversal of the edges. The edges are colored in a greedy manner following this ordering. After each step of the algorithm, the coloring defined so far satisfies the following invariant condition: for every node v, the set of colors used by the edges incident to v forms a circular interval of $[1, C]$.

At the start of the algorithm, we assign the set $[1, x(r)]$ to the root edge r. When an edge e is visited during the traversal, some of the edges incident to the upper node v of e are already colored, and none of the edges incident to the lower node u of e has a coloring yet. By assumption, the colors used by the edges incident to v form a circular interval S. Clearly, the size of $[1, C] \setminus S$ is at least $x(e)$, otherwise $\sum_{e \ni v} x(e)$ would be strictly greater than C. We can assign to edge e a circular interval S' of size $x(e)$ such that S and S' are disjoint, and $S \cup S'$ is also a circular interval. Thus the set of colors used at v remains a circular interval. Because of the top-down traversal, the set of colors used at u is exactly S', a circular interval, hence the invariant condition remains valid and the algorithm can proceed with the next edge. Moreover, if every demand is an integer multiple of q, then it can be shown by induction that every edge receives a circular interval $\Psi(e)$ such that the one or two intervals in $\Psi(e)$ are of the form $[qi_1 + 1, qi_2]$. ■

The fact that for trees a greedy algorithm can minimize the number of colors used was observed in [Kub96]. However, in our applications it will be important that the color sets have the special form described in Theorem 5.4.5.

5.4.2 Scaling and rounding

Consider an instance of minimum sum edge multicoloring: let G be a graph, and let $x(e)$ be an arbitrary demand function on the edges of G. Multiply the demand of every edge by an integer q, that is, consider

the demand function $x'(e) = q \cdot x(e)$. The first observation is that this operation increases the minimum sum at most by a factor of q, that is,

$$\mathrm{OPT}(G, x') \leq q \cdot \mathrm{OPT}(G, x). \tag{5.2}$$

To see this, take an optimum coloring Ψ of (G, x), and replace every color by q consecutive colors: for every $c \in \Psi(e)$, add $\{(c-1)q+1, (c-1)q+2, \ldots, cq\}$ to $\Phi(e)$. Clearly, coloring Φ satisfies the demand x' on every edge, and the finish time of every edge in Φ is exactly q times larger than in Ψ. Therefore the sum of Φ is exactly q times larger than the optimum sum of (G, x), and (5.2) follows.

We mention it without proof that one can construct a bipartite graph G, and choose x, q in such a way that (5.2) holds with strict inequality. The aim of this section is to show that if G is a tree, then there is always equality in (5.2):

Theorem 5.4.6. *For every tree $T(V, E)$, demand function x and integer q, if $x'(e) = q \cdot x(e)$ for every $e \in E$, then $\mathrm{OPT}(T, x') = q \cdot \mathrm{OPT}(T, x)$ holds.*

Before proving Theorem 5.4.6, we have to make some preparations. The following problem is the weighted version of multicoloring (this problem is studied in [Jan00, Jan97] under the name Generalized Optimum Cost Chromatic Partition). Every vertex has a cost function (thus the same color can have different costs at different vertices), and the goal is to minimize the total cost of the colors used in the coloring. As in the case of other coloring problems, we consider here the edge coloring version:

Generalized Minimum Cost Multicoloring

Input: A graph $G(V, E)$, a *demand* function x: $E \to \mathbb{N}$, a set of available colors $\mathcal{C} = \{1, 2, \ldots, C\}$, and a list of costs $c_{e,i}$ (for every $e \in E$, $i \in \mathcal{C}$).

Output: A *multicoloring* Ψ: $E \to 2^C$ such that

- $\Psi(e_1) \cap \Psi(e_2) = \emptyset$ if e_1 and e_2 are adjacent in G, and
- $|\Psi(e)| = x(e)$.

Goal: Minimize the total cost $\sum_{e \in E} \sum_{i \in \Psi(e)} c_{e,i}$.

This problem can be formulated as an integer linear programming problem as follows. If the value of the 0-1 variable $y_{e,i}$ is 1, then we assign color i to edge e. It is easy to verify that the integer solutions of the following linear program correspond to the proper colorings of the graph:

$$\text{minimize} \quad \sum_{e \in E} \sum_{i=1}^{C} c_{e,i} y_{e,i}$$

$$\text{s. t.}$$

$$y_{e,i} \geq 0 \quad \forall e \in E,\ 1 \leq i \leq C \tag{5.3}$$

$$\sum_{e \ni v} y_{e,i} \leq 1 \quad \forall v \in V,\ 1 \leq i \leq C \tag{5.4}$$

$$\sum_{i=1}^{C} y_{e,i} \geq x(e) \qquad \forall e \in E \tag{5.5}$$

$$y_{e,i} \in \{0, 1\} \quad \forall e \in E,\ 1 \leq i \leq C \tag{5.6}$$

The inequalities (5.4) express the requirement that a color i can appear at most once on the edges incident to v, while inequalities (5.5) ensure that edge e receives at least $x(e)$ colors. The cost of an integer solution equals the cost of the corresponding coloring. In general, the linear programming relaxation (dropping (5.6) from the program) does not have an integer optimum solution, but if the graph is a tree, then there is always an integer optimum:

Lemma 5.4.7. *For every tree T with arbitrary demand function x and costs $c_{e,i}$ the linear program has an integer optimum solution.*

Proof. We show that the coefficient matrix of the linear program is a network matrix, hence it is totally unimodular. Since the right-hand side of the linear program is an integer vector, thus the lemma follows from the well-known properties of totally unimodular matrices (cf. [Sch03]).

Recall the definition of network matrices. Let D be a directed graph and T be a spanning tree over the same vertex set V. Denote by n and m the number of edges of T and D, respectively. Direct the edges of T arbitrarily. Consider an $n \times m$ matrix N whose rows correspond to the edges of T and columns correspond to the edges of D. Every directed edge e in D determines a unique path in the tree. If edge f of T lies on this path and its orientation agrees with the directed path, then let the element of M in row f and column e be 1; if its orientation is opposite, then let the element be -1. If f does not lie on the path determined by e, then the element is zero. A matrix M that arises in such a way from some T and D is called a *network matrix*. It is well known that every network matrix is totally unimodular (cf. [Sch03]).

The constraints (5.3) can be left out of the program: if an $n \times m$ matrix is totally unimodular, then it remains totally unimodular after appending to it an $m \times m$ unit matrix. Denote the inequalities in (5.4) by $d_{v,i}$ ($v \in V$, $1 \leq i \leq C$) and those in (5.5) by d_e. Let V_1 and V_2 be the two bipartition classes of T. Direct the edges of T from V_1 to V_2 and identify the directed edges with the constraints d_e. Connect C new vertices v_i ($1 \leq i \leq C$) to every vertex v. Identify the new edges with the constraints $d_{v,i}$ and direct them away from v if $v \in V_1$, and to v if $v \in V_2$. Call this tree T'. The directed graph D is defined as follows: if $e = uv$ ($u \in V_1$, $v \in V_2$) is an edge in T, then add the edges $y_{e,i} = \overrightarrow{v_i u_i}$ ($1 \leq i \leq C$) to D. Now it can be verified that the network matrix corresponding to T' and D is the coefficient matrix of the linear program: the unique path corresponding to edge $y_{e,i} = \overrightarrow{v_i u_i}$ contains the edges $d_{v,i}, d_e, d_{u,i}$, and the variable $y_{e,i}$ appears precisely in these inequalities. ■

Proof (of Theorem 5.4.6). Given a coloring Ψ realizing the optimum $\mathrm{OPT}(T, x')$, we construct a coloring Φ that satisfies the demand function x and has sum at most $\mathrm{OPT}(T, x')/q$. Define the following cost function:

$$c_{e,i} = \begin{cases} 0 & i < \lceil f_\Psi(e)/q \rceil \\ 1 & i = \lceil f_\Psi(e)/q \rceil \\ 2|E| & i > \lceil f_\Psi(e)/q \rceil \end{cases}$$

Consider the generalized minimum cost multicoloring problem on the edges of T, with demand $x(e)$ and color costs $c_{e,i}$. Let C, the number of colors, be an integer larger then the total demand of the tree T. It is clear that the linear program given by inequalities (5.3)–(5.5) always has a feasible solution: since C is large enough, the demands can be satisfied even if every color is used at most once. By Lemma 5.4.7, this program has an integer optimum solution with costs $c_{e,i}$, let $y_{e,i}$ be such a solution. Clearly every variable is 0 or 1. Define coloring Φ with $i \in \Phi(e)$ if and only $y_{e,i} = 1$. Replace every color in the coloring Φ with a sequence of q colors to obtain a coloring Φ', that is, if $i \in \Phi(e)$, then add $\{(i-1)q+1, (i-1)q+2, \ldots, iq\}$ to $\Phi'(e)$. Clearly, $f_{\Phi'}(T) = q \cdot f_\Phi(T)$, thus if it can be shown that $f_{\Phi'}(T) \leq f_\Psi(T) = \mathrm{OPT}(T, x')$, then $\mathrm{OPT}(T, x) \leq f_\Phi(T) \leq \mathrm{OPT}(T, x')/q$ and Theorem 5.4.6 follows.

Let $z_{e,i}$ be $|\Psi(e) \cap \{(i-1)q+1, (i-1)q+2, \ldots, iq\}|/q$. It can be easily verified that this is a feasible solution of the linear program. Furthermore, the cost of this solution is strictly less than $2|E|$ since, by definition, $z_{e,i}$ is 0 if $i > \lceil f_\Psi(e)/q \rceil$. Therefore the optimum integral solution $y_{e,i}$ has cost strictly less than $2|E|$, which implies that $y_{e,i} = 0$ for $i > \lceil f_\Psi(e)/q \rceil$, and $f_\Phi(e) \leq \lceil f_\Psi(e)/q \rceil$.

Let $c_\Phi(e) = \sum_{i=1}^{C} c_{e,i} y_{e,i}$ and $c_\Psi(e) = \sum_{i=1}^{C} c_{e,i} z_{e,i}$. We show that for every edge e, $f_\Phi(e) \leq f_\Psi(e)/q + c_\Phi(e) - c_\Psi(e)$, or equivalently $f_{\Phi'}(e) \leq f_\Psi(e) + q(c_\Phi(e) - c_\Psi(e))$. If the latter inequality holds, then summing for every $e \in E$ gives $f_{\Phi'}(E) \leq f_\Psi(E) + q(\sum_{e \in E} c_\Phi(e) - \sum_{e \in E} c_\Psi(e))$. Since $y_{e,i}$ is an optimum solution of the linear program with costs $c_{e,i}$, thus $\sum_{e \in E} c_\Phi(e) \leq \sum_{e \in E} c_\Psi(e)$, which implies $f_{\Phi'}(E) \leq f_\Psi(E)$, proving the theorem.

There are two cases to consider: (a) $f_\Phi(e) = \lceil f_\Psi(e)/q \rceil$ and (b) $f_\Phi(e) < \lceil f_\Psi(e)/q \rceil$ (we have seen that $f_\Phi(e) \leq \lceil f_\Psi(e)/q \rceil$). Since $\Psi(e)$ contains at most $f_\Psi(e) - (\lceil f_\Psi(e)/q \rceil q - q)$ colors above $\lceil f_\Psi(e)/q \rceil q - q$, thus $q \cdot c_\Psi(e) \leq f_\Psi(e) - (\lceil f_\Psi(e)/q \rceil q - q)$ and $f_\Psi(e)/q - c_\psi(e) \geq \lceil f_\Psi(e)/q \rceil - 1$. If (a) holds, then $c_\Phi(e) = 1$, thus $f_\Psi(e)/q + c_\Phi(e) - c_\Psi(e) \geq \lceil f_\Psi(e)/q \rceil = f_\Phi(e)$, as required. In case (b), $c_\Phi(e) = 0$, therefore $f_\Psi(e)/q + c_\Phi(e) - c_\Psi(e) \geq \lceil f_\Psi(e)/q \rceil - 1 \geq f_\Phi(e)$, what we had to prove. ■

In Section 5.3, we have shown that the preemptive minimum sum edge coloring problem is NP-hard for trees even if every demand is 1 or 2. However, it becomes polynomial time solvable if every demand is 2, or more generally, if every edge has the same demand. By Theorem 5.4.6, the case where every edge has the same demand can be reduced to the case where every edge has unit demand, which is polynomial time solvable [GK00, Sal03, ZN04].

Corollary 5.4.8. *The minimum sum edge multicoloring problem can be solved in polynomial time in trees if every edge has the same demand.* ■

The following lemma is another corollary of Theorem 5.4.6: if the demand of every edge is increased to at most λ times the original demand, then the optimum increases by at most a factor of λ. If λ is an integer, then this is trivial (replace every color in the optimum coloring by λ consecutive colors), but the lemma states that in trees this is true even if λ is not an integer. This observation will be used in Section 5.4.5.

Lemma 5.4.9. *Let (T, x) be an instance of minimum sum edge multicoloring and let x' be a demand function with $x'(e) \leq \lambda \cdot x(e)$ for every edge e. If T is a tree, then* $\mathrm{OPT}(T, x') \leq \lambda \cdot \mathrm{OPT}(T, x)$.

Proof. Assume that λ is rational and $\lambda = a/b$ where a and b are integers. Let $x_2(e) = a \cdot x(e)$. By Theorem 5.4.6, $\mathrm{OPT}(T, x_2) = a \cdot \mathrm{OPT}(T, x)$. Round down x_2 to the nearest integer multiple of b, denote by x_3 the obtained demand function. Let $x_4(e) = x_3(e)/b = \lfloor ax(e)/b \rfloor \geq \lfloor x'(e) \rfloor = x'(e)$. By Theorem 5.4.6, $\mathrm{OPT}(T, x_4) = \mathrm{OPT}(T, x_3)/b \leq \mathrm{OPT}(T, x_2)/b = (a/b)\mathrm{OPT}(T, x) = \lambda \cdot \mathrm{OPT}(T, x)$. Thus $x_4(e) \geq x'(e)$ implies $\mathrm{OPT}(T, x') \leq \mathrm{OPT}(T, x_4) \leq \lambda \cdot \mathrm{OPT}(T, x)$. ■

5.4.3 Bounded demand

In Section 5.3 we have shown that the minimum sum edge multicoloring problem is NP-hard in trees, even if every demand is at most 2. Therefore we cannot hope to find a polynomial time algorithm in the special case where every demand is at most p, for a fixed constant $p > 1$. However, the problem admits a linear time PTAS:

Theorem 5.4.10. *For every $\epsilon_0 > 0$, there is a $2^{O(p^2/\epsilon_0^2)} \cdot n$ time algorithm that gives a $(1+\epsilon_0)$-approximate solution to the minimum sum edge multicoloring problem in trees.*

Proof. Let $\epsilon := \epsilon_0/4$. A node with at least $D := 4p/\epsilon^2$ children will be called a *high-degree node*. We partition the edge set of the tree T into connected subgraphs. If v is a high-degree node, then the child edges of v form a class of the partition, which will be called the *high-degree star* at v. After deleting the edges of the high-degree stars from the tree, some connected *low-degree components* (with maximum degree D) remain, every such component is a class of the defined partition (see Figure 5.11). The tree defines a tree structure on the components in a natural way, all but one component has a parent component. The parent of a low-degree component is always a high-degree star.

First we color each component of the partition separately, later we show how these colorings can be combined to obtain a coloring of the whole tree. Using the algorithm of Theorem 5.4.4, a low-degree component having n' edges and maximum degree D can be colored optimally in $2^{O(pD)} \cdot n' = 2^{O(p^2/\epsilon^2)} \cdot n'$ time, thus coloring all these components requires $2^{O(p^2/\epsilon^2)} \cdot n$ time.

The optimum coloring of a high-degree star is obtained by coloring the edges non-preemptively one after the other, in the order of increasing demand size. We color the high-degree stars in a way that

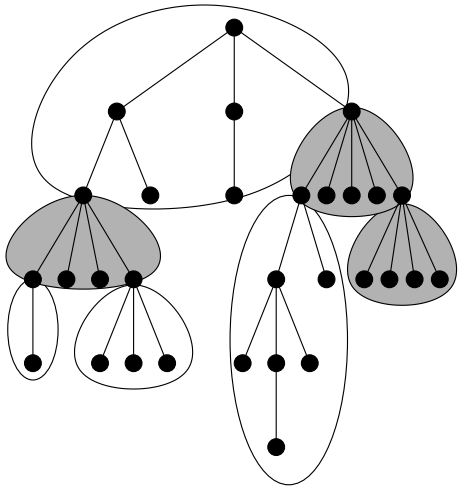


Figure 5.11: Partitioning the tree into high-degree stars (shaded regions) and low-degree components ($D = 4$).

is slightly worse than this optimum coloring. Assume that the parent component of the star S_v at v already has a coloring, this coloring assigns a set of colors to the edge e that connects v to its parent. We color the edges in the star using only colors greater than $B := p/\epsilon$. The edges are colored in the order of increasing demand size, starting with the color $B + 1$, skipping the (at most p) colors used by e, if necessary. Compared to the optimum coloring given above, the finish time of every edge in the star is delayed by at most $B + p$. Therefore if v has d children, then the sum of the star in this coloring is greater than the optimum by at most $d(B + p) = d(p/\epsilon + p) = dp(1/\epsilon + 1) \leq 2dp/\epsilon \leq dD\epsilon/2 \leq \epsilon d^2/2$. On the other hand, the sum of these d edges is at least $\sum_{i=1}^{d} i > d^2/2$ in every optimum coloring. Therefore,

$$\epsilon \cdot \mathrm{OPT}(S_v) \geq \epsilon d^2/2$$

and the sum of the coloring given to S_v is not greater than $(1 + \epsilon)\mathrm{OPT}(S_v)$.

Putting together the colorings of the components, we obtain a coloring Ψ, which is not necessarily a proper coloring: there might be conflicts between a low-degree component and its parent component, a high-degree star. The second phase of the algorithm resolves these conflicts with only a $1 + \epsilon$ factor increase in the total sum. First we modify the coloring Ψ such that every color greater than B is shifted up by p. This increases by p the finish time of those edges that have finish time greater than B, and leaves the edges with finish time at most B unchanged. Therefore the finish time of every node is at most $(B + p)/B \leq 1 + \epsilon$ times the original.

Assume that $e = vu$ is an edge in the high-degree star at v that conflicts with some of the edges incident to u, the lower node of e. The conflict will be solved the following way: instead of using the (at most p) colors assigned to e, the child edges of e will use the p colors $\{B + 1, \ldots, B + p\}$ (in any order). This will not increase the finish time of any of the edges since the colors used by e, and therefore the conflicting colors, are all greater than $B + p$. To see that this modification does not introduce additional conflicts, notice first that conflicts can appear only between edges that received new colors, since the colors $\{B + 1, \ldots, B + p\}$ were not used by any of the edges. Every recolored edge is in a low-degree component, and its parent edge is in a high-degree star. Therefore if two recolored edge are adjacent, then they have a common parent edge e. But this means that there cannot be conflict between the two edges, since every color in $\{B + 1, \ldots, B + p\}$ was given to at most one child of e.

The first phase of the algorithm produced a (not necessarily proper) coloring having sum at most $(1+\epsilon)\mathrm{OPT}(T)$, since the sum of the coloring was at most $(1+\epsilon)$ times the optimum in each component of the partition. The second phase resolved the conflicts and increased the sum by a factor of at most $1+\epsilon$, thus the final coloring has sum at most $(1+\epsilon)^2\mathrm{OPT}(T) \leq (1+\epsilon_0)\mathrm{OPT}(T)$. The running time of the algorithm is dominated by the first phase, the second phase can be done in $O(np)$ time. ■

5.4.4 Bounded degree

If a tree T has maximum degree Δ, then the line graph of T is a partial $(\Delta-1)$-tree. Halldórsson and Kortsarz [HK02] gave a PTAS with running time $n^{O(k^2/\epsilon^5)}$ for minimum sum multicoloring the vertices of partial k-trees, therefore there is a PTAS for minimum sum edge multicoloring in bounded degree trees as well. However, the method can be made simpler and more efficient in line graphs of trees. In this section we present a linear time PTAS for minimum sum edge multicoloring in bounded degree trees, which makes use of the special structure of trees. Furthermore, our algorithm works even in the more general class of *almost bounded degree trees*: in trees that become bounded degree after deleting the degree one nodes. Equivalently, we can say that a tree is an almost bounded degree tree if every node has at most a bounded number of non-leaf child edges.

Most of the ideas presented in this section are taken from [HK02], with appropriate modifications. In Section 5.4.5 a PTAS is given for general trees, which uses the result in this section as a subroutine.

Layers and Zones

An important idea of the approximation schemes given in [HK02, HKP+03] is to divide the color spectrum into geometrically increasing layers, and to solve the problem in these layers separately. We use a similar method for the minimum sum edge multicoloring problem in bounded degree trees (Theorem 5.4.14) and general trees (Theorem 5.4.15).

For some $\epsilon > 0$ and integer $\ell \geq 0$, the (ϵ,ℓ)-*decomposition* divides the infinite set of colors into *layers* $L_0, L_1, \ldots$ and *zones* $Z_0, Z_1, \ldots, Z_\ell$. The layers are of geometrically increasing size: layer L_i contains the range of colors from q_i to $q_{i+1}-1$, where $q_i = \lfloor (1+\epsilon)^i \rfloor$. If $q_i = q_{i+1}$, then layer L_i is empty. Denote by $Q_i = |L_i| = q_{i+1} - q_i$ the size of the ith layer. The total size of layers L_0, L_1, …, L_i is $q_{i+1}-1$. Later we will use that $(1+2\epsilon)q_i \geq q_{i+1}-1$:

$$(1+2\epsilon)q_i > (1+\epsilon)((1+\epsilon)^i - 1) + \epsilon q_i = (1+\epsilon)^{i+1} - 1 - \epsilon + \epsilon q_i \geq (1+\epsilon)^{i+1} - 1 \geq q_{i+1} - 1. \qquad (5.7)$$

That is, if we replace a color from layer L_i with another color from L_i, then the new color is at most $(1+2\epsilon)$ times larger than the original.

Each layer is divided into a main block and an extra block. The extra block is further divided into extra segments (see Figure 5.12). Layer L_i is divided into two parts: the first $\frac{1}{1+\epsilon\ell}Q_i$ colors form the *main block* of layer L_i and the remaining $\frac{\epsilon\ell}{1+\epsilon\ell}Q_i$ colors the *extra block*. The main block of layer L_i is denoted by $\overline{L}_i$. The union of the main block of every layer L_i is the *main zone* Z_0. Divide the extra block of every layer L_i into ℓ equal parts: these are the ℓ *extra segments* of L_i. The union of the jth extra segment of every layer L_i forms the jth *extra zone* Z_j. Each extra zone contains $\frac{\epsilon}{1+\epsilon\ell}Q_i$ colors from layer L_i.

Rounding problems will be handled as follows. Layer i is divided such that the first $g_{i,0}$ colors are assigned to the main zone Z_0, and extra zone Z_j receives $g_{i,j}$ colors. We show that the values $g_{i,j}$ can be determined in such a way that the resulting zones approximate the "ideal" case where the main zone contains the $\frac{1}{1+\epsilon\ell}$ part of the color spectrum, and each extra zone contains the $\frac{\epsilon}{1+\epsilon\ell}$ part of the colors.

First set $g_{i,0} = \lceil \frac{1}{1+\epsilon\ell}(q_{i+1}-1) \rceil - \lceil \frac{1}{1+\epsilon\ell}(q_i - 1) \rceil \leq \lceil \frac{1}{1+\epsilon\ell} Q_i \rceil$. The inequality follows from $\lceil a \rceil + \lceil b \rceil \geq \lceil a+b \rceil$. This ensures that $\sum_{k=0}^{i} g_{k,0} = \lceil \frac{1}{1+\epsilon\ell}(q_{i+1}-1) \rceil$. Now there remains $g_i = Q_i - g_{i,0}$ colors for the extra zones in layer L_i. The following lemma shows that these colors can be evenly divided among the ℓ layers:

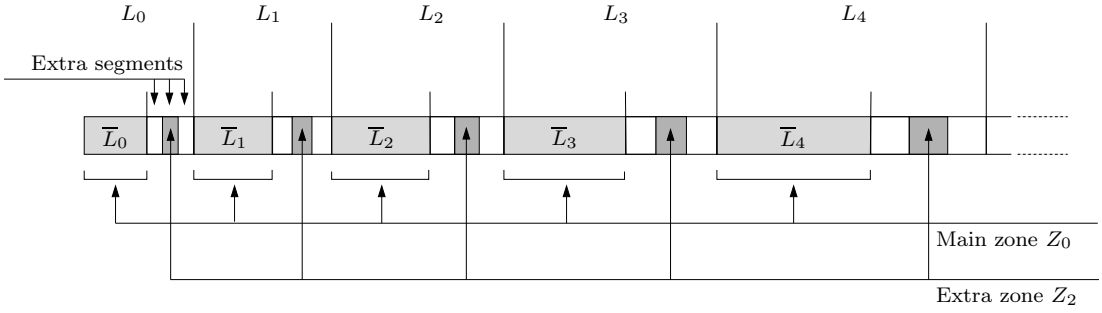


Figure 5.12: The decomposition of the colors into layers ($\ell = 3$)

Lemma 5.4.11. *If ℓ and g_i ($0 \le i \le n$) are nonnegative integers, then there are nonnegative integers $g_{i,j}$ ($0 \le i \le n$, $1 \le j \le \ell$) such that for every i and j*

$$\sum_{k=0}^{i} g_{k,j} \ge \left\lfloor \frac{1}{\ell} \sum_{k=0}^{i} g_k \right\rfloor \tag{5.8}$$

and

$$\sum_{j=1}^{\ell} g_{i,j} \le g_i \tag{5.9}$$

hold. Moreover, if $\sum_{k=0}^{i} g_k$ can be calculated with a constant number of arithmetic operations, then each $g_{i,j}$ can be also calculated with a constant number of arithmetic operations.

Proof. We calculate $g_{i,j}$ by determining the values $G_{i,j} = \sum_{k=0}^{i} g_{k,j}$, then $g_{i,j}$ can be obtained as $g_{i,j} = G_{i,j} - G_{i-1,j}$. Let

$$G_{i,j} = \begin{cases} \lceil \frac{1}{\ell} \sum_{k=0}^{i} g_k \rceil & \text{if } j \le \sum_{k=0}^{i} g_k - \ell \lfloor \frac{1}{\ell} \sum_{k=0}^{i} g_k \rfloor, \\ \lfloor \frac{1}{\ell} \sum_{k=0}^{i} g_k \rfloor & \text{otherwise.} \end{cases}$$

Clearly, $\sum_{j=1}^{\ell} G_{i,j} = \sum_{k=0}^{i} g_k$. It is clear that (5.8) holds, since $G_{i,j} \ge \lfloor \frac{1}{\ell} \sum_{k=0}^{i} g_k \rfloor$. Furthermore, (5.9) also holds:

$$\sum_{j=1}^{\ell} g_{i,j} = \sum_{j=1}^{\ell} (G_{i,j} - G_{i-1,j}) = \sum_{k=0}^{i} g_k - \sum_{k=0}^{i-1} g_k = g_i.$$

Each $G_{i,j}$ can be calculated from $\sum_{k=0}^{i} g_k$ by a constant number of arithmetic operations, and this is true also for $g_{i,j} = G_{i,j} - G_{i-1,j}$, hence the claim of the lemma follows. ■

We will need the following properties of the defined zones:

Lemma 5.4.12. *For given ℓ and $\epsilon \le \frac{1}{2\ell}$, one can calculate values $g_{i,j}$ such that the resulting (ϵ, ℓ)-decomposition of the colors has the following properties:*

(a) For every $c \ge 1$, zone Z_0 contains at least c colors not greater than $\lfloor (1 + \epsilon\ell)c \rfloor$.

(b) For every $c \ge 1$ and $1 \le j \le \ell$, zone Z_j contains at least c colors not greater than $2/\epsilon \cdot c$.

Moreover, each value $g_{i,j}$ can be calculated using a constant number of arithmetic operations.

Proof. Consider the values $g_{i,j}$ given by Lemma 5.4.11. To verify property (a), notice that for every $d \geq 1$, there are at least $\lceil \frac{1}{1+\epsilon\ell} d \rceil$ colors in the main zone not greater than d. Indeed, if $d = q_{i+1} - 1$ (d is the last color of layer L_i), then this follows from the way $g_{i,0}$ was defined, otherwise it follows from the fact that the main zone uses the first $g_{i,0}$ colors of layer L_i, hence if it is true for $d = q_i - 1$ and $d = q_{i+1} - 1$, then it is true for every value in between. Thus there are at least $\lceil \frac{1}{1+\epsilon\ell} \lfloor (1+\epsilon\ell)c \rfloor \rceil \geq c$ colors below $\lfloor (1+\epsilon\ell)c \rfloor$.

To verify property (b), assume that $q_i - 1 < (1+\epsilon\ell)/\epsilon \cdot c \leq q_{i+1} - 1$ for some i. Since $(1+\epsilon\ell)/\epsilon \cdot c$ is greater than $1/\epsilon$, hence $(1+2\epsilon)(1+\epsilon\ell)/\epsilon \cdot c \geq (1+\epsilon\ell)/\epsilon \cdot c + 2 \geq q_i + 1 > (1+\epsilon)^i$. Multiplying by $1+\epsilon$ we get $(1+\epsilon)(1+2\epsilon)(1+\epsilon\ell)/\epsilon \cdot c \geq (1+\epsilon)^{i+1} > q_{i+1} - 1$. If $\epsilon \leq \frac{1}{2\ell}$ is sufficiently small, then $q_{i+1} - 1 \leq 2/\epsilon \cdot c$ follows. We use Lemma 5.4.11 to calculate the number of colors in the first i layers (i.e., up to color $q_{i+1} - 1$) that belong to zone Z_j:

$$\sum_{k=0}^{i} g_{k,j} \geq \left\lfloor \frac{1}{\ell} \sum_{k=0}^{i} g_k \right\rfloor = \left\lfloor \frac{1}{\ell} \sum_{k=0}^{i} (Q_k - g_{k,0}) \right\rfloor = \left\lfloor \frac{1}{\ell} \left(\sum_{k=0}^{i} Q_k - \left\lceil \sum_{k=0}^{i} \frac{1}{1+\epsilon\ell} Q_k \right\rceil \right) \right\rfloor$$
$$= \left\lfloor \frac{1}{\ell} \left\lfloor \sum_{k=0}^{i} \frac{\epsilon\ell}{1+\epsilon\ell} Q_k \right\rfloor \right\rfloor \geq \left\lfloor \frac{1}{\ell} \cdot \ell \left\lfloor \sum_{k=0}^{i} \frac{\epsilon}{1+\epsilon\ell} Q_k \right\rfloor \right\rfloor = \left\lfloor \sum_{k=0}^{i} \frac{\epsilon}{1+\epsilon\ell} Q_k \right\rfloor$$
$$= \left\lfloor \frac{\epsilon}{1+\epsilon\ell} (q_{i+1} - 1) \right\rfloor \geq \left\lfloor \frac{\epsilon}{1+\epsilon\ell} \cdot (1+\epsilon\ell)/\epsilon \cdot c \right\rfloor = c$$

Therefore there are at least c colors in zone Z_j not greater than $q_{i+1} - 1 \leq 2/\epsilon \cdot c$, proving property (b). ■

Given a multicoloring Ψ, the operation (ϵ, ℓ)-*augmentation* creates a multicoloring Φ as follows. Consider the (ϵ, ℓ)-decomposition of the colors, and if $\Psi(e)$ contains color c, then let $\Phi(e)$ contain instead the cth color from the main zone Z_0. By Lemma 5.4.12a, $f_\Phi(e) \leq \lfloor (1+\epsilon\ell) f_\Psi(e) \rfloor$, thus this operation increases the sum by at most a factor of $(1+\epsilon\ell)$. In Φ only the colors of the main zone are used.

PTAS for Bounded Degree Trees

The polynomial time algorithm of Theorem 5.4.4 was based on the observation that we have to consider only a constant number of different colorings at each edge if both the demand and the maximum degree are bounded. In general, however, the number of different color sets that can be assigned to an edge is exponentional in the demand. The main idea of the PTAS in [HK02] for vertex coloring partial k-trees is that one can select a polynomial number of color sets for each vertex in such a way that there is a good approximate coloring using only these sets. This gives a PTAS since the best coloring that uses only the selected sets can be found in polynomial time with standard dynamic programming techniques.

Here we also follow this path: Lemma 5.4.13 shows that one can find a good approximate coloring by considering only a constant number of different color sets at each edge. This results in a linear time PTAS for the problem.

Lemma 5.4.13. *For $\epsilon > 0$, define $b_{i,j} = q_i + j\lceil \epsilon^2 Q_i / 8 \rceil$. If each vertex of the tree T has at most D non-leaf child edges and $\epsilon \leq \frac{1}{3D}$, then it has a $(1+3D\epsilon)$-approximate coloring Ψ with the following properties:*

1. *In the (ϵ, D)-decomposition of the colors, if e is a non-leaf edge, then $\Psi(e)$ contains colors from the main zone only between $\epsilon x(e)/4$ and $2x(e)/\epsilon$.*
2. *If e is a non-leaf edge of type k, then $\Psi(e)$ contains the first t_e colors from extra zone Z_k (for some t_e), and it does not contain colors from the other extra zones.*
3. *If e is a leaf edge, then $\Psi(e)$ contains colors only from the main zone.*
4. *If e is a non-leaf edge, then $\Psi(e)$ contains at most two continuous intervals of colors from the main block of each layer, and the intervals in layer L_i are of the form $[b_{i,j_1}, b_{i,j_2} - 1]$ for some j_1 and j_2.*

Proof. Let Φ be an optimum solution, and let Ψ be the result of an $(\epsilon, D+1)$-augmentation on Φ. By Lemma 5.4.12a, $f_\Psi(e) \le (1+(D+1)\epsilon) f_\Phi(e)$ for every e (note that we assumed $\epsilon \le \frac{1}{3D} < \frac{1}{2(D+1)}$).

If $f_\Psi(e) > 2x(e)/\epsilon$ for a non-leaf edge e of type j, then modify $\Psi(e)$ to be the first $x(e)$ colors of zone Z_j. By Lemma 5.4.12b, Z_j contains at least $x(e)$ colors not greater than $2x(e)/\epsilon$. Therefore the $x(e)$ colors assigned to e are not greater than $2x(e)/\epsilon$, hence $f_\Psi(e) \le (1+(D+1)\epsilon) f_\Phi(e)$. In this case requirements 2 and 4 are automatically satisfied for e, thus there is nothing else to do with this edge.

If $\Psi(e)$ contains colors in the main zone below $\epsilon x(e)/4$, then delete these colors and let $\Psi(e)$ contain instead the first $\epsilon x(e)/4$ colors from zone Z_j. There are at least $\epsilon x(e)/4$ colors in Z_i below $2/\epsilon \cdot \epsilon x(e)/4 = x(e)/2$. Since the finish time of e is at least $x(e)$, hence this modification does not increase the finish time of e. Therefore Ψ satisfies the first three properties of the lemma.

Finally, we make Ψ satisfy the fourth requirement as well. For each non-leaf edge e, let $x_i(e)$ be $|\Psi(e) \cap \overline{L}_i|$ rounded down to the next integer multiple of $\lceil \epsilon^2 Q_i/8 \rceil$. If we use x_i as a demand function on the non-leaf edges of the tree, then there is multicoloring satisfying x_i that uses at most $|\overline{L}_i|$ colors: $\Psi(e) \cap \overline{L}_i$ is such a coloring. Therefore by Theorem 5.4.5, there is a multicoloring Ψ_i that uses at most $|\overline{L}_i|$ colors, satisfies x_i, and where each $\Psi(e)$ consists of at most two intervals of the form $[1+j_1\lceil \epsilon^2 Q_i/8 \rceil, j_2 \lceil \epsilon^2 Q_i/8 \rceil]$ for some j_1, j_2. Modify coloring Ψ: let Ψ_i determine how the colors are assigned in the main block of layer i. Now the third requirement is satisfied, but it is possible that Ψ assigns less than $x(e)$ colors to an edge. We can lose at most $\lceil \epsilon^2 Q_i/8 \rceil - 1 < \epsilon^2 Q_i/8$ colors in layer i, hence we lose at most the $\epsilon^2/8$ part of each layer. Assume that the highest color of $\Psi(e)$ is in layer L_i. Since $\Psi(e)$ contains colors only up to $2x(e)/\epsilon$, hence the last color of layer L_i is less than $2(1+2\epsilon)x(e)/\epsilon \le 4x(e)/\epsilon$ (Inequality (5.7)). Thus we lose only at most $\epsilon^2/8 \cdot 4x(e)/\epsilon = \epsilon x(e)/2$ colors. If non-leaf edge e is of type j, then we use extra zone Z_j to replace the lost colors. So far, edge e uses at most $\epsilon x(e)/2$ colors from Z_j (previous paragraph), hence there is still place for at least $\epsilon x(e)/2$ colors in Z_j below $(1+(D+1)\epsilon)x(e) \le (1+(D+1)\epsilon) f_\Phi(e)$.

The modification in the previous paragraph can change the finish times of the non-leaf edges, but the largest color of each edge remains in the same layer. By Inequality (5.7), $(1+2\epsilon)q_i \ge q_{i+1} - 1$, therefore the finish time of an edge can increase by at most a factor of $(1+2\epsilon)$. Moreover, since we modified only the non-leaf edges, there can be conflicts between the non-leaf and the leaf edges. But that problem is easy to solve: since the number of colors used by the non-leaf edges at vertex v from the main block of layer i does not increase, there are enough colors in layer i for assigning new colors to the leaf edges. After recoloring the leaf edges, the largest color of each edge remains in the same layer, hence the finish time of each leaf edge can increase by at most a factor of $1+2\epsilon$, and $f_\Psi(e) \le (1+2\epsilon)(1+(D+1)\epsilon) f_\Phi(e) \le (1+3D\epsilon) f_\Phi(e)$ follows for every edge e. ■

Call a coloring satisfying the requirements of Lemma 5.4.13 a *standard* coloring. Notice that on a non-leaf edge e only a constant number of different color sets can appear in standard colorings: the main zone is not empty only in a constant number of layers, and in each layer the (at most two) intervals can be placed in a constant number of different ways. More precisely, in a standard coloring edge e can use the main zone only from layer $\lfloor \log_{1+\epsilon} \epsilon x(e)/4 \rfloor$ to layer $\lceil \log_{1+\epsilon} 2x(e)/\epsilon \rceil$, that is, only in at most $\log_{1+\epsilon}((2x(e)/\epsilon)/(\epsilon x(e)/4)) + 2 = \log_{1+\epsilon} 8/\epsilon^2 + 2 = O(1/\epsilon \cdot \log 1/\epsilon)$ layers. In each layer, the end points of the intervals can take only at most $8/\epsilon^2$ different values, hence there are $(8/\epsilon^2)^2$ different possibilities for each of the two intervals. Therefore if we denote by $\mathcal{C}_e$ the different color sets that can appear in a standard coloring on non-leaf edge e, then $|\mathcal{C}_e| = ((8/\epsilon^2)^4)^{O((1/\epsilon)\cdot \log 1/\epsilon)} = 2^{O((1/\epsilon)\cdot \log^2 1/\epsilon)}$.

Theorem 5.4.14. *If every edge of $T(V,E)$ has at most D non-leaf child edges, then for every $\epsilon_0 > 0$, there is a $2^{O(D^2/\epsilon_0 \cdot \log^2(D/\epsilon_0))} \cdot n$ time algorithm that gives a $(1+\epsilon_0)$-approximate solution to the minimum sum edge multicoloring problem.*

Proof. Set $\epsilon := \epsilon_0/3D$. We use dynamic programming to find the best standard coloring: for every non-leaf edge e, and every set $S \in \mathcal{C}_e$, we determine $\mathrm{OPT}(e,S)$, which is defined to be the sum of the best standard coloring of T_e, with the additional requirement that edge e receives color set S. Clearly, if all the values $\{\mathrm{OPT}(r,S) : S \in \mathcal{C}_r\}$ are determined for the root edge r of T, then the minimum of these values is

the sum of the best standard coloring, which is by Lemma 5.4.13 at most $(1+3D\epsilon) = (1+\epsilon_0)$ times the minimum sum.

The values $\mathrm{OPT}(e,S)$ are calculated in a bottom-up traversal of the edges. Assume that e has k non-leaf child edges $e_1, e_2, \ldots, e_k$ and ℓ leaf child edges $e'_1, e'_2, \ldots, e'_\ell$. When $\mathrm{OPT}(e,S)$ is determined, the values $\mathrm{OPT}(e_i, S_i)$ are already available for every $1 \le i \le k$ and $S_i \in \mathcal{C}_{e_i}$. In a standard coloring of T_e every edge e_i is assigned a color set from $\mathcal{C}_{e_i}$. We enumerate all the $\prod_{i=1}^{k} |\mathcal{C}_{e_i}|$ possibilities for these color sets. For each combination $S_1 \in \mathcal{C}_{e_1}, \ldots, S_k \in \mathcal{C}_{e_k}$, we check whether these sets are pairwise disjoint. If so, then we determine the minimum sum that a standard coloring can have with these assignments. The minimum sum of subtree T_{e_i} with color set S_i on e_i is given by $\mathrm{OPT}(e_i, S_i)$. The finish time of edge e can be calculated from S. Now only the leaf edges $e'_1, \ldots, e'_\ell$ remain to be colored. It is easy to see that the best thing to do is to order these leaf edges by increasing demand size, and color them one after the other, using the colors not already assigned to e, $e_1, \ldots, e_k$. Therefore we can calculate the minimum sum corresponding to a choice of color sets $S_1 \in \mathcal{C}_{e_1}, \ldots, S_k \in \mathcal{C}_{e_k}$, and we set $\mathrm{OPT}(e,S)$ to the minimum over all the combinations.

The algorithm solves at most $\sum_{e \in E} |\mathcal{C}_e| = n \cdot 2^{O((1/\epsilon)\cdot \log^2 1/\epsilon)}$ subproblems. To solve a subproblem, at most $2^{O(D\cdot(1/\epsilon)\cdot\log^2 1/\epsilon)}$ different combinations of the sets $S_1, \ldots, S_k$ have to be considered. Each color set can be described by $O(1/\epsilon \cdot \log 1/\epsilon)$ intervals, and the time required to handle each combination is polynomial in D and the number of intervals. Therefore the total running time of the algorithm is $2^{O(D\cdot 1/\epsilon \cdot \log^2(1/\epsilon))} \cdot n = 2^{O(D^2/\epsilon_0 \cdot \log^2(D/\epsilon_0))} \cdot n$. ■

5.4.5 The general case

In this section we prove that minimum sum edge multicoloring admits a PTAS for arbitrary trees. The edges of the tree are partitioned into subtrees in such a way that each subtree is an almost bounded degree tree (recall that in an almost bounded degree tree each node has a bounded number of non-leaf child edges). Now the algorithm presented in Section 5.4.4 can be used to obtain a good approximate coloring for each subtree. These colorings can be merged into a coloring of the whole tree, but this coloring will not be necessarily a proper coloring, since there might be conflicts between edges that were in different subtrees. However, using a series of transformations, these conflicts can be resolved with only a small increase of the sum.

Theorem 5.4.15. *For every $\epsilon_0 > 0$, there is a $2^{O(1/\epsilon_0^{11} \cdot \log^2(1/\epsilon_0))} \cdot n$ time algorithm that gives a $(1+\epsilon_0)$-approximate solution to the minimum sum edge multicoloring problem for every tree T and demand function x_0.*

Proof. Let $\epsilon := \epsilon_0/32$. The algorithm consists of a series of phases. The last phase produces a proper coloring of (T, x_0), and has cost at most $(1+\epsilon_0)\mathrm{OPT}(T, x_0)$. In the following we describe these phases.

Phase 1: Rounding the Demands. Let $x(e)$ be the smallest q_i that is not smaller than $x_0(e)$. Since $q_{i+1} \le (1+\epsilon)^{i+1} \le (1+\epsilon)(q_i+1)$, thus $x(e) \le (1+\epsilon)x_0(e)$. Therefore by Lemma 5.4.9, this modification increases the minimum sum by at most a factor of $1+\epsilon$. An edge e with demand q_i will be called a *class* i edge (if $x(e) = q_i$ for more than one i, then take the smallest i). The class of edge e will be denoted by $\mathrm{class}(e)$.

Phase 2: Partitioning the Tree. We partition the edges of the tree into connected components such that in every subtree the number of non-leaf child edges of a node is bounded by a constant. To obtain this partition, the edges of the tree are divided into *large edges*, *small edges*, and *frequent edges*. It will be done in such a way that every node has at most $D := 6/\epsilon^5$ large child edges. If a node has less than D children, then its child edges are large edges. Let v be a node with at least D children, and denote by $n(v,i)$ the number of class i child edges of v. Let $N(v)$ be the largest i such that $n(v,i) > 0$ and set

$F := 6/\epsilon^3$. Let e be a class i child edge of v. If $n(v,i) > F$, then e is a *frequent edge*. If $n(v,i) \le F$ and $i \le N(v) - 1/\epsilon^2$, then e is a *small edge*. Otherwise, if $n(v,i) \le F$ and $i > N(v) - 1/\epsilon^2$, then e is a *large edge*. Clearly, v can have at most $F \cdot 1/\epsilon^2 = 6/\epsilon^5 = D$ large child edges: for each class $N(v)$, $N(v)-1$, $\ldots$, $N(v) - 1/\epsilon^2 + 1$, there are at most F such edges.

The tree is split at the lower node of every small and frequent edge, the connected components of the resulting forest form the classes of the partition. Another way to describe this partition: delete every small and large edge, make the connected components of the remaining graph the classes of the partition, and put every deleted edge into the class where its upper node belongs. Clearly, every small and frequent edge becomes a leaf edge in its subtree, thus if every node has at most D large child edges in the tree, then in every subtree each node has at most D non-leaf child edges.

Color each subtree with the algorithm of Theorem 5.4.14. This step can be done in $2^{O(D^2/\epsilon \cdot \log^2(D/\epsilon))} \cdot n = 2^{O(36/\epsilon^{10} \cdot 1/\epsilon \cdot \log^2(6/\epsilon^6))} \cdot n = 2^{O(1/\epsilon^{11} \cdot \log^2(1/\epsilon))} \cdot n$ time. Each coloring is a $(1+\epsilon)$-approximate coloring of the given subtree, thus merging these colorings yields a (not necessarily proper) coloring Ψ_1 of T such that $f_{\Psi_1}(T) \le (1+\epsilon)OPT(T,x)$. In the rest of the proof, we transform Ψ_1 into a proper coloring in such a way that the sum of the coloring does not increase too much.

Phase 3: Small Edges. Consider the (ϵ,ℓ)-augmentation of the coloring Ψ_1 with $\ell := 6$. This results in a coloring Ψ_2 such that $f_{\Psi_2}(G) \le (1+\epsilon\ell) f_{\Psi_1}(G)$ (see Section 5.4.4). First we modify Ψ_2 in such a way that the small edges use only the extra zones Z_1 and Z_2. More precisely, if a small edge e has parity $r \in \{1,2\}$, then e is recolored using the colors in Z_r (recall that the parity of the edge is the parity of its upper node). Since the extra zones contain only a very small fraction of the color spectrum, the recoloring can significantly increase the finish time of the small edges, but not more than by a factor of $2/\epsilon$ (Lemma 5.4.12b). However, we show that the total demand of the small edges at v is so small compared to the largest demand on the child edges of v, that their total finish time will be negligible, even after this large increase. By definition, the largest child edge of v has demand $q_{N(v)}$.

Let S_v be the set of those small edges whose upper node is v. Let r be the parity of v. Color the edges in S_v one after the other, in the order of increasing demand size, using only the colors in Z_r. Call the resulting coloring Ψ_3. We claim that $f_{\Psi_3}(S_v) \le \epsilon q_{N(v)}$ for every node v, thus transforming Ψ_2 into Ψ_3 increases the total sum by at most $\sum_{v \in T} f_{\Psi_3}(S_v) \le \epsilon \sum_{v \in T} q_{N(v)} \le \epsilon f_{\Psi_2}(T)$ and $f_{\Psi_3}(T) \le (1+\epsilon) f_{\Psi_2}(T)$ follows. To give an upper bound on $f_{\Psi_3}(S_v)$, we assume the worst case, that is, $n(v,i) = F$ for every $i \le N(v) - 1/\epsilon^2$. Imagine first that the small edges are colored using the full color spectrum, not only with the colors of zone Z_r. Assume that the small edges are colored in the order of increasing demand size, and consider a class k edge e. In the coloring, only edges of class not greater than k are colored before e. Hence the finish time of e is at most

$$\sum_{i=0}^{k} n(v,i) q_i \le F \sum_{i=0}^{k} (1+\epsilon)^i \le 6(1+\epsilon)/\epsilon^4 \cdot (1+\epsilon)^k$$
$$\le 14/\epsilon^4 \cdot \frac{1}{2}(1+\epsilon)^k \le 14/\epsilon^4 \cdot \lfloor (1+\epsilon)^k \rfloor = 14/\epsilon^4 \cdot q_k.$$

That is, the finish time of an edge is at most $14/\epsilon^4$ times its demand (in the second inequality, we used $\sum_{i=0}^{k} (1+\epsilon)^i = ((1+\epsilon)^{k+1} - 1)/\epsilon < (1+\epsilon)^{k+1}/\epsilon$). Therefore the total finish time of the small edges is at most $14/\epsilon^4$ times the total demand, which is

$$\frac{14}{\epsilon^4} \sum_{i=0}^{N(v)-1/\epsilon^2} n(v,i) q_i \le \frac{84}{\epsilon^7} \sum_{i=0}^{N(v)-1/\epsilon^2} (1+\epsilon)^i$$
$$\le \frac{85}{\epsilon^8} (1+\epsilon)^{N(v)-1/\epsilon^2} \le \frac{85}{\epsilon^8} \cdot 2^{-1/\epsilon} \cdot (1+\epsilon)^{N(v)}$$
$$\le \frac{\epsilon^2}{2} \cdot \frac{1}{2} (1+\epsilon)^{N(v)} \le \frac{\epsilon^2}{2} \cdot \lfloor (1+\epsilon)^{N(v)} \rfloor = \frac{\epsilon^2}{2} \cdot q_{N(v)}.$$

(In the third inequality we use $(1+\epsilon)^{1/\epsilon} \geq 2$, in the fourth inequality it is assumed that ϵ is sufficiently small that $2^{1/\epsilon} \geq 4 \cdot 85/\epsilon^{10}$ holds.) However, the small edges do not use the full color spectrum, only the colors in zone Z_r. By Lemma 5.4.12b, zone Z_r contains at least c colors up to $2/\epsilon \cdot c$, thus every finish time in the calculation above should be multiplied by at most $2/\epsilon$. Therefore the sum of the small edges is at most

$$f_{\Psi_3}(S_v) \leq 2/\epsilon \cdot \frac{\epsilon^2}{2} \cdot q_{N(v)} \leq \epsilon q_{N(v)},$$

as claimed.

Phase 4: Shifting the Frequent Edges. Now we have a coloring Ψ_3 that is still not a proper coloring, but conflicts appear only between some frequent edges and their child edges. In Phases 4 and 5 we ensure that every frequent edge e uses only colors greater than $2x(e)/\epsilon$ from the main zone. In Phase 6, the conflicts are resolved using a set of so far unused colors, the colors in extra zones Z_5 and Z_6.

Let F_v be the set of frequent child edges of v, and let $\Lambda_v = \bigcup_{e \in F_v} \Psi_3(e)$ be the colors used by the frequent child edges of node v. We recolor the edges in F_v using only the colors in Λ_v and some colors from zones Z_3 and Z_4. Let $e_1, e_2, \ldots, e_{|F_v|}$ be an ordering of the edges in F_v by increasing demand size. Recall that the algorithm in Theorem 5.4.14 assigned the colors to the leaf edges in increasing order of demand size, thus it can be assumed that frequent edge e_1 uses the first $x(e_1)$ colors in Λ_v, edge e_2 uses the $x(e_2)$ colors after that, etc. Denote by $t(c) = |\{e \in F_v : f_{\Psi_3}(e) \geq c\}|$ the number of edges whose finish time is at least c, and denote by $t(c,i) = |\{e \in F_v : f_{\Psi_3}(e) \geq c,\ \text{class}(e) = i\}|$ the number of class i edges among them. Clearly, $t(c) = \sum_{i=0}^{\infty} t(c,i)$ holds. Moreover, it can be easily verified that the total finish time of the edges in F_v can be expressed as $f_{\Psi_3}(F_v) = \sum_{c=1}^{\infty} t(c)$.

The first step is to produce a coloring Ψ_4 where every frequent edge e has only $(1-2\epsilon/5)x(e)$ colors, but these colors are all greater than $2x(e)/\epsilon$. The demand function is split into two parts: $x(e) = x_1(e) + x_2(e)$, where $x_1(e)$ is $(1-2\epsilon/5)x(e)$ and $x_2(e)$ is $2\epsilon x(e)/5$, but rounding has to be done carefully. What we want to achieve is that

$$\sum_{j=1}^{k} x_2(e_j) \leq \frac{2\epsilon}{5} \sum_{j=1}^{k} x(e_j) \tag{5.10}$$

holds for every $1 \leq k \leq |F_v|$, and the total demand of the class i edges in x_1 is at most

$$\sum_{e \in F_v,\ \text{class}(e)=i} x_1(e) \leq \lceil n(e,i)(1-2\epsilon/5)q_i \rceil. \tag{5.11}$$

It can be easily verified that these two requirements hold if x_1 is defined as $x_1(e) = \lceil (1-2\epsilon/5)q_i \rceil$ for the first m edges of class i, and $x_1(e) = \lfloor (1-2\epsilon/5)q_i \rfloor$ for the rest of the class i edges, where

$$m = \lceil n(v,i)(1-2\epsilon/5)q_i \rceil - n(v,i) \lfloor (1-2\epsilon/5)q_i \rfloor .$$

This phase of the algorithm produces a coloring Ψ_4 of F_v that assigns only $x_1(e)$ colors to every edge $e \in F_v$, but satisfies the condition that it uses only the colors in Λ_v, and every edge e receives only colors greater than $2x(e)/\epsilon$. In the next phase we will extend this coloring using the colors in zones Z_3 and Z_4: every edge e will receive an additional $x_2(e)$ colors.

Coloring Ψ_4 is defined as follows. Consider the edges $e_1, \ldots, e_{|F_v|}$ in this order, and assign to e_k the first $x_1(e_k)$ colors in Λ_v greater than $2x(e_k)/\epsilon$ and not already assigned to an edge e_j $(j < k)$. Notice the following property of Ψ_4: if $j < k$, then every color in $\Psi_4(e_j)$ is less than every color in $\Psi_4(e_k)$. This follows from $2x(e_j)/\epsilon \leq 2x(e_k)/\epsilon$: every color usable for e_k is also usable for e_j if $j < k$. Define $t'(c) = |\{e \in F_v : f_{\Psi_4}(e) \geq c\}|$ and $t'(c,i) = |\{e \in F_v : f_{\Psi_4}(e) \geq c,\ \text{class}(e) = i\}|$ as before, but now using the coloring Ψ_4. We claim that $t'(c,i) \leq (1+\epsilon)t(c,i)$ holds for every $c \geq 1, i \geq 0$. If this is true, then $t'(c) \leq (1+\epsilon)t(c)$ holds and $f_{\Psi_4}(F_v) \leq (1+\epsilon)f_{\Psi_3}(F_v)$ follows from $f_{\Psi_4}(F_v) = \sum_{c=1}^{\infty} t'(c)$. Summing this for every node v gives $f_{\Psi_4}(T) \leq (1+\epsilon)f_{\Psi_3}(T)$.

First we show that $t'(c,i) \leq t(c,i) + 2/\epsilon$. If every class i edge has finish time at least c in Ψ_3, then $t(c,i) = n(c,i) \geq t'(c,i)$ and we are ready. Therefore there is at least one class i edge that has finish time less than c in Ψ_3. This implies that the frequent edges of class 0, 1, ..., $i-1$ use only colors less than c. Denote by X the total demand of these edges (in the demand function $x(e)$), and denote by Y the number of colors used by the class i edges below c in Ψ_3.

Now recall the way Ψ_4 was defined, and consider the step when every edge with class less than i is already colored. At this point at most X colors of Λ_v are used below c (possibly less, since Ψ_4 assigns only $x_1(e)$ colors to every edge e, and only colors above $2x(e)/\epsilon$). Therefore at least Y colors are still unused in Λ_v below c. From these colors at least $Y - 2q_i/\epsilon$ of them are above $2q_i/\epsilon$. Thus Ψ_4 can color at least $(Y - 2q_i/\epsilon)/q_i = Y/q_i - 2/\epsilon$ edges of class i using only colors below c. However, Ψ_3 uses Y colors below c for the class i edges, hence it can color at most Y/q_i such edges below c, and $t'(c,i) \leq t(c,i) + 2/\epsilon$ follows.

We consider two cases. If $t(c,i) \geq 2/\epsilon^2$, then $t'(c,i) \leq t(c,i) + 2/\epsilon \leq (1+\epsilon)t(c,i)$, and we are ready. Let us assume therefore that $t(c,i) \leq 2/\epsilon^2$, it will turn out that in this case $t'(c,i) = 0$. There are $n(v,i) - t(c,i) \geq n(v,i) - 2/\epsilon^2$ class i edges that has finish time less than c in Ψ_3. Therefore, as in the previous paragraph, before Ψ_4 starts coloring the class i edges, there are at least $(n(v,i) - 2/\epsilon^2) \cdot q_i$ unused colors less than c in Λ_v. By (5.11), the total demand of the class i edges in demand function $x_1(e)$ is at most $\lceil n(e,i)(1 - 2\epsilon/5)q_i \rceil$. The following calculation shows that the unused colors below c in Λ_v is sufficient to satisfy all these edges, thus Ψ_4 assigns to these edges only colors less than c. We have to skip the colors not greater than $2q_i/\epsilon$, these colors cannot be assigned to the edges of class i, which means that the number of usable colors is at least

$$\begin{aligned}(n(v,i) - 2/\epsilon^2) \cdot q_i - 2q_i/\epsilon &\geq (n(v,i) - 12/5\epsilon^2) \cdot q_i + 1 \\ &\geq (1 - 2\epsilon/5)n(v,i)q_i + 1 \geq \lceil n(e,i)(1 - 2\epsilon/5)q_i \rceil,\end{aligned}$$

since $n(v,i) \geq 6/\epsilon^3$ by the definition of the frequent edges. Therefore Ψ_4 assigns to the class i edges only colors less than c, and $t(c,i) = 0$ follows.

Phase 5: Full Demand for the Frequent Edges. The next step is to modify Ψ_4 such that every frequent edge receives $x(e)$ colors, not only $x_1(e)$. Coloring Ψ_5 is obtained from Ψ_4 by assigning to every frequent edge e an additional $x_2(e)$ colors from zone Z_3 or Z_4. More precisely, let v be a node with parity r, and let $e_1, \ldots, e_{|F_v|}$ be its frequent child edges, ordered in increasing demand size, as before. Assign to e_1 the first $x_2(e_1)$ colors from Z_{2+r}, to e_2 the first $x_2(e_2)$ colors from Z_{2+r} not used by e_1, etc. It is clear that no conflict arises with the assignment of these colors.

We claim that these additional colors do not increase the finish time of the frequent edges. Let $x_i^* = \sum_{j=1}^{i} x_1(e_j)$ be the total demand in x_1 of the first i frequent edges at v. The finish time of e_i in Ψ_4 is clearly at least x_i^*, since Ψ_4 colors every edge e_j with $j < i$ before e_i. On the other hand, by Lemma 5.4.12b, zone Z_{2+r} contains at least $\lfloor \epsilon x_i^*/2 \rfloor$ colors not greater than x_i^*. These colors are sufficient to satisfy the additional demand of the first i edges: by (5.10) the first i edges need a total of at most $\frac{2\epsilon}{5} \sum_{j=1}^{i} x(e) \leq \epsilon x_i^*/2$ colors.

Phase 6: Resolving the Conflicts. Now we have a coloring Ψ_5 such that there are conflicts only between frequent edges and their child edges. Furthermore, if e is a frequent edge, then $\Psi_5(e)$ contains only colors greater than $2x(e)/\epsilon$ from the main zone. It is clear from the construction of Ψ_5 that only the colors in the main zone can conflict.

Let e be a frequent edge that conflicts with some of its children. Assume that the child edges of e have parity r. There are at most $x(e)$ colors that are used by both e and a child of e. We resolve this conflict by recoloring the child edges of e in such a way that they use the first at most $x(e)$ colors in zone Z_{4+r} instead of the colors in $\Psi_5(e)$. It is clear that if this operation is applied for every frequent edge e, then the resulting color Ψ_6 is a proper coloring.

Notice that if a child edge e' of e is recolored, then it has finish time at least $2x(e)/\epsilon$, otherwise it does not conflict with e. On the other hand, by Lemma 5.4.12b, zone Z_{4+r} contains at least $x(e)$ colors up to $2x(e)/\epsilon$, thus the recoloring does not add colors above that. Therefore the finish time of e' is not increased.

Analysis. The sum of the coloring Ψ_6 can be bounded as follows (assuming that ϵ is sufficiently small):

$$\begin{aligned} f_{\Psi_6}(T) &= f_{\Psi_5}(T) = f_{\Psi_4}(T) \leq (1+\epsilon) f_{\Psi_3}(T) \\ &\leq (1+\epsilon)^2 f_{\Psi_2}(T) \leq (1+(\ell+1)\epsilon)(1+\epsilon)^2 f_{\Psi_1}(T) \\ &\leq (1+(\ell+1)\epsilon)(1+\epsilon)^3 \mathrm{OPT}(T,x) \leq (1+(\ell+1)\epsilon)(1+\epsilon)^4 \mathrm{OPT}(T,x_0) \\ &\leq (1+7\epsilon)(1+8\epsilon)\mathrm{OPT}(T,x_0) \leq (1+32\epsilon)\mathrm{OPT}(T,x_0) \\ &= (1+\epsilon_0)\mathrm{OPT}(T,x_0) \end{aligned}$$

Therefore Ψ_6 is a $(1+\epsilon_0)$-approximate solution to the minimum sum edge multicoloring instance (T, x_0).

The running time of the algorithm is dominated by the coloring of the low-degree components with the algorithm of Theorem 5.4.14. This phase requires $2^{O(36/\epsilon^{10} \cdot 1/\epsilon \cdot \log^2(6/\epsilon^6))} \cdot n = 2^{O(1/\epsilon_0^{11} \log^2(1/\epsilon_0))} \cdot n$ time. The other parts of the algorithm can be done in time linear in the size of the input. Therefore the total running time is $2^{O(1/\epsilon_0^{11} \log^2(1/\epsilon_0))} \cdot n$, which completes the proof of Theorem 5.4.15. ■

CHAPTER 6

Clique coloring

The worst cliques are those which consist of one man.
George Bernard Shaw (1856–1950)

In clique coloring we have to satisfy weaker requirements than in ordinary vertex coloring. Instead of requiring that the two end points of each edge have two different colors, we only require that every inclusionwise maximal (nonextendable) clique contains at least two different colors. It is possible that a graph is k-clique-colorable, but its chromatic number is greater than k. For example, a clique of size n is 2-clique-colorable, but its chromatic number is n.

Clique coloring can be also thought of as coloring the clique hypergraph. Given a graph $G(V, E)$, the *clique hypergraph* $\mathcal{C}(G)$ of G is defined on the same vertex set V, and a subset $V' \subseteq V$ is a hyperedge of $\mathcal{C}(G)$ if and only if $|V'| > 1$ and V' induces an inclusionwise maximal clique of G. The study of the clique hypergraph was initiated by Gallai. Most of the research on the clique hypergraph focuses on the transversals of $\mathcal{C}(G)$, that is, how many vertices are required to meet all the maximal cliques [EGT92, AF96]. However, it is equally natural to study the colorings of the clique hypergraph. Coloring a hypergraph means assigning colors to the vertices in such a way that every hyperedge contains at least two colors. Clearly, a graph G is k-clique-colorable if and only if the hypergraph $\mathcal{C}(G)$ is k-colorable. If the graph G is triangle-free, then the maximal cliques are the edges, hence $\mathcal{C}(G)$ is the same as G.

Clique coloring can be very different from ordinary vertex coloring. The most notable difference is that clique coloring is not a hereditary property: it is possible that a graph is k-clique-colorable, but it has a subgraph that is not. The reason why this can happen is that deleting vertices can create new inclusionwise maximal cliques: it is possible that in the original graph a clique is contained in a larger clique, but after deleting some vertices this clique becomes maximal. Another difference is that a large clique is not an obstruction for clique colorability, even 2-clique-colorable graphs can contain large cliques. In fact, it is conjectured that every prefect graph (or perhaps every odd-hole free graph) is 3-clique-colorable (see [BGG$^+$04]). There are no counterexamples known for this conjecture, but so far only some special cases have been proved.

In this chapter we prove complexity results for clique coloring and related problems. Clique coloring is harder than ordinary vertex coloring: it is coNP-complete even to check whether a 2-clique-coloring is valid [BGG$^+$04]. The complexity of 2-clique-colorability is investigated in [KT02], where they show that it is NP-hard to decide whether a perfect graph is 2-clique-colorable. However, it is not clear whether this problem belongs to NP. A valid 2-clique-coloring is not a good certificate, since we cannot verify it in

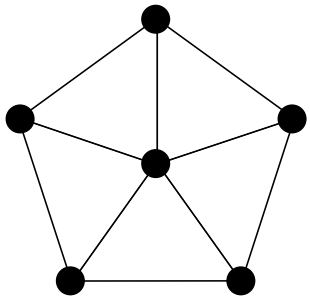


Figure 6.1: The graph is 2-clique-colorable, but it does not remain 2-clique-colorable after deleting the central vertex.

polynomial time: as mentioned above, it is coNP-complete to check whether a 2-clique-coloring is valid. In Section 6.2 we determine the exact complexity of the problem: we show that it is Σ_2^p-complete to check whether a graph is 2-clique-colorable.

A graph is *k-clique-choosable* if whenever a list of k colors are assigned to each vertex (the lists of the different vertices do not have to be the same), then the graph has a clique coloring where the color of each vertex is taken from its list. This notion is an adaptation of choosability introduced for graphs independently by Erdős, Rubin, and Taylor [ERT80] and by Vizing [Viz76]. In [MŠ99] it is shown that every planar or projective planar graph is 4-clique-choosable. In Section 6.3 we investigate the complexity of clique-choosability. It turns out that the complexity of clique-choosability lies higher in the polynomial hierarchy than either clique-coloring or choosability: we show that for every $k \geq 2$ it is Π_3^p-complete to decide whether a graph is k-clique-choosable or not.

As mentioned above, a k-clique-colorable graph can contain an induced subgraph that is not k-clique-colorable. Therefore it is natural to investigate graphs that are *hereditary k-clique-colorable:* graphs where every induced subgraph is k-clique-colorable. In Section 6.4 we show that recognizing such graphs is Π_3^p-complete for every $k \geq 3$.

The results in this chapter are taken from [Mar04b].

6.1 Preliminaries

In this section we introduce notations and make some preliminary observations about clique colorings. We also introduce the complexity classes that appear in our completeness results (see also Appendix A.3 for more background on these classes).

A *clique* is a complete subgraph of at least 2 vertices. A clique is *maximal* if it cannot be extended to a larger clique. An edge is *flat* if it is not contained in any triangle. Since a flat edge is a maximal clique of size 2, the two end vertices of a flat edge receive different colors in every proper clique coloring. The *core* of G is the subgraph containing only the flat edges. Clearly, a proper clique coloring of G is a proper vertex coloring of the core of G. A vertex v of G is *simple* if it is not contained in any triangle, or, equivalently, all the edges incident to it are flat.

Unlike k-vertex-coloring, a k-clique-coloring of the graph G does not necessarily give a proper k-clique-coloring for the induced subgraphs of G. In fact, it can happen that deleting vertices from G makes it impossible to k-clique-color it. For example, the 5-wheel shown on Figure 6.1 is 2-clique-colorable, but after deleting the central vertex, the remaining C_5 is not. On the other hand, the following proposition shows that G remains k-clique-colorable if we delete only simple vertices:

Proposition 6.1.1. *Let $S \subseteq V$ be a set of simple vertices in $G(V, E)$. If ψ is a proper clique coloring of G, then ψ induces a proper clique coloring of $G \setminus S$.*

Proof. Consider the coloring ψ' of $G \setminus S$ induced by ψ. If ψ' is not a clique coloring of G, then there is a monochromatic maximal clique K in $G \setminus S$. This is not a maximal clique in G, otherwise ψ would not be a proper k-clique-coloring. Therefore K is properly contained in a maximal clique K' of G. Since K' is not a maximal clique of $G \setminus S$, it contains at least one vertex v of S. However, K' has size at least 3, contradicting the assumption that vertex $v \in S$ is simple. ■

The following two propositions will also be useful:

Proposition 6.1.2. *Let $S \subseteq V$ be an arbitrary subset of the vertices in $G(V, E)$. If ψ induces a proper clique coloring of $G \setminus S$, and every vertex in S has different color from its neighbors, then ψ is a proper clique coloring of G.*

Proof. If there is a monochromatic maximal clique K in G, then it contains a vertex v from S. However, vertex v has different color from its neighbors, a contradiction. ■

Proposition 6.1.3. *Let $S \subseteq V$ be the set of simple vertices in $G(V, E)$. If ψ is a k-clique-coloring of G, then it induces a proper k-vertex-coloring of $G[S]$, the graph induced by S.*

Proof. Observe that every edge in $G[S]$ is a flat edge, they are maximal cliques in G. Therefore ψ assigns different colors to the end vertices of every edge in $G[S]$, thus it induces a proper k-vertex-coloring of $G[S]$. ■

The complexity class $\Sigma_2^p = \mathrm{NP}^{\mathrm{NP}}$ contains those problems that can be solved by a polynomial time nondeterministic Turing machine equipped with an NP-oracle. An oracle can be thought of as a subroutine that is capable of solving instantaneously a certain problem. More formally, let L be a language. A Turing machine equipped with an L oracle has a special tape called the oracle tape. Whenever the Turing machine wishes, it can ask the oracle whether the contents of the oracle tape is a word from L or not (there are special states for this purpose). Asking the oracle counts as only one step. If the language L is simple, then this oracle does not help very much. On the other hand, if L is a computationally hard language, then this oracle can increase the power of the Turing machine. We say that a Turing machine is equipped with an NP-oracle, if the language L is NP-complete. Note that here it is not really important which particular NP-complete language is L: any NP-complete language gives the same power to the Turing machine, up to a polynomial factor. Thus the class Σ_2^p contains those problems that can be solved in polynomial time by a polynomial time nondeterministic Turing machine if one NP-complete problem (say, the satisfiability problem) can be solved in constant time.

Similarly to NP, the class Σ_2^p has an equivalent characterization using certificates. A problem is in NP if there is a polynomial-size certificate for each yes-instance, and verifying this certificate is a problem in P. The characterization of the class Σ_2^p is similar, but here we require only that verifying the certificate is in coNP (cf. [Pap94] for more details).

Like SAT, which is the canonical complete problem for NP, the problem QSAT_2 is the canonical Σ_2^p-complete problem:

2-Quantified Satisfiability (QSAT_2)

Input: An $n + m$ variable boolean 3DNF formula $\varphi(\mathbf{x}, \mathbf{y})$ (where $\mathbf{x} = (x_1, \ldots, x_n)$, $\mathbf{y} = (y_1, \ldots, y_m)$)

Question: Is there a vector $\mathbf{x} \in \{0,1\}^n$ such that for every $\mathbf{y} \in \{0,1\}^m$, $\varphi(\mathbf{x}, \mathbf{y})$ is true? (Shorthand notation: is it true that $\exists \mathbf{x} \forall \mathbf{y} \varphi(\mathbf{x}, \mathbf{y})$?)

A 3DNF (disjunctive normal form) formula is a disjunction of terms, where each term is a conjunction of 3 literals. To see that QSAT_2 is in Σ_2^p, observe that if the answer is yes to the problem, then an assignment $\mathbf{x}_0$ satisfying $\forall \mathbf{y} \varphi(\mathbf{x}_0, \mathbf{y})$ is a good certificate. To verify the certificate, we have to check that $\forall \mathbf{y} \varphi(\mathbf{x}_0, \mathbf{y})$ holds, or equivalently, we have to check whether there is a $\mathbf{y}$ such that $\varphi(\mathbf{x}_0, \mathbf{y})$ is false. This verification problem is in coNP, hence QSAT_2 is in Σ_2^p. For the proof that QSAT_2 is hard for Σ_2^p, see e.g., [Pap94].

The complexity class Π_2^p contains those languages whose complements are in Σ_2^p. The class Σ_3^p contains the problems solvable by a polynomial time nondeterministic Turing machine equipped with a Σ_2^p oracle. The following problem is complete for Σ_3^p:

3-Quantified Satisfiability (QSAT_3)

Input: An $n + m + p$ variable boolean 3CNF formula $\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ ($\mathbf{x} = (x_1, \ldots, x_n)$, $\mathbf{y} = (y_1, \ldots, y_m)$, $\mathbf{z} = (z_1, \ldots, z_p)$)

Question: Is there a vector $\mathbf{x} \in \{0,1\}^n$ such that for every $\mathbf{y} \in \{0,1\}^m$, there is a vector $\mathbf{z} \in \{0,1\}^p$ with $\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ true? (Shorthand notation: is it true that $\exists \mathbf{x} \forall \mathbf{y} \exists \mathbf{z} \varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$?)

Repeating this construction, we obtain the polynomial hierarchy: let Σ_{i+1}^p contain those problems that can be solved by a polynomial time nondeterministic Turing machine equipped with a Σ_i^p-oracle. The class Π_i^p contains those languages whose complement is in Σ_i^p. The definition of these classes might seem very technical, but as this chapter shows, there are natural problems whose complexities are exactly characterized by these classes.

6.2 Complexity of clique coloring

In this section we investigate the complexity of the following problem:

***k*-Clique-Coloring**

Input: A graph $G(V, E)$

Question: Is there a *k-clique-coloring* c: $V \to \{1, 2, \ldots, k\}$ such that in every maximal clique K of G, there are two vertices $u, v \in K$ with $c(u) \neq c(v)$?

Unlike ordinary vertex coloring, which is easy for two colors, this problem is hard even for $k = 2$:

Theorem 6.2.1. *2-Clique-Coloring is Σ_2^p-complete.*

Proof. To see that k-Clique-Coloring is in Σ_2^p, notice that the problem of verifying whether a coloring is a proper k-clique-coloring is in coNP: a monochromatic maximal clique is a polynomial time verifiable certificate that the coloring is *not* proper. A proper k-clique-coloring is a certificate that the graph is k-clique-colorable, and this certificate can be verified in polynomial time if an NP-oracle is available. Thus clearly the problem is in $\mathrm{NP}^{\mathrm{NP}} = \Sigma_2^p$.

We prove that 2-Clique-Coloring is Σ_2^p-complete by a reduction from QSAT_2. For a formula $\varphi(\mathbf{x}, \mathbf{y})$, we construct a graph G that is 2-clique-colorable if and only if there is an $\mathbf{x} \in \{0,1\}^n$ such that $\varphi(\mathbf{x}, \mathbf{y})$ is true for every $\mathbf{y} \in \{0,1\}^m$. Graph G has $4(n + m) + 2q$ vertices, where q is the number of terms in φ. For every variable x_i ($1 \leq i \leq n$), the graph contains a path of 4 vertices x_i, x_i', $\overline{x}_i'$, $\overline{x}_i$. For every variable y_j ($1 \leq j \leq m$), the graph contains 4 vertices y_j, y_j', $\overline{y}_j$, $\overline{y}_j'$. These $4m$ vertices form a path y_1, y_1', $\overline{y}_1$, $\overline{y}_1'$, y_2, $\ldots$, $\overline{y}_m$, $\overline{y}_m'$. Furthermore, the vertices x_i, $\overline{x}_i$, y_j, $\overline{y}_j$ form a clique of size $2(n+m)$ minus a matching: there are no edges between x_i and $\overline{x}_i$, and between y_j and $\overline{y}_j$.

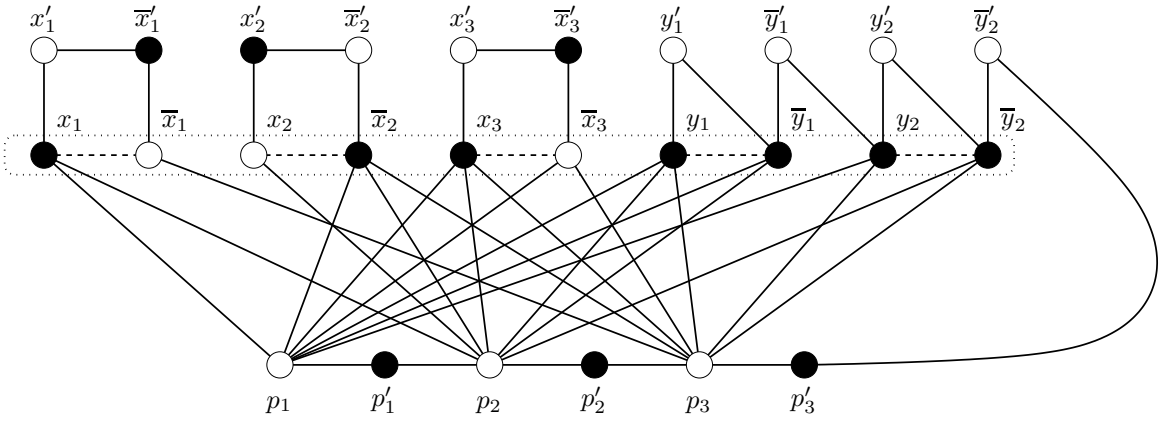


Figure 6.2: The construction for the formula $\varphi = (x_1 \wedge \overline{x}_2 \wedge y_2) \vee (x_1 \wedge x_3 \wedge \overline{y}_2) \vee (\overline{x}_1 \wedge \overline{x}_2 \wedge y_1)$. The vertices x_1, $\overline{x}_1$, x_2, ..., y_2, $\overline{y}_2$ form a clique minus the five dashed edges. The coloring shown on the figure is a proper 2-clique-coloring, implying that $x_1 = 1$, $x_2 = 0$, $x_3 = 1$ satisfy φ regardless of the values of y_1 and y_2.

For every term P_ℓ $(1 \leq \ell \leq q)$ of the DNF formula φ, the graph contains two vertices p_ℓ and p'_ℓ. These vertices form a path p_1, p'_1, p_2, p'_2, ..., p_q, p'_q of $2q$ vertices, and vertex p'_q is connected to $\overline{y}'_m$. Vertex p_ℓ is connected to those literals that correspond to literals not contradicting P_ℓ. That is, if x_i (resp. $\overline{x}_i$) is in P_ℓ, then p_ℓ and x_i (resp. $\overline{x}_i$) is connected. (It is assumed that at most one of x_i and $\overline{x}_i$ appears in a term.) If neither x_i nor $\overline{x}_i$ appears in P_ℓ, then p_ℓ is connected to both x_i and $\overline{x}_i$. Vertices y_j and $\overline{y}_j$ are connected to p_ℓ in a similar fashion. This completes the description of the graph G. An example is shown on Figure 6.2. Notice that $\varphi(\mathbf{x}, \mathbf{y})$ is true for some variable assignment $\mathbf{x}$, $\mathbf{y}$ if and only if there is a vertex p_ℓ such that it is connected to all the $n + m$ true literals.

Assume that $\mathbf{x} \in \{0,1\}^n$ such that $\varphi(\mathbf{x}, \mathbf{y})$ is true for every $\mathbf{y} \in \{0,1\}^m$. We define a 2-clique-coloring of the graph G based on $\mathbf{x}$. Vertices p_ℓ $(1 \leq \ell \leq q)$ and y'_j, $\overline{y}'_j$ $(1 \leq j \leq m)$ are colored white. If x_i is true in $\mathbf{x}$, then vertices x'_i and $\overline{x}_i$ are colored white, while vertices x_i and $\overline{x}'_i$ are black; if x_i is false in $\mathbf{x}$, then vertices x'_i and $\overline{x}_i$ are colored black, and vertices x_i, $\overline{x}'_i$ are white. The remaining vertices are black.

It can be verified that the coloring defined above properly colors every flat edge of the graph. Now suppose that there is a monochromatic maximal clique K of size greater than two. Since vertices x'_i, $\overline{x}'_i$, y'_i, $\overline{y}'_i$, p'_ℓ are simple vertices, they cannot appear in K. Assume first that K is colored white, then it contains some of the $2n$ vertices x_i, $\overline{x}_i$ $(1 \leq i \leq n)$, and at most one of the vertices p_ℓ $(1 \leq \ell \leq q)$ (the vertices y_j, $\overline{y}_j$ are black). However, this clique is not maximal: p_ℓ is connected to at least one of y_1 and $\overline{y}_1$, therefore K can be extended by one of these two vertices. Now suppose that K is colored black, then it can contain only vertices of the form x_i , $\overline{x}_i$, y_j, $\overline{y}_j$. Furthermore, for every $1 \leq i \leq n$, clique K contains exactly one of x_i and $\overline{x}_i$, and for every $1 \leq j \leq m$, clique K contains exactly one of y_j and $\overline{y}_j$, otherwise K is not a maximal clique. Define the vector $\mathbf{y}$ such that variable y_j is true if and only if vertex y_j is in K. By the assumption on $\mathbf{x}$, the value of $\varphi(\mathbf{x}, \mathbf{y})$ is true, therefore there is a term P_ℓ that is satisfied in $\varphi(\mathbf{x}, \mathbf{y})$. We claim that $K \cup \{p_\ell\}$ is a clique, contradicting the maximality of K. To see this, observe that $x_i \in K$ if and only if the value of variable x_i is true in $\mathbf{x}$. Therefore K contains those vertices that correspond to true literals in the assignment $(\mathbf{x}, \mathbf{y})$. This assignment satisfies term P_ℓ, thus these literals do not contradict P_ℓ. By the construction, these vertices are connected to p_ℓ, and $K \cup \{p_\ell\}$ is indeed a clique.

Now assume that G is 2-clique-colored, it can be assumed that p_1 is white. Since $\{p_\ell, p'_\ell\}$ and $\{p'_\ell, p_{\ell+1}\}$ are maximal cliques, p_ℓ is white and p'_ℓ is black for every $1 \leq \ell \leq q$. Because $\{p'_q, \overline{y}'_m\}$ is also a maximal clique, $\overline{y}'_m$ is black. Since $\{y_j, y'_j\}$, $\{y'_j, \overline{y}_j\}$, $\{\overline{y}_j, \overline{y}'_j\}$, $\{\overline{y}'_j, y_{j+1}\}$ are maximal cliques, we have that y_j and

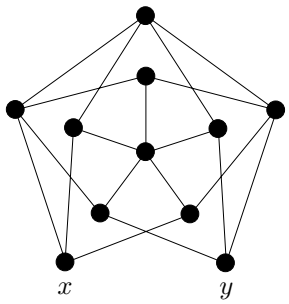


Figure 6.3: The graph D_4, which is the Mycielski graph M_4 (the Grötzch graph) minus the edge xy.

$\overline{y}_j$ are colored black, y'_j and $\overline{y}'_j$ are colored white for every $1 \leq j \leq m$. Furthermore, $\{x_i, x'_i\}$, $\{x'_i, \overline{x}'_i\}$, $\{\overline{x}'_i, \overline{x}_i\}$ are also maximal cliques, thus x_i and $\overline{x}_i$ have different color.

Define the vector $\mathbf{x}$ as variable x_i is true if and only if the color of vertex x_i is colored black. We show that $\varphi(\mathbf{x}, \mathbf{y})$ is true for every $\mathbf{y}$. To see that $\varphi(\mathbf{x}, \mathbf{y})$ is true for the given assignment it is sufficient to see that there is a $1 \leq \ell \leq q$ such that vertex p_ℓ is connected to all true literals. This is obvious, since the true literals have color black and they form a clique, so this is not a maximal clique. The only way to increase it is by adding a p_ℓ. ■

We show that the hardness result holds for every $k > 2$ as well. The proof is by reducing k-clique-colorability to $(k+1)$-clique-colorability. The reduction uses the Mycielski graphs as gadgets.

For every $k \geq 2$, the construction of Mycielski gives a triangle-free graph M_k with chromatic number k. For completeness, we recall the construction here. For $k = 2$, the graph M_2 is a K_2. To obtain the graph M_{k+1}, take a copy of M_k, let $v_1, v_2, \ldots, v_n$ be its vertices. Add $n+1$ new vertices $u_1, u_2, \ldots, u_n, w$, connect u_i to the neighbors of v_i in M_k, and connect w to every vertex u_i. It can be shown that M_{k+1} is triangle-free, and $\chi(M_{k+1}) = \chi(M_k) + 1$. Moreover, M_k is edge-critical (see [Lov93, Problem 9.18]): for every edge e of M_k, the graph $M_k \setminus e$ is $(k-1)$-colorable. Remove an arbitrary edge $e = xy$ of M_k, denote by D_k the resulting graph (see D_4 on Figure 6.3). It follows that in every $(k-1)$-coloring if D_k, the vertices x and y have the same color, otherwise it would be a proper $(k-1)$-coloring of M_k.

The following corollary shows that k-Clique-Coloring remains Σ_2^p-complete for every $k > 2$ (note that the problem becomes trivial for $k = 1$).

Corollary 6.2.2. *k-Clique-Coloring is Σ_2^p-complete for every $k \geq 2$.*

Proof. For every $k \geq 2$, we give a polynomial time reduction from k-Clique-Coloring to $(k+1)$-Clique-Coloring. By Theorem 6.2.1, 2-Clique-Coloring is Σ_2^p-complete, thus the theorem follows by induction.

Let G be a graph with n vertices $v_1, v_2, \ldots, v_n$. Add $n+1$ vertices $u_1, u_2, \ldots, u_n, w$, and connect every vertex u_i with v_i. Furthermore, add n copies of the graph D_{k+2} such that vertex x of the ith copy is identified with w, and vertex y is identified with u_i. Denote the new vertices added to G by W, observe that every vertex in W is simple. We claim that the resulting graph G' is $(k+1)$-clique-colorable if and only if G is k-clique-colorable.

Assume first that there is a $(k+1)$-clique-coloring ψ of G', we show that it induces a k-clique-coloring of G. By Prop. 6.1.3, $G[W]$ is $(k+1)$-vertex-colored in ψ, thus the construction of the graph D_{k+2} implies that $\psi(w) = \psi(u_1) = \cdots = \psi(u_n) = \alpha$, and none of the vertices $v_1, v_2, \ldots, v_n$ has color α. Hence ψ uses at most k colors on $G = G' \setminus W$, and by Prop. 6.1.1, it is a proper k-clique-coloring.

On the other hand, if there is a proper k-clique-coloring of G, then color the vertices $u_1, \ldots, u_n, w$ with color $k+1$, and extend this coloring to the copies of the graph D_{k+2} in such a way that the coloring is a proper $(k+1)$-vertex-coloring on every copy of D_{k+2}. By Prop. 6.1.2, this results in a proper $(k+1)$-clique coloring of G', since each vertex in W has different color from its neighbors. ■

6.3 Clique choosability

We define the list coloring version of clique coloring. In a k-clique-coloring, the vertices can use only the colors 1, 2, ..., k. In the list coloring version, each vertex v has a set $L(v)$ of k admissible colors, the color of the vertex has to be selected from this set. A list assignment L is a *k-list assignment* if the size of $L(v)$ is k for every vertex v. We say that a graph $G(V, E)$ is *k-clique-choosable*, if for every k-list assignment L: $V \to 2^{\mathbb{N}}$ there is a proper clique coloring ψ of G with $\psi(v) \in L(v)$.

k-Clique-Choosability

Input: A graph $G(V, E)$

Question: Is G k-clique-choosable?

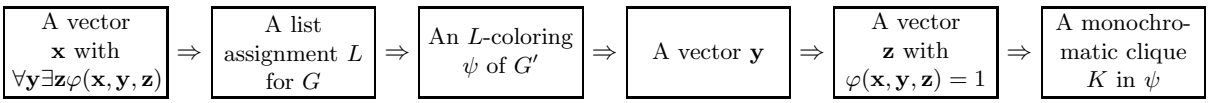
Rubin [ERT80] characterized 2-vertex-choosable graphs, which can be turned into a polynomial time algorithm for the 2-vertex-choosability problem. Therefore 2-vertex-coloring and 2-vertex-choosability have the same complexity, both can be solved in polynomial time. However, 3-vertex-choosability is harder than 3-vertex-coloring: the former is Π_2^p-complete [Gut96], whereas the latter is "only" NP-complete. The situation is different in the case of clique coloring: we show that the 2-Clique-Choosability problem is more difficult than 2-Clique-Coloring, it lies one level higher in the polynomial hierarchy.

Theorem 6.3.1. *2-Clique-Choosability is Π_3^p-complete.*

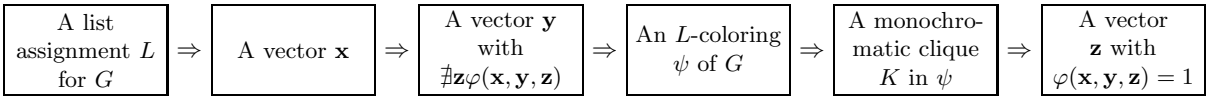
Proof. Notice first that deciding whether a graph has a proper clique coloring with the given lists is in Σ_2^p: a proper coloring is a certificate proving that such a coloring exists, and verifying this certificate is in coNP. Therefore k-Clique-Choosability is in Π_3^p: if the graph is not k-clique-choosable, then an uncolorable list assignment exists, which is a Σ_2^p certificate that the graph is not k-clique-choosable.

We prove that the 2-Clique-Choosability problem is Π_3^p-complete by reducing QSAT_3 to the complement of 2-Clique-Choosability. That is, for every 3CNF formula $\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$, a graph G is constructed in such a way that G is *not* 2-clique-choosable if and only if $\exists\mathbf{x}\forall\mathbf{y}\exists\mathbf{z}\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ holds.

Before describing the construction of the graph G in detail, we present the outline of the proof. Assume first that a vector $\mathbf{x}$ exists with $\forall\mathbf{y}\exists\mathbf{z}\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$, it has to be shown that G is not 2-clique-choosable. Based on this vector $\mathbf{x}$, we define a 2-list assignment L of G, and claim that G is not clique colorable with this assignment. If, on the contrary, such a coloring ψ exists, then a vector $\mathbf{y}$ is defined based on this coloring. By assumption, there is a vector $\mathbf{z}$ with $\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ true. Based on vectors $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$, we construct a clique K, which is monochromatic in ψ, a contradiction. This direction of the proof is summarized in the following diagram:



The other direction is to prove that if G is not 2-clique-choosable, then $\exists\mathbf{x}\forall\mathbf{y}\exists\mathbf{z}\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$. The outline of this direction is the following. Given an uncolorable 2-list assignment L, we define a vector $\mathbf{x}$. Assume indirectly that there is a vector $\mathbf{y}$ with $\nexists\mathbf{z}\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$. Based on this vector $\mathbf{y}$, an L-coloring ψ of G is defined. By assumption, ψ is not a proper clique coloring, thus it contains a monochromatic maximal clique K. Based on K, a vector $\mathbf{z}$ is constructed satisfying $\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$, a contradiction. (The actual proof is slightly more complicated than this: because of technical reasons, not one vector $\mathbf{x}$, but two vectors $\mathbf{x}^1, \mathbf{x}^2$ are defined based on the list assignment L). The summary of this direction:



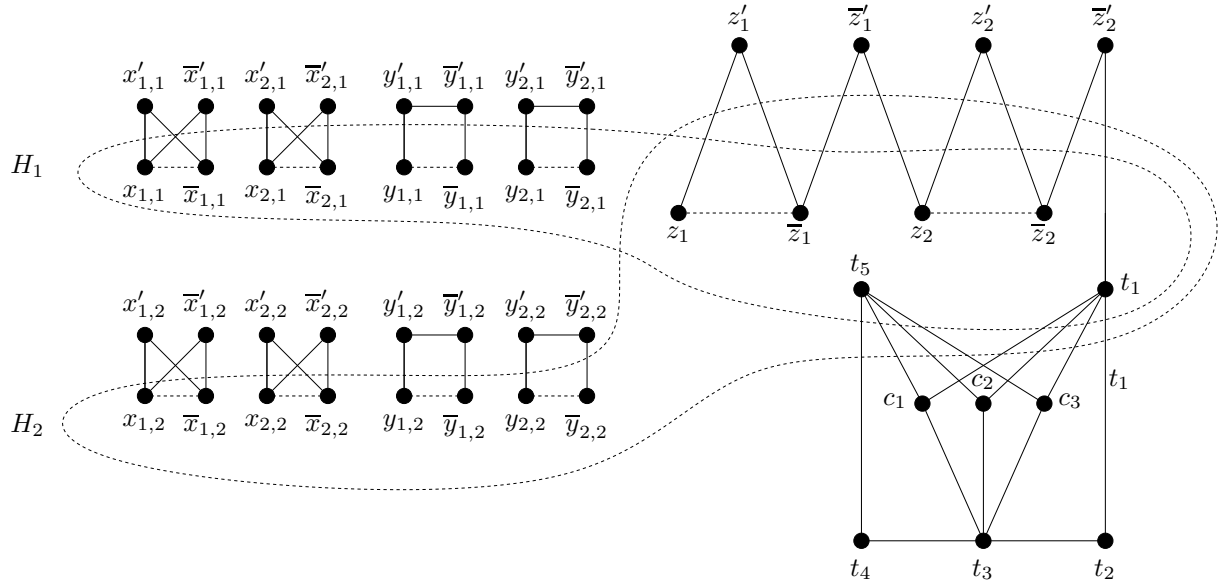


Figure 6.4: The structure of graph G in the proof of Theorem 6.3.1 for $n = 2$, $m = 2$, $p = 2$, $q = 3$. The sets H_1 and H_2 almost form a clique, the pairs connected by dashed lines are not neighbors. For the sake of clarity, the edges connecting the clause vertices c_1, c_2, c_3 to the vertices representing the literals are omitted.

Now we define graph G. For every variable x_i $(1 \leq i \leq n)$, the graph G contains two cycles of 4 vertices: $x_{i,1}$, $x'_{i,1}$, $\overline{x}_{i,1}$, $x''_{i,1}$ and $x_{i,2}$, $x'_{i,2}$, $\overline{x}_{i,2}$, $x''_{i,2}$ (see Figure 6.4). For every variable y_j $(1 \leq j \leq m)$, there are two paths of 4 vertices: $y_{j,1}$, $y'_{j,1}$, $\overline{y}'_{j,1}$, $\overline{y}_{j,1}$ and $y_{j,2}$, $y'_{j,2}$, $\overline{y}'_{j,2}$, $\overline{y}_{j,2}$. For every variable z_k $(1 \leq k \leq p)$, there are 4 vertices z_k, z'_k, $\overline{z}_k$, $\overline{z}'_k$, these $4p$ vertices form a path z_1, z'_1, $\overline{z}_1$, $\overline{z}'_1$, z_2, z'_2, $\ldots$, z_p, z'_p, $\overline{z}_p$, $\overline{z}'_p$. There is a path of 5 vertices t_1, t_2, t_3, t_4, t_5, and for every clause C_ℓ of φ $(1 \leq \ell \leq q)$, there is a vertex c_ℓ, which is connected to t_3. Vertex t_1 is connected to $\overline{z}'_p$. The edges defined so far are all flat edges in G, they form the core of G. The following edges appear in cliques greater than 2. Vertex c_ℓ is connected to t_1, t_5, and to every vertex that corresponds to a literal *not* satisfying clause C_ℓ. That is, if variable x_i does not appear in clause C_ℓ, then c_i is connected to $x_{i,1}$, $\overline{x}_{i,1}$, $x_{i,2}$, $\overline{x}_{i,2}$, and if variable x_i appears in C_ℓ (but $\overline{x}_i$ does not), then c_i is connected to $\overline{x}_{i,1}, \overline{x}_{i,2}$. Moreover, the $2n + 2m + 2p + 2$ vertices $H_1 = \{x_{i,1}, \overline{x}_{i,1}, y_{j,1}, \overline{y}_{j,1}, z_k, \overline{z}_k, t_1, t_5\}$ almost form a clique, the $n + m + p$ edges $x_{i,1}\overline{x}_{i,1}$, $y_{i,1}\overline{y}_{i,1}$, $z_k\overline{z}_k$ are missing from the graph. Similarly the $2n + 2m + 2p + 2$ vertices $H_2 = \{x_{i,2}, \overline{x}_{i,2}, y_{j,2}, \overline{y}_{j,2}, z_k, \overline{z}_k, t_1, t_5\}$ almost form a clique, the $n+m+p$ edges $y_{i,2}\overline{y}_{i,2}$, $x_{i,2}\overline{x}_{i,2}$, $z_k\overline{z}_k$ are missing from the graph. This completes the description of the graph G. Notice that every maximal clique of G greater than 2 contains the vertices t_1 and t_5, and it contains exactly one of z_k and $\overline{z}_k$.

Assume first that there is an $\mathbf{x} \in \{0,1\}^n$ such that $\forall \mathbf{y} \exists \mathbf{z} \varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ holds, we show that there is a list assignment L to the vertices of G such that no proper clique coloring is possible with these lists. We make the following list assignments:

- If x_i is true in $\mathbf{x}$, then set $L(x_{i,1}) = \{1,2\}$, $L(x'_{i,1}) = \{2,3\}$, $L(\overline{x}_{i,1}) = \{1,3\}$, $L(\overline{x}'_{i,1}) = \{1,2\}$. This list assignment forces $x_{i,1}$ to color 1: giving color 2 to $x_{i,1}$ would imply that there is color 3 on $x'_{i,1}$ and there is color 1 on $\overline{x}'_{i,1}$, which means that there is no color left for $\overline{x}_{i,1}$.

- If x_i is true, then set $L(x_{i,2}) = \{1,2\}$, $L(x'_{i,2}) = \{1,3\}$, $L(\overline{x}_{i,2}) = \{2,3\}$, $L(\overline{x}'_{i,2}) = \{1,2\}$, forcing $x_{i,2}$ to color 2.

- If x_i is false, then set $L(x_{i,1}) = \{1,3\}$, $L(x'_{i,1}) = \{2,3\}$, $L(\overline{x}_{i,1}) = \{1,2\}$, $L(\overline{x}'_{i,1}) = \{1,2\}$, forcing $\overline{x}_{i,1}$ to color 1.
- If x_i is false, then set $L(x_{i,2}) = \{2,3\}$, $L(x'_{i,2}) = \{1,3\}$ $L(\overline{x}_{i,2}) = \{1,2\}$, $L(\overline{x}'_{i,2}) = \{1,2\}$, forcing $\overline{x}_{i,2}$ to color 2.

Set $L(v) = \{1,2\}$ for every other vertex v. We claim that there is no proper clique coloring with these list assignments.

Assume that, on the contrary, there is a proper clique coloring ψ. Since the edges $t_i t_{i+1}$ $(1 \leq i \leq 4)$ are flat edges, the vertices t_1, t_3, t_5 have the same color, suppose first that it is color 1. Then every vertex c_ℓ has color 2 (edge $t_3 c_\ell$ is a flat edge) and the vertices z_k, $\overline{z}_k$ $(1 \leq k \leq p$ have color 1 (every edge of the path from z_1 to t_1 is a flat edge and the vertices in the path have the list $\{1,2\}$). For every $1 \leq j \leq m$, exactly one of $y_{j,1}$ and $\overline{y}_{j,1}$ have color 1, since edges $y_{j,1}y'_{j,1}$, $y'_{j,1}\overline{y}'_{j,1}$, $\overline{y}'_{j,1}\overline{y}_{j,1}$ are flat edges. Define the vector $\mathbf{y} \in \{0,1\}^m$ such that variable y_j is true if and only if $y_{j,1}$ has color 1. By assumption, there is a vector $\mathbf{z}$ such that $\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ holds. Based on $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$, we can define a clique K of G as follows:

- $x_{i,1} \in K$ iff x_i is true,
- $\overline{x}_{i,1} \in K$ iff x_i is false,
- $y_{j,1} \in K$ iff y_j is true,
- $\overline{y}_{j,1} \in K$ iff y_j is false,
- $z_k \in K$ iff z_k is true,
- $\overline{z}_k \in K$ iff z_k is false,
- $t_1, t_5 \in K$.

Notice that every vertex in clique K has color 1: if x_i is true (resp. false) then the list assignments force $x_{i,1}$ (resp. $\overline{x}_{i,1}$) to color 1. Moreover, exactly one of $y_{j,1}$ and $\overline{y}_{j,1}$ have color 1, and the definition of $\mathbf{y}$ and K implies that from these two vertices, the one with color 1 is selected into K. By assumption, ψ is a proper clique coloring, therefore K is not a maximal clique. It is clear that only a vertex c_ℓ can extend K to a larger clique, thus there is a c_ℓ such that $K \cup \{c_\ell\}$ is also a clique. However, by the construction, this means that in $\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$, no variable satisfies clause C_ℓ, a contradiction.

If vertices t_1, t_3, t_5 have color 2, then we proceed similarly. Vector $\mathbf{y}$ is defined such that y_j is true if and only if $y_{i,2}$ has color 2. In this case the definition of K is similar, but now K contains the vertices $x_{i,2}, \overline{x}_{i,2}, y_{j,2}, \overline{y}_{j,2}$ instead of $x_{i,1}, \overline{x}_{i,1}, y_{j,1}, \overline{y}_{j,1}$. Every vertex in K has color 2: the list assignment ensures that if variable x_i is true, then vertex $x_{i,2}$ has color 2. The other parts of the proof remain the same.

To prove the other direction, we show that if $\exists \mathbf{x} \forall \mathbf{y} \exists \mathbf{z} \varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ does not hold, then there is proper clique coloring for every list assignment L. The core of G is the disjoint union of trees and even cycles, hence it is 2-choosable. In particular, the path $T = t_1, t_2, \ldots, t_5$ can be colored with the lists, and every coloring of T can be extended to an L-coloring of the core of G. If there is a coloring of this path such that t_1 and t_5 have different color, then this can be extended to a proper vertex coloring of the core of G. Moreover, this coloring is also a proper clique coloring of G: every maximal clique greater than 2 contains both t_1 and t_5, thus the clique contains at least two different colors. Therefore, if the list assignment is uncolorable, then it can be assumed that in every coloring of the path T, vertices t_1 and t_5 receive the same color. It is easy to verify that this is only possible if all 5 vertices of T have the same list, say $\{1,2\}$. In this case we give color 1 to the vertices t_1, t_3, t_5, and color 2 to vertices t_2, t_4. Color the path from z_1 to $\overline{z}'_p$ with arbitrary colors. Furthermore, give a color different from 1 to every vertex c_ℓ.

The rest of the graph is colored as follows. The length 4 cycle formed by the vertices $x_{i,1}, x'_{i,1}, \overline{x}_{i,1}, x''_{i,1}$ is 2-choosable, thus it can be colored with the given lists. Moreover, it can be verified by a simple case

analysis that we can choose a coloring that does not assign color 1 to both $x_{i,1}$ and $\overline{x}_{i,1}$. The vertices $x_{i,2}, x'_{i,2}, \overline{x}_{i,2}, x''_{i,2}$ are colored similarly. Define the vectors $\mathbf{x}^1, \mathbf{x}^2 \in \{0,1\}^n$ such that variable x_i^1 (resp. x_i^2) is true if and only if vertex $x_{i,1}$ (resp, $x_{i,2}$) has color 1.

By assumption, $\exists\mathbf{x}\forall\mathbf{y}\exists\mathbf{z}\varphi(\mathbf{x},\mathbf{y},\mathbf{z})$ does not hold, thus $\forall\mathbf{y}\exists\mathbf{z}\varphi(\mathbf{x}_1,\mathbf{y},\mathbf{z})$ and $\forall\mathbf{y}\exists\mathbf{z}\varphi(\mathbf{x}_2,\mathbf{y},\mathbf{z})$ are false in particular. Therefore there are vectors $\mathbf{y}^1, \mathbf{y}^2 \in \{0,1\}^m$ such that $\exists\mathbf{z}\varphi(\mathbf{x}^1,\mathbf{y}^1,\mathbf{z})$ does not hold and $\exists\mathbf{z}\varphi(\mathbf{x}^2,\mathbf{y}^2,\mathbf{z})$ does not hold. Based on the vectors $\mathbf{y}_1$, $\mathbf{y}_2$, we continue the coloring of G. The path $y_{i,1}$, $y'_{i,1}$, $\overline{y}'_{i,1}$, $\overline{y}_{i,1}$ can be colored with the lists. Moreover, this path has a coloring such that $y_{i,1}$ does not have color 1, and it has another coloring where $\overline{y}_{i,1}$ does not have color 1. If y_i^1 is true (resp. false), then color the path in such a way that $\overline{y}_{i,1}$ (resp. $y_{i,1}$) has a color different from 1. Color the path $y_{i,2}$, $y'_{i,2}$, $\overline{y}'_{i,2}$, $\overline{y}_{i,2}$ analogously. We claim that this coloring is a proper clique coloring. Since the coloring is a proper vertex coloring of the core of G, it is sufficient to check the maximal cliques greater than 2. Since every such clique K contains the vertices t_1 and t_5 having color 1, thus every vertex in K has color 1. This implies that K does not contain any of the vertices c_ℓ, since we have assigned colors different from 1 to these vertices. For every $1 \leq k \leq p$, clique K contains exactly one of z_k and $\overline{z}_k$. Define the vector $\mathbf{z} \in \{0,1\}^p$ such that variable z_k is true if and only if $z_k \in K$. Clique K contains exactly one of $x_{i,1}$, $\overline{x}_{i,1}$, $x_{i,2}$, $\overline{x}_{i,2}$, suppose that it contains $x_{i,1}$ or $\overline{x}_{i,1}$ (the other case is handled analogously). Since K contains only vertices with color 1, and at most one of $x_{i,1}$ and $\overline{x}_{i,1}$ has color 1, thus $x_{i,1} \in K$ if and only if x_i^1 is true. Similarly, K contains exactly one of $y_{i,1}$ and $\overline{y}_{i,1}$, more precisely, $y_{i,1} \in K$ if and only if y_i^1 is true. To arrive to a contradiction, we show that $\varphi(\mathbf{x}^1,\mathbf{y}^1,\mathbf{z})$ is true. Suppose that clause C_ℓ is not satisfied by this variable assignment. The vertices in K correspond to the true literals in the variable assignment $\mathbf{x}^1,\mathbf{y}^1,\mathbf{z}$, therefore by the construction, c_ℓ is connected to every vertex in K, contradicting the assumption that K is a maximal clique. ■

The k-Clique-Choosability problem remains Π_3^p-complete for every $k > 2$. The proof is similar to the proof of Corollary 6.2.2: the case k is reduced to the case $k+1$ by attaching some special graphs. However, here we attach complete bipartite graphs instead of Mycielski graphs.

Lemma 6.3.2. *There is k-vertex-choosable bipartite graph B_k with a distinguished vertex x that has the following property: for every color c there is a k-list assignment such that in every list coloring x receives color c.*

Proof. We claim that the complete bipartite graph $B_k = K_{k,k^k-1}$ is such a graph, with $x \in V_1$ being a vertex of the smaller class V_1. To see that B_k is k-vertex-choosable, assume first that $L(u) \cap L(v) \neq \emptyset$ for some $u, v \in V_1$. In this case the k vertices in V_1 can be colored such that they receive at most $k-1$ distinct colors, thus every vertex $w \in V_2$ can be given a color from $L(w)$ that is not used by the vertices in V_1. If the lists in V_1 are disjoint, then V_1 can be colored in k^k different ways, every such coloring assigns a different set of k colors to the vertices in V_1. A coloring of V_1 can be extended to V_2 unless there is a vertex $w \in V_2$ whose list contains exactly the k colors used by V_1. Since there are only $k^k - 1$ vertices in V_2, they can exclude at most $k^k - 1$ colorings of V_1, thus at least one of the k^k different colorings of V_1 can be extended to V_2.

On the other hand, let $V_1 = \{v_1, \ldots, v_k\}$ and $L(v_i) = \{c_{i,1}, c_{i,2}, \ldots, c_{i,k}\}$. There are k^k sets of the form $\{c_{1,i_1}, c_{2,i_2}, \ldots, c_{k,i_k}\}$ with $1 \leq i_1, i_2, \ldots, i_k \leq k$. Assign these sets, with the exception of $\{c_{1,1}, c_{2,1}, \ldots, c_{k,1}\}$, to the vertices in V_2. It is easy to see that with these list assignments, every coloring gives color $c_{i,1}$ to vertex v_i. Therefore $x = v_1$ and $c = c_{1,1}$ satisfies the requirements. ■

Corollary 6.3.3. *For every $k \geq 2$, k-Clique-Choosability is Π_3^p-complete.*

Proof. For every $k \geq 2$, we give a polynomial time reduction from k-Clique-Choosability to $(k+1)$-Clique-Choosability. By Theorem 6.3.1, 2-Clique-Choosability is Π_3^p-complete, thus the theorem follows by induction.

Let $G(V,E)$ be a graph with n vertices $v_1, v_2, \ldots, v_n$. Add n disjoint copies of the graph B_{k+1} (Lemma 6.3.2) such that vertex x_i, which is the distinguished vertex x of the ith copy, is connected

to v_i. Denote by W the new vertices added to G, observe that every vertex in W is simple (B_{k+1} is bipartite, thus it does not contain triangles). We claim that the resulting graph $G'(V \cup W, E')$ is $(k+1)$-clique-choosable if and only if G is k-clique-choosable.

Assume first that G' is $(k+1)$-clique-choosable, we show that G is k-clique-choosable. Let L be an arbitrary k-assignment of G. Let c be a color not appearing in L. Define the $(k+1)$-assignment L' as $L'(v) = L(v) \cup \{c\}$ for every $v \in V$, and extend L' to W (i.e., to the copies of B_{k+1}) in such a way that in every list coloring of G', the vertices x_i receive the color c. By assumption, G' has a clique coloring ψ with the lists L'. By Prop. 6.1.3, ψ is a proper vertex coloring of W, therefore $\psi(x_i) = c$ for every $1 \leq i \leq n$. Thus $\psi(v_i) \neq c$ and $\psi(v_i) \in L(v_i)$ follow, hence ψ induces a list coloring of G. Moreover, by Prop. 6.1.1, ψ is a proper clique coloring of G, proving this direction of the reduction.

Now assume that G is k-clique-choosable, it has to be shown that G' is $(k+1)$-clique-choosable. Let L be a $k+1$ assignment of $V \cup W$. Since B_{k+1} is $(k+1)$-choosable, every copy of B_{k+1} can be colored with these lists, let ψ be this coloring of W. Define the k-assignment L' of V as $L'(v_i) = L(v_i) \setminus \{\psi(x_i)\}$ if $\psi(x_i) \in L(v_i)$, otherwise let $L'(v_i)$ an arbitrary k element subset of $L(v_i)$. By assumption, there is a proper clique coloring of V with the lists L', extend ψ to V with these assignment of colors. By Prop. 6.1.2, ψ is also a proper clique coloring of G'. ■

6.4 Hereditary clique coloring

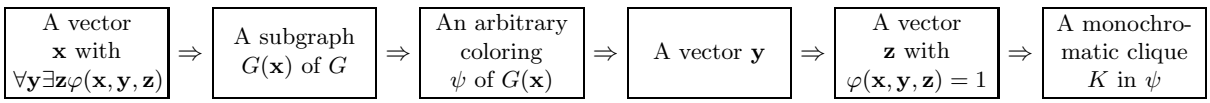
Graph G is *hereditary k-clique-colorable* if every induced subgraph of G is k-clique-colorable. Since clique coloring is not a hereditary property in general, an induced subgraph of a k-clique-colorable graph G is not necessarily k-clique-colorable, thus hereditary k-clique-colorability is not the same as k-clique-colorability. The main result of this section is that we show that Hereditary k-Clique-Coloring is Π_3^p-complete for every $k \geq 3$.

Hereditary k-Clique-Coloring

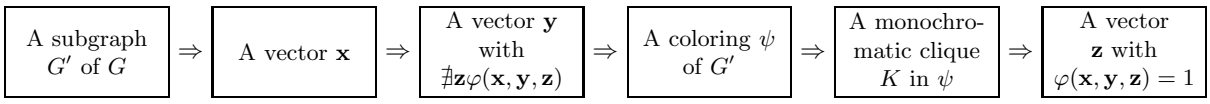
Input: A graph $G(V, E)$

Question: Is it true that every induced subgraph of G is k-clique-colorable?

The proof follows the same general framework as the proof of Theorem 6.3.1, but selecting an induced subgraph of G plays here the same role as selecting a list assignment in that proof. To show that $\exists \mathbf{x} \forall \mathbf{y} \exists \mathbf{z} \varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ implies that G is *not* hereditary 3-clique-colorable, assume that a vector $\mathbf{x}$ exists with $\forall \mathbf{y} \exists \mathbf{z} \varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$. Based on this vector $\mathbf{x}$, we select an induced subgraph $G(\mathbf{x})$ of G. If subgraph $G(\mathbf{x})$ has a 3-clique-coloring ψ, then a vector $\mathbf{y}$ can be defined based on ψ. By assumption, there is a vector $\mathbf{z}$ such that $\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is true. We arrive to a contradiction by showing that vectors $\mathbf{x}, \mathbf{y}, \mathbf{z}$ can be used to find a monochromatic maximal clique K in ψ. The overview of this direction:



The proof of the reverse direction is much more delicate. We have to show that if there is an induced subgraph G' of G that is not 3-clique-colorable, then $\exists \mathbf{x} \forall \mathbf{y} \exists \mathbf{z} \varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ holds. If G' is a subgraph $G(\mathbf{x})$ for some vector $\mathbf{x}$ (as defined by the first direction of the proof), then we proceed as follows. Assume that $\exists \mathbf{x} \forall \mathbf{y} \exists \mathbf{z} \varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ does not hold, then there is vector $\mathbf{y}$ with $\nexists \mathbf{z} \varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$. Based on this vector $\mathbf{y}$, one can define a 3-coloring ψ of G'. By assumption, G' is not 3-clique-colorable, thus ψ contains a monochromatic maximal clique K. Now we can a find a vector $\mathbf{z}$ satisfying $\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$, a contradiction.



However, it might be possible that the uncolorable subgraph G' does not correspond to a subgraph $G(\mathbf{x})$ for any vector $\mathbf{x}$. In this case the above proof does not work, we cannot define $\mathbf{x}$ based on the subgraph. In order to avoid this problem, we implement a delicate "self-destruct" mechanism, which ensures that every such nonstandard subgraph can be easily 3-clique-colored. This will be done the following way. We start with a graph G_0, and G is obtained by attaching several gadgets to G_0. Graph G_0 is easy to color, but a coloring of G_0 can be extended to the gadgets only if the coloring of G_0 satisfies certain requirements (some pairs of vertices have the same color, some pairs have different colors). However, if G' is a nonstandard subgraph of G (e.g., a vertex is missing from G' that cannot be missing in any subgraph $G(\mathbf{x})$), then the gadgets are "turned off," and every coloring of G_0 can be extended easily to G'. The important thing is that a single missing vertex will turn off every gadget. We define these gadgets in the following two lemmas.

Lemma 6.4.1. *There is a graph Z_1 (called the γ-*copier*), with distinguished vertices α, β, γ, satisfying the following properties:*

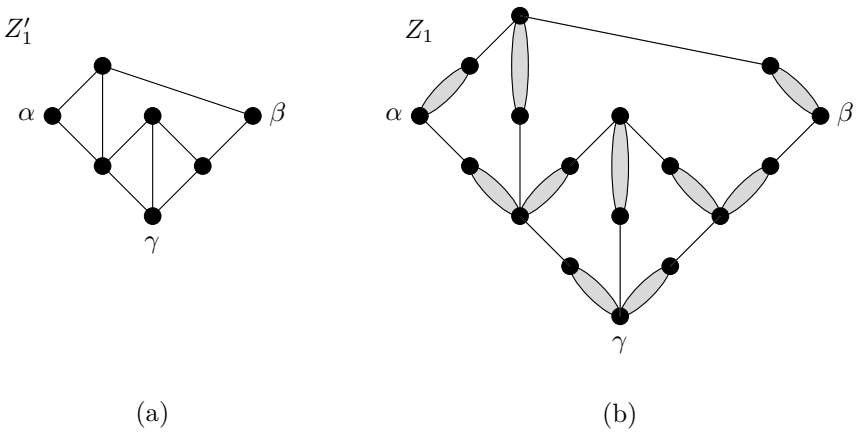
1. *Z_1 is triangle free.*
2. *In every 3-vertex-coloring of Z_1, vertices α and β receive the same color.*
3. *Z_1 can be 3-vertex-colored such that α, β, γ receive the same color, and it can be 3-vertex-colored such that α and γ receive different colors.*
4. *In $Z_1 \setminus \gamma$, every assignment of colors to α and β can be extended to a proper 3-vertex-coloring.*
5. *Vertices α, β, γ form an independent set in Z_1.*

Proof. The graph Z_1' shown on Figure 6.5a is not triangle free, but it can be proved by inspection that Z_1' satisfies properties 2–5. The graph Z_1 is created from Z_1' as follows. Every edge $e = uv$ is replaced by a new vertex e that is connected to u. Furthermore, a copy of D_4 (see Figure 6.3) is added to the graph such that the distinguished vertices x and y are identified with vertices e and v. It is clear that every 3-coloring of Z_1 induces a coloring of Z_1': vertices u and v have different colors, since vertices e and v have the same color in every 3-coloring of Z_1 (because of the properties of the graph D_4) and e, u are neighbors. Moreover, every 3-coloring of Z_1' can be extended to a coloring of Z_1. Therefore properties 2–5 hold for Z_1 as well. ■

Thus the γ-copier ensures that α and β have the same color, but deleting γ turns off the gadget. The gadget defined by the following lemma is similar, but γ is replaced by several vertices, deleting any of them turns off the gadget.

Lemma 6.4.2. *For every $n \geq 1$, there is a graph Z_n (called an n-*copier*), with distinguished vertices $\alpha, \beta, \gamma_1, \gamma_2, \ldots, \gamma_n$, satisfying the following properties:*

1. *Z_n is triangle free.*
2. *In every 3-vertex-coloring of Z_n, vertices α and β receive the same colors.*
3. *Every coloring of the vertices $\alpha, \beta, \gamma_1, \gamma_2, \ldots, \gamma_n$ can be extended to a 3-vertex-coloring of Z_n, if α and β have the same color.*
4. *In $Z_n \setminus \gamma_i$ ($1 \leq i \leq n$), every assignment of colors to $\alpha, \beta, \gamma_1, \gamma_2, \ldots, \gamma_{i-1}, \gamma_{i+1}, \ldots, \gamma_n$ can be extended to a proper 3-vertex-coloring.*
5. *Vertices $\alpha, \beta, \gamma_1, \gamma_2, \ldots, \gamma_n$ form an independent set in Z_n.*



(a) (b)

Figure 6.5: The γ-copier Z_1. Figure (a) shows the graph Z_1' that served as base for constructing Z_1. On Figure (b), every shaded ellipse is a copy of D_4.

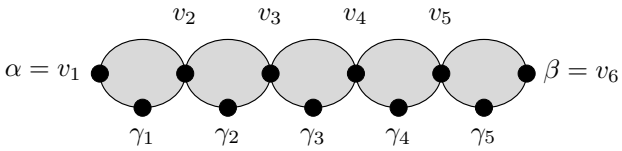


Figure 6.6: The n-copier Z_n. Every shaded ellipse is a copy of the γ-copier.

Proof. Graph Z_n is created by concatenating n copies of the graph Z_1 (see Lemma 6.4.1). Take $n+1$ vertices $v_1, v_2, \ldots, v_{n+1}$ and add n copies of Z_1 such that vertex α of the ith copy is identified with vertex v_i, and vertex β is identified with vertex v_{i+1} (see Figure 6.6). Let $\alpha = v_1$, $\beta = v_{n+1}$, and let γ_i be vertex γ of the ith copy.

It is clear that Z_n is triangle free. Property 2 holds, since by Property 2 of Lemma 6.4.1, vertices v_i and v_{i+1} have the same color for $1 \leq i \leq n$. To see that Property 3 holds, observe that the coloring can be extended on every copy of Z_1 to a 3-vertex-coloring such that the vertices v_i $(1 \leq i \leq n+1)$ have the same color. Property 4 follows from Property 3 if the same color is assigned to α and β. Otherwise assign the same color to $v_1 = \alpha, v_2, \ldots, v_i$, and the same color to $v_{i+1}, \ldots, v_{n+1} = \beta$. This coloring can be extended to a 3-vertex-coloring on every copy of Z_1: for every copy but the ith, the distinguished vertices α and β have the same color, thus there is such a coloring by Property 3 of Lemma 6.4.1. On the other hand, for the ith copy, the distinguished vertex γ_i is missing, thus there is such an extension by Property 4 of Lemma 6.4.1. ■

The *n-edge* is obtained from the n-copier by renaming vertex β to β', and connecting a new vertex β to β'. It has the same properties as the n-copier, except that in Property 2 and 3 of Lemma 6.4.2, vertices α and β must have *different* colors.

Now we are ready to prove the main result of the section:

Theorem 6.4.3. *Hereditary 3-Clique-Coloring is Π_3^p-complete.*

Proof. The problem is in Π_3^p: if G is not hereditary 3-clique-colorable, then it has an induced subgraph G' that is not 3-clique-colorable, which can serve as a certificate proving that G is not hereditary 3-clique-colorable. Checking 3-clique-colorability is in Σ_2^p, thus verifying this certificate is in Σ_2^p, implying that the problem is in Π_3^p.

The Π_3^p-completeness of the problem is proved by reducing the Σ_3^p-complete problem QSAT_3 to the complement of Hereditary 3-Clique-Choosability. That is, for every 3CNF formula $\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$, a graph G is constructed in such a way that G is *not* hereditary 3-clique-colorable if and only if $\exists \mathbf{x} \forall \mathbf{y} \exists \mathbf{z} \varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ holds.

The graph $G(V, E)$ consists of a graph $G_0(V_0, E_0)$ and some added n-copiers and n-edges. G_0 contains

- 5 vertices $x_i,\ x'_i,\ \overline{x}_i,\ \overline{x}'_i,\ x^*_i$ for every variable x_i $(1 \leq i \leq n)$,
- 2 vertices $y_j,\ \overline{y}_j$ for every variable y_j $(1 \leq j \leq m)$,
- 2 vertices $z_k,\ \overline{z}_k$ for every variable z_k $(1 \leq k \leq p)$,
- a vertex c_ℓ for every clause C_ℓ $(1 \leq \ell \leq q)$,
- $2n$ vertices $t_i,\ t'_i$ $(1 \leq i \leq n)$,
- 3 vertices $f_1,\ f_2,\ f_3$.

Graph G_0 has the following edges. The $4n + 2m + 2p + 1$ vertices x_i, $\overline{x}_i$, y_j, $\overline{y}_j$, z_k, $\overline{z}_k$, t_i, t'_i, f_1 $(1 \leq i \leq n,\ 1 \leq j \leq m,\ 1 \leq k \leq p)$ almost form a clique, except that the edges $x_i\overline{x}_i$, $y_j\overline{y}_j$, $z_k\overline{z}_k$ are missing. For every $1 \leq i \leq n$, the 3 vertices $x_i,\ x'_i,\ x^*_i$, and the 3 vertices $\overline{x}_i,\ \overline{x}'_i,\ x^*_i$ form a triangle. Every vertex c_ℓ is connected to those vertices that correspond to literals *not* satisfying clause C_ℓ. Furthermore, vertex c_ℓ is also connected to vertices $f_1,\ t_i,\ t'_i$ $(1 \leq i \leq n)$.

To obtain the graph G, several n-copiers and n-edges are added to G_0. Let S contain every vertex defined above, except x_i and $\overline{x}_i$ $(1 \leq i \leq n)$, thus S has size $5n + 2m + 2p + q + 3$. Adding an S-copier between a and b means the following: let $S' = S \setminus \{a, b\}$, we add an $|S'|$-copier to the graph such that distinguished vertices α, β are identified with a, b, and the vertices $\gamma_1, \ldots, \gamma_{|S'|}$ are identified with the vertices in S' (in any order). Adding an S-edge is defined similarly. Adding an x_i-copier between a and b means that we add a γ-copier to the graph, and identify α, β with a, b, and identify γ with x_i. The description of G is completed by adding an

- S-edge between f_1 and f_2, between f_2 and f_3, between f_1 and f_3,
- S-copier between f_1 and x_i $(1 \leq i \leq n)$,
- S-copier between f_1 and $\overline{x}_i$ $(1 \leq i \leq n)$,
- S-edge between f_3 and x'_i $(1 \leq i \leq n)$,
- S-edge between f_3 and $\overline{x}'_i$ $(1 \leq i \leq n)$,
- S-copier between x'_i and t'_i $(1 \leq i \leq n)$,
- S-copier between $\overline{x}'_i$ and t'_i $(1 \leq i \leq n)$,
- S-copier between f_2 and x^*_i $(1 \leq i \leq n)$,
- x_i-copier between f_1 and t_i $(1 \leq i \leq n)$,
- $\overline{x}_i$-copier between f_1 and t_i $(1 \leq i \leq n)$,
- S-edge between f_3 and y_j $(1 \leq j \leq m)$,
- S-edge between f_3 and $\overline{y}_j$ $(1 \leq j \leq m)$,
- S-edge between y_j and $\overline{y}_j$ $(1 \leq j \leq m)$,

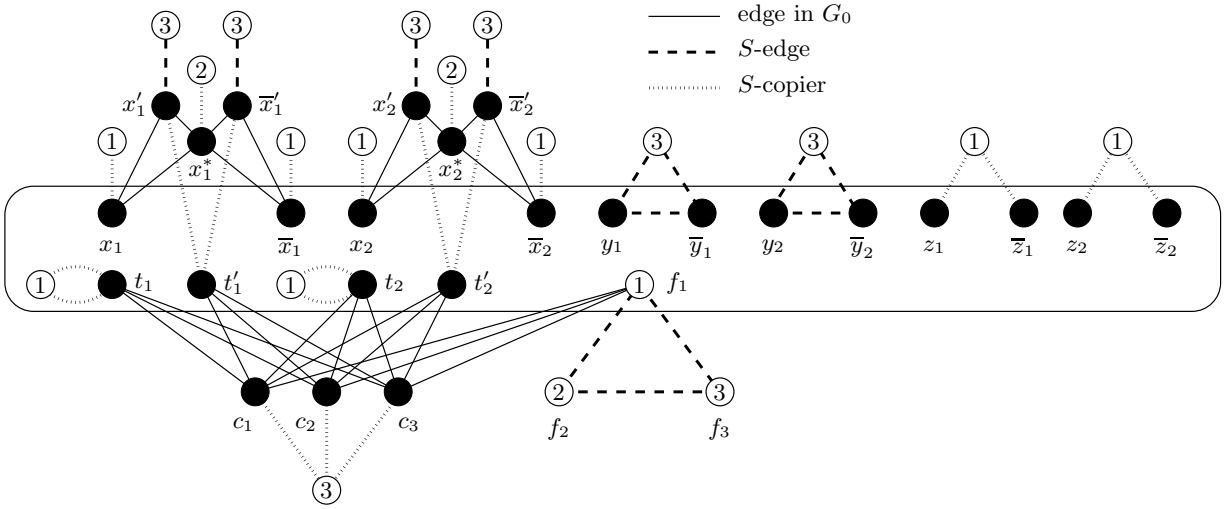


Figure 6.7: Sketch of the construction used in the proof of Theorem 6.4.3. The vertices f_1, f_2, f_3 are shown multiple times, e.g., every appearance of the white vertex 1 is identical to f_1. The two dotted edges between f_1 and t_1 represent the x_1-copier and the $\overline{x}_1$-copier. In the rounded box, every vertex is connected to every other vertex, except the pairs $x_i\overline{x}_i$, $y_j\overline{y}_j$, and $z_k\overline{z}_k$. Depending on the formula φ, vertex c_ℓ is connected to some vertices representing literals.

- S-copier between f_1 and z_k $(1 \le k \le p)$,
- S-copier between f_1 and $\overline{z}_k$ $(1 \le k \le p)$,
- S-copier between f_3 and c_ℓ $(1 \le \ell \le q)$.

The graph G for $n = m = p = 2$, $q = 3$ is shown on Figure 6.7. It can be verified that the maximal cliques of G can be divided into the following three types:

1. The flat edges of G.
2. The *x_i-triangles* x_i, x_i^*, x_i', and the *$\overline{x}_i$-triangles* $\overline{x}_i$, x_i^*, $\overline{x}_i'$.
3. The *assignment* cliques that contain the vertices f_1, t_i, t_i' $(1 \le i \le n)$. Besides these vertices, an assignment clique contains exactly one of x_i and $\overline{x}_i$, exactly one of y_j, $\overline{y}_j$, exactly one of z_k, $\overline{z}_k$, and at most one c_ℓ $(1 \le i \le n,\ 1 \le j \le m,\ 1 \le k \le p,\ 1 \le \ell \le q)$.

First we show that if there is an $\mathbf{x} \in \{0,1\}^n$ such that $\forall \mathbf{y} \exists \mathbf{z} \varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$, then there is an induced subgraph $G(\mathbf{x})$ of G that is not 3-clique-colorable. To obtain $G(\mathbf{x})$, delete vertex $\overline{x}_i$ from G iff variable x_i is true in $\mathbf{x}$, and delete vertex x_i from G iff variable x_i is false. Recall that x_i and $\overline{x}_i$ are not in S.

Assume that there is a 3-clique-coloring ψ of $G(\mathbf{x})$. Since every vertex of S is present in $G(\mathbf{x})$, the S-edge between f_1 and f_2 ensures that $\psi(f_1) \neq \psi(f_2)$, thus it can be assumed that $\psi(f_1) = 1$ and $\psi(f_2) = 2$. Because of the S-edge between f_1 and f_3, and between f_2 and f_3, we also have that $\psi(f_3) = 3$. We claim that x_i, $\overline{x}_i$ (if they are present in $G(\mathbf{x})$), t_i, t_i', z_k, $\overline{z}_k$ all have color 1. Assume that x_i is in $G(\mathbf{x})$ (the argument is similar, if $\overline{x}_i$ is in $G(\mathbf{x})$, and x_i is not). Vertex x_i has color 1 because of the S-copier between f_1 and x_i. There is an S-copier between f_2 and x_i^*, thus $\psi(x_i^*) = 2$. Since $x_i \in G(\mathbf{x})$, the x_i-copier between f_1 and t_i ensures that $\psi(t_i) = 1$. If x_i is in $G(\mathbf{x})$, then $\overline{x}_i$ is not in $G(\mathbf{x})$ and the edge $x_i^*\overline{x}_i'$ is a maximal clique, thus $\psi(\overline{x}_i') \neq \psi(x_i^*) = 2$. Moreover, because of the S-edge between $\overline{x}_i'$ and f_3, we have

$\psi(\overline{x}'_i) \neq 3$, implying $\psi(\overline{x}'_i) = 1$. Since there is an S-copier between $\overline{x}'_i$ and t'_i, hence $\psi(t'_i) = 1$. Finally, the S-copier between f_1 and z_k, and between f_1 and $\overline{z}_k$ implies that $\psi(z_k) = \psi(\overline{z}_k) = 1$.

The S-edge between f_3 and y_j, and between f_3 and $\overline{y}_j$ ensure that y_j and $\overline{y}_j$ have color 1 or 2. Furthermore, because of the S-edge between y_j and $\overline{y}_j$, one of them has color 1, and the other has color 2. Define the vector $\mathbf{y} \in \{0,1\}^m$ such that variable y_j is true if and only if $\psi(y_j) = 1$. By assumption, there is a vector $\mathbf{z} \in \{0,1\}^p$ such that $\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is true. Let K contain all the vertices of $G(\mathbf{x})$ that corresponds to true literals in $\mathbf{x}, \mathbf{y}, \mathbf{z}$. Moreover, add to K the vertices $f_1,\ t_i,\ t'_i$ $(1 \leq i \leq n)$. Clearly, K is a clique. Furthermore, because of the way K was constructed, every vertex in K has color 1. We claim that K is a monochromatic maximal clique, contradicting the assumption that ψ is a proper 3-clique-coloring of $G(\mathbf{x})$. It is easy to see that only a vertex c_ℓ corresponding to a clause might extend K to larger clique. However, in this case the assignment $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ does not satisfy φ since clause C_ℓ is not satisfied: otherwise there is a vertex in K that corresponds to a literal satisfying C_ℓ, and by the construction c_ℓ is not connected to this vertex.

To prove the other direction of the reduction, assume that there is an induced subgraph G' of G that is not 3-clique-colorable, we have to show that $\exists\mathbf{x}\forall\mathbf{y}\exists\mathbf{z}\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ holds. By Prop. 6.1.1, it can be assumed that G' contains all the simple vertices of G: adding simple vertices to G' does not make it 3-clique-colorable.

Call an induced subgraph of G *standard*, if for every $1 \leq i \leq n$, it contains exactly one of x_i and $\overline{x}_i$, and it contains every other vertex of G (in particular, it contains every vertex of S). First we show that every nonstandard subgraph of G is 3-clique-colorable, thus G' must be standard. Next we show that if there is a standard subgraph G' of G that is not 3-clique-colorable, then there is an $\mathbf{x} \in \{0,1\}^n$ satisfying $\forall\mathbf{y}\exists\mathbf{z}\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$. These two lemmas complete the proof of this direction of the reduction.

Lemma 6.4.4. *If G' is a nonstandard induced subgraph of G, then G' is 3-clique-colorable.*

Proof. Let G' be a nonstandard subgraph of G. By Lemma 6.1.1 it can be assumed that G' contains every simple vertex of G. Moreover, we show that G' contains every vertex of S. Assume that a vertex $v \in S$ is missing from G'. The absence of v turns off the S-copiers and the S-edges, which makes the coloring very easy. However, the x_i-copiers might still be working. Let G'_0 be the induced subgraph of G' containing only those vertices that are in G_0. We show that there is a 3-clique-coloring of G'_0 with the following property: if both f_1 and t_i are in G'_0 for some $1 \leq i \leq n$, and at least one of $x_i, \overline{x}_i$ is in G'_0, then f_1 and t_i have the same color. If this is true, then this coloring can be extended to a 3-clique-coloring of G': by Property 4 of Lemma 6.4.2, the coloring can be extended to every S-copier and S-edge, since $v \in S$ is missing from G'. Here we use Prop. 6.1.2: if we extend the coloring of G'_0 such that every gadget is 3-vertex-colored, then it gives a 3-clique-coloring of G. Moreover, the x_i-copier and the $\overline{x}_i$-copier between f_1 and t_i can be colored as well, since both x_i and $\overline{x}_i$ are missing (Property 4 of Lemma 6.4.1), or f_1 and t_i have the same color (Property 3 of Lemma 6.4.1).

To see that such a coloring of G'_0 exists, consider the following assignment of colors:

- Vertices $f_1,\ x'_i,\ \overline{x}'_i,\ t_i,\ t'_i$ have color 1 $(1 \leq i \leq n)$.
- Vertices $f_2,\ x_i,\ \overline{x}_i,\ y_j,\ \overline{y}_j,\ z_k,\ \overline{z}_k$ have color 2 $(1 \leq i \leq n,\ 1 \leq j \leq m,\ 1 \leq k \leq p)$.
- Vertices $f_3,\ x^*_i,\ c_\ell$ have color 3 $(1 \leq i \leq n,\ 1 \leq \ell \leq q)$.

This is a proper 3-clique-coloring of G'_0 if at least one of $T_1 = \{f_1, x'_i, \overline{x}'_i, t_i, t'_i\}$ and at least one of $T_2 = \{f_2, x_i, \overline{x}_i, y_j, \overline{y}_j, z_k, \overline{z}_k\}$ is present in G'_0, it can be verified that a clique with color 1 can be extended by a vertex with color 2, and a clique with color 2 can be extended by a vertex with color 1. The two special cases, when either all of T_1 or all of T_2 is missing, have to be considered separately. However, these cases are easy to handle, the details are left to the reader.

Now assume that G' is a nonstandard subgraph of G and every vertex of S is in G'. Since the graph is nonstandard, there is an $1 \leq i_0 \leq n$ such that either G' contains both x_{i_0} and $\overline{x}_{i_0}$, or G' contains neither x_{i_0} nor $\overline{x}_{i_0}$. The following coloring of G'_0 can be extended to a proper 3-clique-coloring of G':

- Vertices f_1, x_i, $\overline{x}_i$, x'_i, $\overline{x}'_i$, t_i, t'_i have color 1, where $1 \leq i \leq n$ and $i \neq i_0$.
- Vertices y_j, $\overline{y}_j$, z_k, $\overline{z}_k$ have color 1 ($1 \leq j \leq m$, $1 \leq k \leq p$)
- Vertices f_2, x^*_i have color 2 ($1 \leq i \leq n$).
- Vertices f_3, c_ℓ have color 3 ($1 \leq \ell \leq q$).
- If both x_{i_0} and $\overline{x}_{i_0}$ are in G', then x'_{i_0}, $\overline{x}'_{i_0}$, t'_{i_0} have color 2 and x_{i_0}, $\overline{x}_{i_0}$, t_{i_0} have color 1.
- If neither x_{i_0} nor $\overline{x}_{i_0}$ are in G', then x'_{i_0}, $\overline{x}'_{i_0}$, t'_{i_0} have color 1 and t_{i_0} has color 2.

This coloring can be extended to G' in such a way that the flat edges are properly colored (that is, it can be extended to the internal vertices of the copier and edge gadgets). Indeed, it can be verified by inspection that the two distinguished vertices of the S-copiers (resp. S-edges) have the same (resp. different) colors, respectively. Moreover, for $i \neq i_0$, both f_1 and t_i have color 1, thus the coloring can be extended to the x_i-copier and $\overline{x}_i$-copier between f_1 and t_i. However, if both x_{i_0} and $\overline{x}_{i_0}$ are missing from G', then f_1 has color 1 and t_{i_0} has color 2. But in this case the absence of x_{i_0} and $\overline{x}_{i_0}$ ensures that the two copiers between f_1 and t_{i_0} can be colored, regardless of the color of f_1 and t_{i_0} (Property 4 of Lemma 6.4.1).

The triangles x_i, x^*_i, x'_i and $\overline{x}_i$, x^*_i, $\overline{x}'_i$ contain both color 1 and 2. Therefore only the assignment cliques can be monochromatic in this coloring. However, every assignment clique contains t_{i_0} and t'_{i_0}, and these two vertices have different colors. ■

Therefore we can assume that G' is a standard subgraph. We show that based on G' we can define an assignment $\mathbf{x}$ such that $\forall \mathbf{y} \exists \mathbf{z} \varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$. The proof is similar to the proof of the first direction.

Lemma 6.4.5. *If there is a standard subgraph G' of G that is not 3-clique-colorable, then there is an $\mathbf{x} \in \{0,1\}^n$ satisfying $\forall \mathbf{y} \exists \mathbf{z} \varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$.*

Proof. Define vector $\mathbf{x} \in \{0,1\}^n$ by setting variable x_i to true if $x_i \in G'$, and to false if $\overline{x}_i \in G'$. We claim that $\forall \mathbf{y} \exists \mathbf{z} \varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$. Suppose that, on the contrary, there is a vector $\mathbf{y} \in \{0,1\}^m$ such that $\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is false for every $\mathbf{z} \in \{0,1\}^p$.

Consider the following coloring of G':

- Vertices f_1, x_i, $\overline{x}_i$, x'_i, $\overline{x}'_i$, t'_i, t_i, z_k, $\overline{z}_k$ have color 1 ($1 \leq i \leq n$, $1 \leq k \leq p$).
- Vertices f_2, x^*_i have color 2 ($1 \leq i \leq n$).
- Vertices f_3, c_ℓ have color 3 ($1 \leq \ell \leq q$).
- If variable y_j is true in $\mathbf{y}$, then vertex y_j has color 1 and vertex $\overline{y}_j$ has color 2 ($1 \leq j \leq m$).
- If variable y_j is false in $\mathbf{y}$, then vertex y_j has color 2 and vertex $\overline{y}_j$ has color 1 ($1 \leq j \leq m$).

As in the proof of the previous lemma, this coloring can be extended to the whole G' in such a way that every flat edge and every x_i-triangle is properly colored. By assumption, this coloring is not a proper 3-clique-coloring, thus there is a monochromatic maximal clique K, which must be an assignment clique. Since every assignment clique contains f_1, therefore every vertex of K has color 1. By the definition of the coloring, this means that K contains y_j if and only if y_j is true in $\mathbf{y}$. For every $1 \leq k \leq p$, an assignment clique contains exactly one of z_k and $\overline{z}_k$, define the vector $\mathbf{z} \in \{0,1\}^p$ by setting variable z_k to true if and only if $z_k \in K$. Notice that apart from f_1, t_i, t'_i, clique K contains those vertices that correspond to true literals in the assignment $\mathbf{x}, \mathbf{y}, \mathbf{z}$.

We claim that $\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is true. To see this, assume that clause C_ℓ is not satisfied by this assignment. Vertex c_ℓ is not in K, since c_ℓ has color 3. Now clique K contains the vertices f_1, t_i, t'_i, and vertices corresponding literals not satisfying C_ℓ, therefore $K \cup \{c_\ell\}$ is also a clique, contradicting the maximality of K. ■

Putting together these two lemmas completes the proof of the theorem. ■

Hereditary k-clique-coloring is also Π_3^p-complete for every $k > 3$. The proof is analogous to the proof of Corollary 6.2.2, the same construction can be used to reduce the case of k colors to $k + 1$ colors.

Corollary 6.4.6. *For every $k \geq 3$, Hereditary k-Clique-Coloring is Π_3^p-complete.* ■

The complexity of the case $k = 2$ remains an open question. The problems seems to be very different if there are only 2-colors. For example, we cannot build the copier and edge gadgets.

CHAPTER 7

Open questions

I am sorry that I have had to leave so many problems unsolved.
I always have to make this apology, but the world
really is rather puzzling and I cannot help it.
Bertrand Russell (1872–1970)

In this chapter we list some open questions related to result discussed in the previous chapters. Resolving these questions might be fruitful directions for future work.

1. In Section 2.3 we gave a polynomial time algorithm for list edge multicoloring odd cycles, and a randomized polynomial time algorithm for even cycles. Find a deterministic polynomial time algorithm for even cycles as well.

2. Biró, Hujter, and Tuza [BHT92] have shown that precoloring extension is NP-hard for interval graphs, and in Section 3.3 we have strengthened this result by showing that the problem remains NP-hard for unit interval graphs. However, [BHT92] proves that precoloring extension is NP-hard for interval graphs even if every color is used at most twice in the precoloring (2-PrExt). Does the problem remain NP-hard for unit interval graphs with this restriction?

3. For interval graphs and chordal graphs the 1-PrExt problem is easier then the general precoloring extension problem: the former is polynomial-time solvable, while the latter is NP-hard. Are there any other classes of graphs where 1-PrExt is easy? Line graphs of bipartite graphs or line graphs of planar bipartite graphs might be good candidates.

4. In Section 4.4 we have seen graphs where the chromatic index and the chromatic edge strength are different. In particular, we have presented a 3-edge-colorable graph with chromatic edge strength 4. Can we find such a graph that is planar as well?

5. Kovács [Kov04] gave a pseudopolynomial algorithm for the minimum sum multicoloring of paths. The algorithm is polynomial only if the demands can be bounded by a polynomial of the size of the path. Is it possible to find an algorithm that runs in polynomial time even if the demands are exponentially large?

6. In Section 5.3 we have shown that minimum sum edge coloring is NP-hard for trees. In Section 5.4 we presented a PTAS for the problem. First we gave a PTAS for bounded degree trees, the PTAS for general trees uses the algorithm for bounded degree trees as a subroutine. But we haven't shown that the problem is NP-hard, so it might be possible that there is a polynomial-time algorithm for bounded degree trees. What is the complexity of minimum sum edge multicoloring for bounded degree trees?

7. In Section 6.4 we have shown that Hereditary k-Clique-Coloring is Π_3^p-complete for every $k \geq 3$. But what is the complexity of the case $k = 2$? It seems that in the case of two colors, the combinatorics of the problem is very different. Thus it might happen that the problem is easier in this case, and it is not Π_3^p-complete.

CHAPTER 8

Conclusions

In this dissertation we have considered several different coloring problems. The study of these problems is important, since they appear in many applications. Our goal was to better explore the borderline between the polynomial-time solvable and NP-hard cases of these problems. We achieved this goal by presenting algorithms and proving complexity results.

Tighter complexity results were obtained for the list edge coloring problem. List edge coloring is polynomial-time solvable for trees, but it turned out that the problem becomes NP-hard for partial 2-trees. This result is somewhat surprising, since usually we expect that the algorithms for trees can be generalized for partial k-trees.

Algorithms were presented for list edge multicoloring on graphs with few cycles. These algorithms can be used to schedule file transfers or other biprocessor tasks. The vertices represent the processors, the edges represent the tasks, and the colors correspond to the available time slots. The tasks have to be scheduled in such a way that a processor can work on only one task at the same time (as discussed in Section 2.3.3, the results can be generalized to the case where a processor can participate in at most f tasks simultaneously). Previous algorithms for list edge multicoloring worked only on acyclic graphs, therefore the new algorithms allow us to solve the problem for more general networks. We also discussed a worker-assignment application that can be modeled by list multicoloring. The problem of assigning workers to one-day and two-day jobs can be reduced to the list edge multicoloring of a tree. Furthermore, we can consider additional constraints such as a worker cannot be assigned to one-day jobs on two consecutive days. If we want to handle these constraints, then it is no longer possible to reduce the problem to the list edge multicoloring of a tree, we have to solve the list edge multicoloring problem in a graph that has cycles. If there are only a few of these additional constraints, then the problem can be efficiently solved with the algorithm of Theorem 2.3.10. Thus the new results on list edge multicoloring increase the range of problems that can be efficiently solved.

In Chapter 3 we answered two open questions of Hujter and Tuza [HT96] on precoloring extension. First, we have shown that 1-PrExt is polynomial-time solvable for chordal graphs. In Section 3.1.5 we have discussed an application of this result to the configuration of WDM optical networks. In WDM networks we have to assign wavelengths to the connections in such a way that if two connections use the same optical equipment, then they have to receive different wavelengths. We have investigated a version of the problem on tree networks. It turned out that the problem can be formulated as the coloring of a chordal graph. Using the algorithm developed for precoloring extension, we can solve the more general

problem where the connections going through one of the switches are already configured, and we have to extend this configuration to the whole network.

Answering an open question of Biró, Hujter, and Tuza, we have shown that precoloring extension remains NP-hard for unit interval graphs. The first step of proving this result was to show the NP-completeness of an Eulerian disjoint paths problem, which answered an open question of Vygen [Vyg94].

Complexity results were given for several cases of minimum sum edge coloring. These results nicely complement the positive results. In particular, we know that the problem is polynomial-time solvable for trees, but NP-hard for partial 2-trees; for planar graphs the problem is NP-hard, but admits a PTAS; and for bipartite graphs there are constant-factor approximation algorithms, but the problem is APX-hard. The complexity of chromatic strength and chromatic edge strength was also investigated, it turned out that the less-known complexity class Θ_2^p appears naturally in the study of these concepts.

The multicoloring version of minimum sum coloring is motivated by applications in scheduling. It can be used to minimize the average completion time of interfering jobs. The vertices correspond to the jobs, two jobs are connected if they cannot be performed at the same time. The colors correspond to the available time slots, the demand of each vertex is the number of time slots required by the corresponding job. Assigning color sets to the vertices is equivalent to finding a scheduling of the jobs. By minimizing the sum of the largest color at each vertex we minimize the average completion time of the jobs. Minimizing the average completion time is an important goal if the jobs belong to separate users: the users want their jobs to be finished quickly, and they are not interested in the length of the schedule.

We answer an open question of Halldórsson et al. [HKP$^+$03] by showing that minimum sum multicoloring is NP-hard for binary trees. In the edge coloring case we show that minimum sum edge multicoloring is NP-hard for trees, but admits a polynomial-time approximation scheme. For minimum sum edge multicoloring, there were only constant-factor approximation algorithms known in the literature, this is the first time that an approximation scheme is obtained for the problem. The approximation scheme is extended for planar graphs and partial k-trees in [Mar04f]. As noted above, edge multicoloring models biprocessor task scheduling (file transfers, mutual diagnostic testing, etc.) if the goal is to minimize the average completion time of the tasks. The approximation schemes give good approximate solutions for these problems if the network is tree-like or planar. These graph classes are very important, as they often appear in practical applications.

The clique coloring problem was known to be NP-hard, we have determined the exact complexity of the problem by showing that the problem is in fact Σ_2^p-complete. The choosability and hereditary versions of the problem are also considered, we have shown that these problems lie higher in the polynomial hierarchy: they are Π_3^p-complete.

APPENDIX A

Technical background

appendix: /əˈpendɪks/ appendices /əˈpendɪsi:z/ *or* appendixes.
Your appendix *is a small closed tube inside*
your body which is attached to your digestive system.
It has no particular function.
COLLINS COBUILD Learner's Dictionary, 1996

In this appendix we review some of the most important notions and concepts that are used throughout this work.

A.1 Treewidth

The notion of treewidth was introduced by Robertson and Seymour in their long series of papers on graph minors [RS86]. Treewidth turned out to be a very powerful tool for designing algorithms and proving structural results on graphs. For a detailed treatment of treewidth and related concepts, the reader is referred to [Bod93, Bod98]. Here we summarize the most important facts about treewidth, and give some background on its importance to motivate the study of bounded treewidth graphs.

Treewidth can be defined in three equivalent ways. None of the definitions seem overwhelmingly natural, but the applications show that it is indeed an important concept worth studying.

Definition 1: Tree decomposition. Let $G(V, E)$ be an undirected graph. A *tree decomposition* of G is a tree $T(U, F)$ together with a set $S_x \subseteq V$ for each node $x \in U$ such that

1. $\bigcup_{x \in U} S_x = V$,
2. if $v, w \in V$ are neighbors in G, then there is an $x \in U$ with $v, w \in S_x$, and
3. for each $x, y, z \in U$, if y is on the unique path connecting x and z in T, then $S_x \cap S_z \subseteq S_y$.

For the sake of clarity, it is customary to call V the vertices of G and U the nodes of T. The *width* of a tree decomposition is $\max_{x \in U} |S_x| - 1$. The *treewidth* of a graph is the minimum width of its tree decompositions. The only reason for subtracting 1 in the definition is to ensure that every tree has

treewidth 1. Indeed, it is not difficult to show that every tree has a tree decomposition where every set S_x has size 2. (Assume that the graph G is a rooted tree. The nodes in U correspond to the edges of G, the tree T is obtained by connecting $x, y \in U$ if the lower end point of edge x is the same as the upper end point of edge y. For every $x \in U$, let S_x contain the two end points of edge x.)

Another way to look at the tree decomposition is the following. Let T_v contain those nodes x of T where $v \in S_x$. The third property ensures that T_v is a connected subset of T, that is, it induces a subtree of T. The second property says that the subtrees of two adjacent vertices have to intersect each other. Therefore a tree decomposition is a way of embedding the graph into a tree such that the subtrees do not overlap very much. Thus if a graph has small treewidth, then it is "tree-like" in some sense.

Definition 2: Chordal graphs. A graph is *chordal* if every cycle of length greater than 3 contains at least one chord, i.e., an edge connecting two vertices not adjacent in the cycle. Equivalently, a graph is chordal if and only if it does not contain a cycle of length greater than 3 as an induced subgraph. Chordal graphs can be also characterized as the intersection graphs of subtrees of a tree (see e.g., [Gol80]):

Theorem A.1.1. *The following two statements are equivalent:*

1. *$G(V, E)$ is chordal.*
2. *There exists a tree $T(U, F)$ and a subtree $T_v \subseteq T$ for each vertex $v \in V$ such that vertices $u, v \in V$ are neighbors in $G(V, E)$ if and only if $T_u \cap T_v \neq \emptyset$.*

Let $x \in U$ be an arbitrary node of T, it is clear that the those vertices whose subtree contain x induce a clique. Therefore Theorem A.1.1 implies that every chordal graph with clique size $k + 1$ has a width k tree decomposition. Thus if it is possible to add edges to the graph in such a way that it becomes a chordal graph with clique size $k+1$, then the graph has treewidth at most k. In fact, it can be shown that the treewidth of a graph G is $\min \omega(H) - 1$, where the minimum is taken over all chordal supergraphs H of G.

Definition 3: Partial k-trees. We define *k-trees* recursively as follows:

1. A clique of size k is a k-tree.
2. If G is a k-tree and K is clique of size k in G, then the graph obtained by adding a new vertex v to G and connecting v with every vertex of K is also a k-tree.
3. Every k-tree can be obtained with the previous two rules.

In particular, 1-trees are exactly the trees. A graph is a *partial k-tree* if it is the subgraph of a k-tree. A partial k-tree has clique size at most $k + 1$. Every k-tree is chordal, hence using the second definition it follows that every partial k-tree has treewidth at most k. Moreover, it can be shown that a graph is a partial k-tree if and only if it has treewidth at most k, thus we obtain another characterization of treewidth. We will use the notion of partial k-trees and graphs with bounded treewidth interchangeably.

Algorithms. A large number of NP-hard problems can be solved efficiently for partial k-trees. Most of these algorithms generalize the dynamic programming method that can solve many problems for trees. To give the flavor of these algorithms, we show how to solve maximum weighted independent set for trees, and briefly sketch how the algorithm can be generalized to partial k-trees.

In the weighted independent set problem a graph is given with a positive integer weight $w(v)$ for each vertex v. The goal is to find an independent set I such that $\sum_{v \in I} w(v)$ is maximal. Assume that the graph is a rooted tree with root r. For a node v of T, let T^v be the subtree of T rooted at v (i.e., T^v contains v and its descendants). We give a dynamic programming algorithm that solves a large number of

subproblems. Most of these subproblems are of no direct interest to us, but we have to solve all of them to be able to answer the original question.

We solve the weighted maximum independent set problem for each subtree T_v. More precisely, we solve two subproblems for each node v of T: let A_v be the weight of the maximum independent set in T_v with the further requirement that v has to be in the set, and let B_v be the maximum set if the set must not contain v. Clearly, the answer to the problem is $\max\{A_r, B_r\}$ for the root r.

The values A_v, B_v are determined in a bottom-up fashion. If v is a leaf, then trivially $A_v = w(v)$, $B_v = 0$. Now assume that $v_1, \ldots, v_t$ are the children of v, and $A_{v_1}, \ldots, A_{v_t}$, $B_{v_1}, \ldots, B_{v_t}$ are already determined. It is easy to see that

$$A_v = w(v) + \sum_{i=1}^{t} B_{v_t},$$

since if v is selected, then $v_1, \ldots, v_t$ cannot be selected and therefore B_{v_i} is the maximum weight that we can achieve in the subtree T_{v_i}. On the other hand, if v is not selected into the independent set, then the weight that can be selected from T_{v_i} is $\max\{A_{v_i}, B_{v_i}\}$. Thus we obtain

$$B_v = \sum_{i=1}^{t} \max\{A_{v_i}, B_{v_i}\}.$$

Using these two recurrence relations repeatedly, we can arrive to the root r, which solves the problem in linear time.

How can we generalize this method to partial k-trees? Consider the tree decomposition $T(U, F)$ of the partial k-tree $G(V, E)$. For a node $x \in U$, let T^x be the subtree of T rooted at x. Let $V_x = \bigcup_{y \in T^x} S_y$, that is, V_x contains those vertices that appear in the set of a node of T^x. The subgraph of G induced by V_x will be denoted by $G_x = G[V_x]$. The following important property of G_x can be verified directly: if a vertex v of G_x has a neighbor outside G_x, then v is in S_x. When we were considering trees, we used the fact that for each subtree only one vertex, the root, can be "seen" from outside. For each subgraph G_x only the at most $k + 1$ vertices in S_x can be seen from outside G_x. Therefore we will solve 2^{k+1} subproblems for each node $x \in T$: for each $Z \subseteq S_x$ we determine the maximum weight of an independent set that contains Z, but does not contain any of $S_x \setminus Z$. It can be shown that these subproblems can be solved using a bottom-up method similar to the one used in the case of trees. However, the recurrence relations for the subproblems will be more complicated, we spare the reader the gory details.

Courcelle's Theorem [Cou90] (see also [DF99, Section 6.5]) gives a clean way of quickly showing that a problem is linear time solvable on bounded treewidth graphs. Sentences in the *Extended Monadic Second Order Logic of Graphs* contain quantifiers, logical connectives ($\neg$, $\vee$, and $\wedge$), vertex variables, edge variables, vertex set variables, edge set variables, and the following binary relations: $\in$, $=$, $\mathrm{inc}(e, v)$ (edge variable e is incident to vertex variable v), and $\mathrm{adj}(u, v)$ (vertex variables u, v are neighbors). If a graph property can be described in this language, then this description can be turned into an algorithm:

Theorem A.1.2 (Courcelle [Cou90]). *If a graph property can be described in the Extended Monadic Second Order Logic of Graphs, then for every w, there is a linear-time algorithm for the recognition of this property on graphs with treewidth at most w.*

For example, we can easily describe 3-colorability in this language. By Theorem A.1.2, this immediately implies that we can decide in linear time whether a partial k-tree is 3-colorable.

$$\begin{aligned}\text{3-colorable}(V,E) \quad := \quad & \exists C_1 \subseteq V,\ \exists C_2 \subseteq V,\ \exists C_3 \subseteq V : (\forall v : v \in C_1 \vee v \in C_2 \vee v \in C_3) \\ & \wedge\ (\forall u,v \in V : \neg\text{adj}(u,v) \vee u \notin C_1 \vee v \notin C_1) \\ & \wedge\ (\forall u,v \in V : \neg\text{adj}(u,v) \vee u \notin C_2 \vee v \notin C_2) \\ & \wedge\ (\forall u,v \in V : \neg\text{adj}(u,v) \vee u \notin C_3 \vee v \notin C_3)\end{aligned}$$

The three sets C_1, C_2, C_3 correspond to the three color classes. The first line expresses the requirement that every vertex should belong to one of the color classes. The next three lines ensure that each color class is independent.

If a class of graph has bounded treewidth, then the methods for partial k-trees can be used to solve problems for this graph class. A graph is *outerplanar* if it can be embedded in the plane such that every vertex is on the exterior infinite face. Every outerplanar graph has treewidth at most 2, thus the results above imply that weighted maximum independent set and 3-coloring can be solved in linear time for such graphs. *Series-parallel* graphs are defined as follows:

1. The graph consisting only an edge is a series-parallel graph.
2. Subdividing an edge of a series-parallel graph gives another series-parallel graph.
3. Adding a new edge parallel to an old edge of a series-parallel graph gives another series-parallel graph.
4. Every series-parallel graph can be obtained with these operations.

Every series-parallel graph has treewidth at most 2. The converse is almost true: if a 2-connected graph has treewidth at most 2, then it is series-parallel.

Besides finding algorithms for special graph classes, treewidth can be useful when handling general graphs as well. There are algorithms that first check whether the graph has small treewidth, if so, then they use a method suited for partial k-trees, if not, then they make use of the fact that the graph has large treewidth and do something else. For example, this pattern is used for finding disjoint paths [RS95]. Parameterized complexity (see [DF99]) uses bounded-treewidth methods extensively.

We remark that there are some problems that can be solved in polynomial time for trees, but become NP-hard for partial k-trees. For example, the edge disjoint paths problem is trivial for trees: there is a unique path connecting each pair of terminals. However, the problem becomes NP-hard for partial 2-trees [NVZ01]. We show that the situation is similar with the list edge coloring, edge precoloring extension, and minimum sum edge coloring problems. These problems are polynomial-time solvable for trees, but we prove in Section 2.1.2, 3.4, and 4.3, respectively, that they are NP-hard for partial 2-trees.

A.2 Approximation algorithms

The theory of NP-completeness, introduced by Cook [Coo71], Levin [Lev73] and Karp [Kar72] in the early 70s, tells us that we should not try to find polynomial-time algorithms for certain problems, because most probably no such algorithms exist. It turned out that this theory can be applied to a wide range of practically interesting problems. The ubiquity of NP-complete problems made NP-completeness one of the most important concepts of theoretical computer science. However, proving that certain problems are hard does not alleviate the need to solve these problems. The most common way to cope with hard problems is to use heuristic algorithms, which do not guarantee an optimum solution, but we hope that they provide solutions that are "good enough."

Approximation algorithms are special kinds of heuristic algorithms: they guarantee that the solutions they find are close to the optimum. In a minimization problem (such as minimum vertex cover, traveling salesperson, etc.) we are trying to find a solution with cost as small as possible. An algorithm for a minimization problem is an *α-approximation algorithm* if it always produces a solution with cost at most α times the optimum. Clearly, α is always at least 1, and the closer it is to 1, the better is the algorithm. If α is a constant, then we say that the algorithm provides a constant-factor approximation. For example, there is a simple 2-approximation algorithm for the minimum vertex cover problem (find a set of vertices such that every edge has a selected vertex). However, for certain problems no such constant-factor approximation is known, in every approximation algorithm the value of α is a function of the instance size. For example, as shown by Johnson [Joh74] and Lovász [Lov75], the set cover problem (given a family of sets, find a subset of the elements such that every set contains at least one selected element) can be $O(\log n)$-approximated by a simple greedy method. However, as shown by [LY94] and [BGLR93], it is unlikely that the problem has an approximation algorithm significantly better than that.

For a maximization problem, an α-approximation algorithm produces a solution that is always at least $1/\alpha$ times the optimum. As in the case of the minimization problems, the smaller is α, the better is the algorithm.

There are problems where there is no best approximation algorithm: there is a polynomial-time α-approximation algorithm for every $\alpha > 1$. In this case we say that the problem admits a *polynomial-time approximation scheme (PTAS)*. A PTAS can be also thought of as an algorithm that has an additional parameter $\epsilon > 0$, and the algorithm produces an $(1+\epsilon)$-approximate solution. Moreover, for a fixed value of ϵ, the running time of the PTAS has to be polynomial (typically it is something like $O(2^{1/\epsilon} \cdot n)$ or $O(n^{1/\epsilon})$). It is important to realize that the possibility of having an arbitrarily close approximation in polynomial time does not contradict the fact that finding the optimum is NP-hard.

NP-completeness gives a way of showing that a problem cannot be solved in polynomial time, unless P = NP. Similarly, we have tools to show that a problem cannot be α-approximated in polynomial time for some α. Some of these techniques are based on the theory of Probabilistically Checkable Proofs (PCP), which was the most important development of computational complexity in the 90s (see e.g., [ACG$^+$99]). Here we concentrate on how to show that a problem does not have a PTAS.

The class APX contains those optimization problems that have polynomial-time α-approximation algorithms for some constant value of α. Given a problem instance x, we denote by $\mathrm{OPT}(x)$ the cost of the optimum solution, and we denote by $c(x, s)$ the cost of a particular solution s. Following [Pap94], we define a notion of reducibility between optimization problems. This reduction has the property that if a problem A is reducible to problem B, and B has a PTAS, then it follows that A has a PTAS as well. Formally, we say that A is *L-reducible* to B if there are two polynomial-time computable functions f and g, and two positive constants λ and μ such that

- If x is an instance of A, then $f(x)$ is an instance of B with $\mathrm{OPT}(f(x)) \leq \lambda \cdot \mathrm{OPT}(x)$.
- If s is a solution for instance $f(x)$, then $g(s)$ is a solution for instance x with

$$|c(x, g(s)) - \mathrm{OPT}(x)| \leq \mu \cdot |c(f(x), s) - \mathrm{OPT}(f(x))|.$$

That is, an instance of A can be transformed into an instance of B in such a way that the optimum does not increase by too much. Moreover, if a solution for the transformed instance is close to the optimum, then we can obtain a solution for the original instance that is also close to the optimum. It is easy to see that if A is L-reducible to B and B has a PTAS than A has a PTAS as well.

An optimization problem is APX-hard if every problem in APX is L-reducible to it. If an APX-hard problem has a PTAS, then every problem in APX would have a PTAS, which is considered highly unlikely. Moreover, it can be shown (and here is where the deep results of PCP theory come into play) that if an APX-hard problem has a PTAS, then P = NP follows, which makes this possibility even less likely.

As a nice case study, we end this section by briefly summarizing the approximability of the different variants of the famous *traveling salesperson problem (TSP)*. Given a set of cities and a distance function $d(x, y)$ between the cities, the salesperson has to visit every city exactly once. Our goal is to find an ordering of the cities such that the total distance between the cities is minimized. Depending on the assumptions we have on the distance function, the complexity of the problem is the following:

- Without any restriction on the distance function, there cannot be a constant factor approximation algorithm for TSP, unless P = NP. This follows fairly easily from the NP-completeness of the Hamiltonian cycle problem (see e.g., [Pap94]).
- In the *metric TSP* problem we assume that the triangle inequality $d(x, y) \leq d(x, z) + d(z, y)$ holds for every three cities x, y, z. In this case the algorithm of Christofides (see e.g., [ACG+99]) gives a $\frac{3}{2}$-approximation. However, the problem is APX-hard [PY93], thus there is no PTAS unless P = NP.
- In the *Euclidean TSP* problem the cities are given with coordinates in the 2-dimensional plane, and the distance is the usual Euclidean distance. Euclidean TSP has a PTAS [Aro98].
- The PTAS for Euclidean TSP can be generalized to the d-dimensional version of the problem for every fixed d. However, if the dimension can be as high as $\Theta(\log n)$, then the problem becomes APX-hard [Tre00].

A.3 Oracles and the polynomial hierarchy

In computational complexity we often want to say statements like "assuming we have a subroutine for problem A, we can solve problem B." For example, the main result in Cook's original paper introducing NP-completeness [Coo71] is stated the following way: if we have a magical subroutine that solves the SAT problem instantaneously, then every language in NP can be solved in polynomial time. (Later Karp [Kar72] introduced many-one reductions, which shows that NP-completeness can be defined without subroutines.)

The formal way of talking about subroutines is to use oracle Turing-machines. Let A be an arbitrary language. A *Turing-machine equipped with an A-oracle* is a Turing-machine with three special states $q_?$, q_{yes}, q_{no}, and a distinguished tape called the *oracle tape.* The machine works identically to a classical Turing-machine, with one exception. If it reaches the query state $q_?$, then the next state is q_{yes} if the contents of the oracle tape is a member of A, and the next state is q_{no} if the oracle tape is not in A. Thus the machine can quickly determine whether a word is in A or not: it just copies the word to the oracle tape and goes to the state $q_?$. The machine can go to the state $q_?$ multiple times during a computation, thus the "subroutine" for A can be called as many times as required.

The important thing is that, however complicated is the language A, the transition from $q_?$ to q_{yes} or to q_{no} counts only as one step. Therefore if A has high complexity, then equipping a machine with an A-oracle can highly increase its power. For example, if A is undecidable, then a machine with an A-oracle can solve problems that no Turing-machine can solve. If A is NP-hard, then it may be possible that a machine with an A-oracle solves in polynomial time a problem that a classical Turing-machine can solve only in exponential time.

As we have seen above, one use of oracle machines is to define reductions between problems. We say that a language B is *polynomial-time Turing-reducible* to language A, if B can be solved in polynomial time by a polynomial-time Turing-machine equipped with an A-oracle. Therefore Cook's Theorem is formally stated as every language in NP is polynomial-time Turing-reducible to SAT.

Another use of oracles is to define new complexity classes. If A is an arbitrary language, then we denote by P^A the set of languages that can be decided by a polynomial-time Turing-machine equipped with an A-oracle. The classes NP^A, EXP^A, etc. are similarly defined.[1] Baker, Gill, and Solovay have shown in

[1] However, we have to be careful with the oracle notation. The notation X^A has a meaning only if X is a complexity class defined by Turing-machines. If X is an arbitrary collection of languages, then we cannot define X^A.

their seminal paper [BGS75] that there are languages A and B such that $\mathrm{P}^A = \mathrm{NP}^A$ and $\mathrm{P}^B \neq \mathrm{NP}^B$. Why is this result important? Its importance comes from the fact that it rules out certain types of approaches for resolving $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$. For example, $\mathrm{P} \neq \mathrm{EXP}$ can be proved by a simple argument based on diagonalization. The proof uses just one property of Turing-machines: there are universal Turing-machines that can simulate other Turing-machines. Therefore the proof remains the same if we add an A-oracle to the Turing-machines. We say that the proof of $\mathrm{P} \neq \mathrm{EXP}$ *relativizes*, thus $\mathrm{P}^A \neq \mathrm{EXP}^A$ follows for every language A. The result of Baker, Gill, and Solovay implies that $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ cannot be resolved by a proof that relativizes, since there are oracles such that there is equality with one oracle, and there is inequality with the other. There are very few proofs that do not relativize, hence we need fundamentally new ideas to resolve $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$.

If A and B are two NP-complete languages, then P^A and P^B are the same complexity classes: an NP-complete language can simulate another with a polynomial amount of additional work. We will denote this class by P^{NP}: polynomial-time deterministic computation with an NP-complete oracle. When we use oracle notation and there is a complexity class in the exponent, then this will mean that the oracle is a complete set for that class, and it doesn't really matter which complete set we choose.

The *polynomial hierarchy* was introduced by Stockmeyer [Sto76]. The hierarchy starts with $\Delta_1^p = \mathrm{P}$, $\Sigma_1^p = \mathrm{NP}$, $\Pi_1^p = \mathrm{coNP}$, and for $i > 1$ we have

$$\begin{aligned} \Delta_i^p &= \mathrm{P}^{\Sigma_{i-1}^p}, \\ \Sigma_i^p &= \mathrm{NP}^{\Sigma_{i-1}^p}, \\ \Pi_i^p &= \mathrm{coNP}^{\Sigma_{i-1}^p}. \end{aligned}$$

We can characterize the class Σ_i^p also by certificates. Recall that a language L is in NP if we can present a polynomial-time verifiable certificate for each member of L. Formally, a language is in NP if there is a language $L' \in \mathrm{P}$ such that if $x \in L$, then there is a polynomially-bounded y with $(x, y) \in L'$, but if $x \notin L$, then $(x, y) \notin L'$ for any such y. It can be shown that if instead of requiring $L' \in P$ we require only $L' \in \Pi_i^p$, then we obtain the class Σ_{i+1}^p: the class Σ_{i+1}^p contains exactly those problems where there is a Π_i^p-verifiable certificate. To illustrate these concepts, we finish this section by presenting three problems complete for higher levels of the polynomial-hierarchy:

- Consider the following problem: given a boolean formula $\varphi(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m)$, is it possible to assign values to the variables $x_1, x_2, \ldots, x_n$ in such a way that the formula is satisfied no matter what values are assigned to $y_1, \ldots, y_m$? For example, if $\varphi(x_1, y_1, y_2, y_3) = (x_1 \vee y_1 \vee y_2) \wedge (x_1 \vee y_1 \vee y_3)$, then the answer is yes (set $x_1 = 1$). However, for $\varphi(x_1, y_1, y_2, y_3) = (x_1 \vee y_1) \wedge (y_2 \vee y_3)$ the answer is no. Notice that if the answer is yes, then the assignment to $x_1, \ldots, x_n$ is a certificate for the yes answer. To verify that the certificate is correct, we have to check whether the formula is satisfied for every assignment to the variables $y_1, \ldots, y_m$. Thus checking the certificate is in $\mathrm{coNP} = \Pi_1^p$, therefore it follows that the problem is in Σ_2^p. It can be shown that the problem is in fact Σ_2^p-complete (see e.g., [Pap94]).

- Given a formula in disjunctive normal form (DNF), is it possible to find an equivalent DNF with length at most k? This problem is Σ_2^p-complete [Uma01]. If the answer is yes, then an equivalent DNF of length at most k can be a certificate. To verify this certificate, it has to be checked whether it is equivalent to the original formula. It is coNP-complete to check whether two DNFs are equivalent.

- Given a graph G with a set of vertices S, is it true that every 2-coloring of S can be extended to a 3-coloring of G? This problem is Π_2^p-complete [Sze02], which means that now we have a coNP-verifiable certificate for every *no* instance. If the answer is no, then we can certify this by presenting a 2-coloring of S that is not extendable. Checking whether a 2-coloring is really unextendable is in coNP.

Bibliography

If I have seen further it is by standing on the shoulders of giants.
Sir Isaac Newton (1642–1727)

Note: At the end of each entry, the page numbers after the symbol "→" show the pages where the entry is referenced.

[ACG+99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and approximation.* Springer-Verlag, Berlin, 1999. → *pp. 153–154*

[AF96] T. Andreae and C. Flotow. On covering all cliques of a chordal graph. *Discrete Math.*, 149(1-3):299–302, 1996. → *p. 127*

[AK00] P. Alimonti and V. Kann. Some APX-completeness results for cubic graphs. *Theoret. Comput. Sci.*, 237(1-2):123–134, 2000. → *p. 58*

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows.* Prentice Hall Inc., Englewood Cliffs, NJ, 1993. Theory, algorithms, and applications. → *p. 34*

[Aro98] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998. → *p. 154*

[ART01] D. Amar, A. Raspaud, and O. Togni. All-to-all wavelength-routing in all-optical compound networks. *Discrete Math.*, 235(1-3):353–363, 2001. Combinatorics (Prague, 1998). → *p. 37*

[Bak94] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. Assoc. Comput. Mach.*, 41(1):153–180, 1994. → *p. 111*

[BGG+04] G. Bacsó, S. Gravier, A. Gyárfás, M. Preissmann, and A. Sebő. Coloring the maximal cliques of graphs. *SIAM J. Discrete Math.*, 17(3):361–376, 2004. → *pp. 6, 8, 127*

[BGLR93] M. Bellare, S. Goldwasser, C. Lund, and A. Russeli. Efficient probabilistically checkable proofs and applications to approximations. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 294–304. ACM Press, 1993. → *p. 152*

[BGP+00] J.-C. Bermond, L. Gargano, S. Perennes, A. A. Rescigno, and U. Vaccaro. Efficient collective communciation in optical networks. *Theoret. Comput. Sci.*, 233(1-2):165–189, 2000. → *p. 2*

[BGS75] T. Baker, J. Gill, and R. Solovay. Relativizations of the $\mathcal{P} =? \mathcal{NP}$ question. *SIAM J. Comput.*, 4(4):431–442, 1975. → *p. 154*

[BHT92] M. Biró, M. Hujter, and Zs. Tuza. Precoloring extension. I. Interval graphs. *Discrete Math.*, 100(1-3):267–279, 1992. → *pp. 4, 6, 9, 25–26, 45, 69, 145*

[BHT93] M. Biró, M. Hujter, and Zs. Tuza. Cross fertilisation of graph theory and aircraft maintenance scheduling. In G. Davidson, editor, *Thirty Second Annual Symposium AGIFORS (Airline Group of the International Federation of Operation Research Societies*, pages 307–317, 1993. → *pp. 25, 45*

[BJW94] H. L. Bodlaender, K. Jansen, and G. J. Woeginger. Scheduling with incompatible jobs. *Discrete Appl. Math.*, 55(3):219–232, 1994. → *pp. 25–26, 70*

[BNBH+98] A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai, and T. Tamir. On chromatic sums and distributed resource allocation. *Inform. and Comput.*, 140(2):183–202, 1998. → *pp. 52–53*

[BNHK+99] A. Bar-Noy, M. M. Halldórsson, G. Kortsarz, R. Salman, and H. Shachnai. Sum multicoloring of graphs. In *Algorithms—ESA '99 (Prague)*, pages 390–401. Springer, Berlin, 1999. → *pp. 7, 88*

[BNHK+00] A. Bar-Noy, M. M. Halldórsson, G. Kortsarz, R. Salman, and H. Shachnai. Sum multicoloring of graphs. *J. Algorithms*, 37(2):422–450, 2000. → *pp. 5, 8, 88–89, 110*

[BNK98] A. Bar-Noy and G. Kortsarz. Minimum color sum of bipartite graphs. *J. Algorithms*, 28(2):339–365, 1998. → *p. 52*

[Bod93] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernet.*, 11(1-2):1–21, 1993. → *p. 149*

[Bod98] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoret. Comput. Sci.*, 209(1-2):1–45, 1998. → *pp. 12, 149*

[Bri92] P. Briggs. *Register Allocation via Graph Coloring.* PhD thesis, Rice University, Houston, Texas, 1992. → *p. 3*

[BW99] K. P. Bogart and D. B. West. A short proof that "proper = unit". *Discrete Math.*, 201(1-3):21–23, 1999. → *p. 44*

[CGJL85] E. G. Coffman, Jr., M. R. Garey, D. S. Johnson, and A. S. LaPaugh. Scheduling file transfers. *SIAM J. Comput.*, 14(3):744–780, 1985. → *p. 88*

[CMLF00] T. Cinkler, D. Marx, C. P. Larsen, and D. Fogaras. Heuristic algorithms for joint configuration of the optical and electrical layer in multi-hop wavelength routing networks. In *Proceedings of IEEE INFOCOM 2000*, pages 1000–1009, 2000. → *p. 2*

[Col84] C. J. Colbourn. The complexity of completing partial Latin squares. *Discrete Appl. Math.*, 8(1):25–30, 1984. → *pp. 4, 6, 25, 49*

[Coo71] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM Press, 1971. → *pp. 152, 154*

[Cou90] B. Courcelle. Graph rewriting: an algebraic and logic approach. In *Handbook of theoretical computer science, Vol. B*, pages 193–242. Elsevier, Amsterdam, 1990. → *p. 151*

[DF99] R. G. Downey and M. R. Fellows. *Parameterized complexity*. Monographs in Computer Science. Springer-Verlag, New York, 1999. → *pp. 151–152*

[EGT92] P. Erdős, T. Gallai, and Zs. Tuza. Covering the cliques of a graph with vertices. *Discrete Math.*, 108(1-3):279–289, 1992. Topological, algebraical and combinatorial structures. Frolík's memorial volume. → *p. 127*

[EIS76] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, 1976. → *pp. 6, 9–10*

[EJ98] Erlebach and Jansen. Maximizing the number of connections in optimal tree networks. In *ISAAC: 9th International Symposium on Algorithms and Computation*, 1998. → *p. 37*

[EJ01] T. Erlebach and K. Jansen. The complexity of path coloring and call scheduling. *Theoret. Comput. Sci.*, 255(1-2):33–50, 2001. → *pp. 2, 37*

[EJK+99] T. Erlebach, K. Jansen, C. Kaklamanis, M. Mihail, and P. Persiano. Optimal wavelength routing on directed fiber trees. *Theoret. Comput. Sci.*, 221(1-2):119–137, 1999. ICALP '97 (Bologna). → *pp. 2, 37*

[EP01] T. Easton and R. G. Parker. On completing Latin squares. *Discrete Appl. Math.*, 113(2-3):167–181, 2001. → *pp. 6, 10, 25–26, 49*

[Erl99] T. Erlebach. Maximum Weight Edge-Disjoint Paths in Bidirected Trees. In *Communication and Data Management in Large Networks. Workshop of INFORMATIK'99*, pages 13–19, 1999. → *p. 37*

[ERT80] P. Erdős, A. L. Rubin, and H. Taylor. Choosability in graphs. In *Proceedings of the West Coast Conference on Combinatorics, Graph Theory and Computing (Humboldt State Univ., Arcata, Calif., 1979)*, pages 125–157, Winnipeg, Man., 1980. Utilitas Math. → *pp. 4, 9, 128, 133*

[Eul36] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Opera Omnia*, 7:128–140, 1736. → *p. 1*

[Fia03] J. Fiala. NP completeness of the edge precoloring extension problem on bipartite graphs. *J. Graph Theory*, 43(2):156–160, 2003. → *pp. 6–7, 10, 25–26, 49*

[FIZN03] T. Fujino, I. Isobe, X. Zhou, and T. Nishizeki. Linear algorithm for finding list edge-colorings of series-parallel graphs. *IEICE Trans. Fundamentals*, E86-D(No. 2):186–190, 2003. → *p. 9*

[FKP01] J. Fiala, J. Kratochvíl, and A. Proskurowski. Distance constrained labeling of precolored trees. In *Theoretical computer science (Torino, 2001)*, volume 2202 of *Lecture Notes in Comput. Sci.*, pages 285–292. Springer, Berlin, 2001. → *p. 4*

[Fra82] A. Frank. Disjoint paths in a rectilinear grid. *Combinatorica*, 2(4):361–371, 1982. → *p. 38*

[Fra90] A. Frank. Packing paths, circuits, and cuts—a survey. In *Paths, flows, and VLSI-layout (Bonn, 1988)*, pages 47–100. Springer, Berlin, 1990. → *pp. 38–39*

[FZN03] T. Fujino, X. Zhou, and T. Nishizeki. List edge-colorings of series-parallel graphs. *IEICE Trans. Fundamentals*, E86-A(No. 5):1034–1045, 2003. → *p. 9*

[GHKS04] R. Gandhi, M. M. Halldórsson, G. Kortsarz, and H. Shachnai. Improved bounds for the sum multicoloring problem and scheduling dependent jobs with minsum criteria, 2004. To appear in 2nd Workshop on Approximation and Online Algorithms, Bergen, 2004. → *p. 52*

[GJ79] M. R. Garey and D. S. Johnson. *Computers and intractability*. W. H. Freeman and Co., San Francisco, Calif., 1979. A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences. → *p. 75*

[GJKM02] K. Giaro, R. Janczewski, M. Kubale, and M. Małafiejski. A 27/26-approximation algorithm for the chromatic sum coloring of bipartite graphs. In *APPROX 2002*, volume 2462 of *Lecture Notes in Comput. Sci.*, pages 135–145. Springer, Berlin, 2002. → *p. 52*

[GJS76] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoret. Comput. Sci.*, 1(3):237–267, 1976. → *p. 39*

[GK00] K. Giaro and M. Kubale. Edge-chromatic sum of trees and bounded cyclicity graphs. *Inform. Process. Lett.*, 75(1-2):65–69, 2000. → *pp. 7–8, 52–53, 55, 60, 89, 116*

[GLS88] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*. Springer-Verlag, Berlin, 1988. → *p. 101*

[Gol80] M. C. Golumbic. *Algorithmic graph theory and perfect graphs*. Academic Press, New York, 1980. → *pp. 26–27, 34, 150*

[Gut96] S. Gutner. The complexity of planar graph choosability. *Discrete Math.*, 159(1-3):119–130, 1996. → *p. 133*

[Hem89] L. A. Hemachandra. The strong exponential hierarchy collapses. *J. Comput. System Sci.*, 39(3):299–322, 1989. → *p. 73*

[HH00] E. Hemaspaandra and L. A. Hemaspaandra. Computational politics: electoral systems. In *Mathematical Foundations of Computer Science 2000 (Bratislava)*, pages 64–83. Springer, Berlin, 2000. → *p. 74*

[HHR97] E. Hemaspaandra, L. A. Hemaspaandra, and J. Rothe. Exact analysis of Dodgson elections: Lewis Carroll's 1876 voting system is complete for parallel access to NP. *J. ACM*, 44(6):806–825, 1997. → *p. 74*

[HK73] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2:225–231, 1973. → *p. 19*

[HK02] M. M. Halldórsson and G. Kortsarz. Tools for multicoloring with applications to planar graphs and partial k-trees. *J. Algorithms*, 42(2):334–366, 2002. → *pp. 5, 88, 90, 110, 118, 120*

[HKP+03] M. M. Halldórsson, G. Kortsarz, A. Proskurowski, R. Salman, H. Shachnai, and J. A. Telle. Multicoloring trees. *Inform. and Comput.*, 180(2):113–129, 2003. → *pp. 5, 8, 88, 110, 118, 148*

[HKS03] M. M. Halldórsson, G. Kortsarz, and H. Shachnai. Sum coloring interval and k-claw free graphs with application to scheduling dependent jobs. *Algorithmica*, 37(3):187–209, 2003. → *pp. 8, 52–53*

[HM00] M. Holzer and P. McKenzie. Alternating and empty alternating auxiliary stack automata. In *Mathematical Foundations of Computer Science 2000 (Bratislava)*, pages 415–425. Springer, Berlin, 2000. → *p. 73*

[HM02] M. Holzer and P. McKenzie. On auxiliary pushdown and stack automata. *ACM SIGACT News*, 33(1):32–45, 2002. → *p. 73*

[HMT00] H. Hajiabolhassan, M. L. Mehrabadi, and R. Tusserkani. Minimal coloring and strength of graphs. *Discrete Math.*, 215(1-3):265–270, 2000. → *pp. 5, 53, 68, 79*

[Hol81] I. Holyer. The NP-completeness of edge-coloring. *SIAM Journal on Computing*, 10(4):718–720, November 1981. → *pp. 9, 69*

[HR98] E. Hemaspaandra and J. Rothe. Recognizing when greed can approximate maximum independent sets is complete for parallel access to NP. *Inform. Process. Lett.*, 65(3):151–156, 1998. → *p. 73*

[HT93] M. Hujter and Zs. Tuza. Precoloring extension. II. Graph classes related to bipartite graphs. *Acta Mathematica Universitatis Comenianae*, 62(1):1–11, 1993. → *pp. 4, 6, 9, 25–26, 69*

[HT96] M. Hujter and Zs. Tuza. Precoloring extension. III. Classes of perfect graphs. *Combin. Probab. Comput.*, 5(1):35–56, 1996. → *pp. 6, 25–26, 45, 147*

[HvdVV94] J. A. Hoogeveen, S. L. van de Velde, and B. Veltman. Complexity of scheduling multiprocessor tasks with prespecified processor allocations. *Discrete Appl. Math.*, 55(3):259–272, 1994. → *p. 88*

[Izb64] H. Izbicki. An edge colouring problem. In *Theory of Graphs and its Applications (Proc. Sympos. Smolenice, 1963)*, pages 53–61. Publ. House Czechoslovak Acad. Sci., Prague, 1964. → *p. 71*

[Jan97] K. Jansen. The optimum cost chromatic partition problem. In *Algorithms and complexity (Rome, 1997)*, volume 1203 of *Lecture Notes in Comput. Sci.*, pages 25–36. Springer, Berlin, 1997. → *pp. 4, 6, 9, 52, 114*

[Jan00] K. Jansen. Approximation results for the optimum cost chromatic partition problem. *J. Algorithms*, 34(1):54–89, 2000. → *pp. 5, 114*

[JMT99] M. Juvan, B. Mohar, and R. Thomas. List edge-colorings of series-parallel graphs. *Electron. J. Combin.*, 6(1):Research Paper 42, 6 pp. (electronic), 1999. → *p. 9*

[Joh74] D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. System Sci.*, 9:256–278, 1974. Fifth Annual ACM Symposium on the Theory of Computing (Austin, Tex., 1973). → *p. 152*

[JS97] K. Jansen and P. Scheffler. Generalized coloring for tree-like graphs. *Discrete Appl. Math.*, 75(2):135–155, 1997. → *pp. 6, 9, 12, 15, 25*

[Kad89] J. Kadin. $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ and sparse Turing-complete sets for NP. *J. Comput. System Sci.*, 39(3):282–298, 1989. → *p. 73*

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*, pages 85–103. Plenum, New York, 1972. → *pp. 152, 154*

[KG02] W. F. Klostermeyer and J. L. Goldwasser. List multi-coloring problems. *Bull. Inst. Combin. Appl.*, 34:71–76, 2002. → *pp. 16, 22*

[Klo94] T. Kloks. *Treewidth*, volume 842 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1994. Computations and approximations. → *p. 27*

[Kov04] A. Kovács. Sum-multicoloring on paths. In *STACS 2004: 21st Annual Symposium on Theoretical Aspects of Computer Science.*, volume 2996 of *Lecture Notes in Comput. Sci.*, pages 68–80, Heidelberg, 2004. Springer. → *pp. 5, 7–8, 88, 110, 145*

[KP98] J. Kececioglu and A. Pecqueur. Computing maximum-cardinality matchings in sparse general graphs. In *2nd Workshop on Algorithm Engineering (WAE 1998)*, pages 121–132, 1998. → *p. 19*

[Kra93] J. Kratochvíl. Precoloring extension with fixed color bound. *Acta Mathematica Universitatis Comenianae*, 62(2):139–153, 1993. → *p. 26*

[KS89] E. Kubicka and A. J. Schwenk. An introduction to chromatic sums. In *Proceedings of the ACM Computer Science Conf.*, pages 15–21. Springer, Berlin, 1989. → *pp. 5, 52, 68*

[KT94] J. Kratochvíl and Zs. Tuza. Algorithmic complexity of list colorings. *Discrete Appl. Math.*, 50(3):297–302, 1994. → *p. 6*

[KT02] J. Kratochvíl and Zs. Tuza. On the complexity of bicoloring clique hypergraphs of graphs. *J. Algorithms*, 45(1):40–54, 2002. → *pp. 8, 127*

[Kub89] E. Kubicka. *The Chromatic Sum of a Graph.* PhD thesis, Western Michigan University, 1989. → *p. 52*

[Kub93] M. Kubale. Interval edge coloring of a graph with forbidden colors. *Discrete Math.*, 121(1-3):135–143, 1993. Graph theory (Niedzica Castle, 1990). → *p. 9*

[Kub96] M. Kubale. Preemptive versus nonpreemptive scheduling of biprocessor tasks on dedicated processors. *European Journal of Operational Research*, 94(2):242–251, 1996. → *pp. 110, 113*

[Lev73] L. A. Levin. Universal sorting problems. *Problemy Peredachi Informatsii*, 9(3):265–266, 1973. In Russian. → *p. 152*

[LG83] D. Leven and Z. Galil. NP completeness of finding the chromatic index of regular graphs. *Journal of Algorithms*, 4(1):35–44, March 1983. → *pp. 9, 69, 71*

[Lov75] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Math.*, 13(4):383–390, 1975. → *p. 152*

[Lov93] L. Lovász. *Combinatorial problems and exercises.* North-Holland Publishing Co., Amsterdam, second edition, 1993. → *p. 132*

[Lov97] L. Lovász. The membership problem in jump systems. *J. Combin. Theory Ser. B*, 70(1):45–66, 1997. → *p. 54*

[LP86] L. Lovász and M. D. Plummer. *Matching Theory.* North-Holland Publishing Co., Amsterdam, 1986. Annals of Discrete Mathematics, 29. → *pp. 19, 21*

[LR94] K.-J. Lange and K. Reinhardt. Empty alternation. In *Mathematical Foundations of Computer Science 1994 (Košice, 1994)*, pages 494–503. Springer, Berlin, 1994. → *p. 73*

[LY94] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *J. Assoc. Comput. Mach.*, 41(5):960–981, 1994. → *p. 152*

[Mah82] S. R. Mahaney. Sparse complete sets for NP: solution of a conjecture of Berman and Hartmanis. *J. Comput. System Sci.*, 25(2):130–143, 1982. → *p. 73*

[Mar02] D. Marx. The complexity of tree multicolorings. In *Mathematical Foundations of Computer Science 2002 (Warsaw-Otwock)*, pages 532–542. Springer, Berlin, 2002. → *pp. 10, 89, 110*

[Mar03a] D. Marx. List edge multicoloring in bounded cyclicity graphs. In *Proceedings of the 3rd Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications*, pages 164–170, 2003. → *p. 10*

[Mar03b] D. Marx. A short proof of the NP-completeness of minimum sum interval coloring, 2003. To appear in *Operations Research Letters.* → *p. 52*

[Mar04a] D. Marx. Complexity of chromatic strength and chromatic edge strength, 2004. Submitted. → *p. 53*

[Mar04b] D. Marx. Complexity of clique coloring and related problems, 2004. Manuscript. → *p. 128*

[Mar04c] D. Marx. Complexity results for minimum sum edge multicoloring, 2004. Manuscript. → *p. 53*

[Mar04d] D. Marx. Eulerian disjoint paths problem in grid graphs is NP-complete. *Discrete Appl. Math.*, 143(1-3):336–341, 2004. → *p. 26*

[Mar04e] D. Marx. List edge multicoloring in graphs with few cycles. *Inform. Process. Lett.*, 89(2):85–90, 2004. → *p. 10*

[Mar04f] D. Marx. Minimum sum multicoloring on the edges of planar graphs and partial k-trees, 2004. To appear in 2nd Workshop on Approximation and Online Algorithms, Bergen, 2004. → *pp. 8, 111, 148*

[Mar04g] D. Marx. Minimum sum multicoloring on the edges of trees. In *1st International Workshop on Approximation and Online Algorithms (WAOA) 2003 (Budapest)*, volume 2909 of *Lecture Notes in Computer Science*, pages 214–226. Springer, Berlin, 2004. → *p. 89*

[Mar04h] D. Marx. Minimum sum multicoloring on the edges of trees, 2004. Submitted. → *p. 89*

[Mar04i] D. Marx. NP-completeness of list coloring and precoloring extension on the edges of planar graphs, 2004. To appear in *Journal of Graph Theory.* → *pp. 10, 26*

[Mar04j] D. Marx. Precoloring extension on chordal graphs, 2004. Manuscript. Preliminary version presented at Graph Theory 2004, Paris. → *p. 26*

[Mar04k] D. Marx. Precoloring extension on unit interval graphs, 2004. Submitted. → *p. 26*

[McK90] B. McKay. Nauty user's guide, version 1.5. Technical Report TR-CS-90-02, Computer Science Department, Australian National University, 1990. Also `http://cs.anu.edu.au/people/bdm/nauty/`. → *p. 72*

[MMS97] J. Mitchem, P. Morriss, and E. Schmeichel. On the cost chromatic number of outerplanar, planar, and line graphs. *Discuss. Math. Graph Theory*, 17(2):229–241, 1997. → *pp. 5, 68, 79*

[MR01] C. Moore and J. M. Robson. Hard tiling problems with simple tiles. *Discrete Comput. Geom.*, 26(4):573–590, 2001. → *pp. 11, 40*

[MS90] O. Marcotte and P. D. Seymour. Extending an edge-coloring. *J. Graph Theory*, 14(5):565–573, 1990. → *pp. 6, 10, 16*

[MŠ99] B. Mohar and R. Škrekovski. The Grötzsch theorem for the hypergraph of maximal cliques. *Electron. J. Combin.*, 6(1):Research Paper 26, 13 pp., 1999. → *p. 128*

[MTW98] B. Mohar, Zs. Tuza, and G. Woeginger, 1998. Manuscript. → *p. 26*

[Mur00] K. Murota. *Matrices and matroids for systems analysis*, volume 20 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 2000. → *p. 35*

[MV80] S. Micali and V. V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science*, pages 17–27. IEEE, 1980. → *p. 19*

[MVV87] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987. → *p. 21*

[NSS99] S. Nicoloso, M. Sarrafzadeh, and X. Song. On the sum coloring problem on interval graphs. *Algorithmica*, 23(2):109–126, 1999. → *p. 52*

[NVZ01] T. Nishizeki, J. Vygen, and X. Zhou. The edge-disjoint paths problem is NP-complete for series-parallel graphs. *Discrete Appl. Math.*, 115(1-3):177–186, 2001. → *pp. 10, 60, 152*

[OBB81] J. B. Orlin, M. A. Bonuccelli, and D. P. Bovet. An $O(n^2)$ algorithm for coloring proper circular arc graphs. *SIAM J. Algebraic Discrete Methods*, 2(2):88–93, 1981. → *p. 45*

[OS81] H. Okamura and P. D. Seymour. Multicommodity flows in planar graphs. *J. Combin. Theory Ser. B*, 31(1):75–81, 1981. → *p. 38*

[Pap94] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley Publishing Company, Reading, MA, 1994. → *pp. 13, 74, 129, 153–155*

[PY93] C. H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Math. Oper. Res.*, 18(1):1–11, 1993. → *p. 154*

[PZ82] C. H. Papadimitriou and S. Zachos. Two remarks on the power of counting. In *Proceedings of the 6th GI-Conference on Theoretical Computer Science*, pages 269–276. Springer-Verlag, 1982. → *p. 73*

[Rec89] A. Recski. *Matroid theory and its applications in electric network theory and statics*, volume 6 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, New York and Akadémiai Kiadó, Budapest, 1989. → *pp. 35–36*

[Rec92] A. Recski. Minimax results and polynomial algorithms in VLSI routing. In *Fourth Czechoslovakian Symposium on Combinatorics, Graphs and Complexity (Prachatice, 1990)*, pages 261–273. North-Holland, Amsterdam, 1992. → *p. 45*

[Rob69] F. S. Roberts. Indifference graphs. In *Proof Techniques in Graph Theory (Proc. Second Ann Arbor Graph Theory Conf., Ann Arbor, Mich., 1968)*, pages 139–146. Academic Press, New York, 1969. → *p. 44*

[RS86] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. → *p. 149*

[RS95] N. Robertson and P. D. Seymour. Graph minors. XIII. The disjoint paths problem. *J. Combin. Theory Ser. B*, 63(1):65–110, 1995. → *p. 152*

[RTL76] D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5(2):266–283, 1976. → *pp. 27, 34*

[Sal00] M. Salavatipour. On sum coloring of graphs. Master's thesis, University of Toronto, 2000. → *p. 53*

[Sal03] M. R. Salavatipour. On sum coloring of graphs. *Discrete Appl. Math.*, 127(3):477–488, 2003. → *pp. 7–8, 52–53, 60, 68–69, 89, 116*

[Sch03] A. Schrijver. *Combinatorial optimization. Polyhedra and efficiency.*, volume 24 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 2003. → *p. 115*

[Sto76] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoret. Comput. Sci.*, 3(1):1–22, 1976. → *p. 155*

[Sup87] K. J. Supowit. Finding a maximum planar subset of nets in a channel. *IEEE Trans. Comput. Aided Design*, 6(1):93–94, 1987. → *p. 52*

[Sze02] S. Szeider. Generalizations of matched CNF formulas, 2002. To appear in *Annals of Mathematics and Artificial Intelligence*. → *p. 155*

[Szk99] T. Szkaliczki. Routing with minimum wire length in the dogleg-free Manhattan model is NP-complete. *SIAM J. Comput.*, 29(1):274–287, 1999. → *p. 52*

[Tre00] L. Trevisan. When Hamming meets Euclid: the approximability of geometric TSP and Steiner tree. *SIAM J. Comput.*, 30(2):475–485, 2000. → *p. 154*

[TT85] A. Teng and A. Tucker. An $O(qn)$ algorithm to q-color a proper family of circular arcs. *Discrete Math.*, 55(2):233–243, 1985. → *p. 45*

[Tuz97] Zs. Tuza. Graph colorings with local constraints—a survey. *Discuss. Math. Graph Theory*, 17(2):161–228, 1997. → *pp. 4, 10, 15*

[Uma01] C. Umans. The minimum equivalent DNF problem and shortest implicants. *J. Comput. System Sci.*, 63(4):597–611, 2001. Special issue on FOCS 98 (Palo Alto, CA). → *p. 155*

[Viz76] V. G. Vizing. Coloring the vertices of a graph in prescribed colors. *Diskret. Analiz*, (29 Metody Diskret. Anal. v Teorii Kodov i Shem):3–10, 101, 1976. → *pp. 4, 9, 128*

[Vyg94] J. Vygen. Disjoint paths. Technical Report 94816, Research Institute for Discrete Mathemathics, University of Bonn, 1994. → *pp. 6, 26, 37–39, 43–44, 148*

[Vyg95] J. Vygen. NP-completeness of some edge-disjoint paths problems. *Discrete Appl. Math.*, 61(1):83–90, 1995. → *pp. 39, 43*

[Wag87] K. W. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theoret. Comput. Sci.*, 51(1-2):53–80, 1987. → *pp. 73–74*

[Wag90] K. W. Wagner. Bounded query classes. *SIAM J. Comput.*, 19(5):833–846, 1990. → *p. 73*

[Wes95] D. West. Open problems section. *The SIAM Activity Group on Discrete Mathematics Newsletter*, 5(2):9, Winter 1994–95. → *pp. 68, 71*

[Wu00] J. L. Wu. List-edge coloring of series-parallel graphs. *Shandong Daxue Xuebao Ziran Kexue Ban*, 35(2):144–149, 2000. (in Chinese). → *p. 9*

[ZFN00] X. Zhou, K. Fuse, and T. Nishizeki. A linear algorithm for finding $[g, f]$-colorings of partial k-trees. *Algorithmica*, 27(3-4):227–243, 2000. → *p. 22*

[ZN99] X. Zhou and T. Nishizeki. Edge-coloring and f-coloring for various classes of graphs. *J. Graph Algorithms Appl.*, 3:no. 1, 18 pp., 1999. → *p. 22*

[ZN04] X. Zhou and T. Nishizeki. Algorithm for the cost edge-coloring of trees. *J. Comb. Optim.*, 8(1):97–108, 2004. → *pp. 7, 52–53, 60, 116*