Theoretical Computer Science Department
Faculty of Mathematics and Computer Science
Jagiellonian University

On-line choosability

Grzegorz Gutowski
grzegorz.gutowski@tcs.uj.edu.pl

Ph.D. Thesis
Adviser: Paweł M. Idziak
Kraków, September 2011

Abstract

We consider a natural task scheduling scenario with two types of constraints on the execution schedule. The constraints of the first type specify pairs of tasks which can not be executed at the very same time. The constraints of the second type describe which tasks can be executed in a given moment. Such a scheduling scenario describes a system in which tasks have conflicts, known to the scheduler in advance, and there are some auxiliary requirements that are being discovered over the time.

This kind of scheduling is perfectly modelled by a game "Mr. Paint and Mrs. Correct". This is an on-line game which received some attention in literature. The off-line case, in which the status of auxiliary requirements over the time is known a priori, is modelled by the classical list colouring. Many results from the list colouring realm were successfully adapted to the on-line one. On the other hand, inspection of scheduling strategies in "Mr. Paint and Mrs. Correct" allowed to improve some off-line results.

This thesis gives answers to two natural questions. First, we examine the computational complexity of the decision problems concerning existence of good scheduling strategies. We introduce tools and techniques which allow us to show that these problems are NP-hard. Parts of the presented argument rely heavily on computer assisted proofs.

In our second problem we assume that the constraints of the second type are such that each task can be scheduled in a huge number of rounds. Can the scheduling strategy guarantee to schedule each task not just once but in a prescribed constant fraction of the rounds it can be scheduled in? How big can this fraction be? We show that this fraction is the same in the off-line and in the on-line case. However, there is an important difference between the two notions. The asymptotic limit used in the definition is always attained in the off-line case. We show that this is no longer true in the on-line case.

Acknowledgements

I would like to thank my Ph.D. adviser Professor Paweł M. Idziak, who helped me in innumerable ways. None of this work would have been possible, if it hadn't been for his support and attention.

This is an opportunity to express my love and appreciation to my wife Marysia and our son Antoni. They gave me endless amounts of encouragement, understanding and love.

My fellow students, colleagues and friends from Theoretical Computer Science Department kept on inspiring me everyday. I want to thank them for all the time spent together.

Finally, I would like to express my special thanks to the department computers `sphinx` and `miracle` for hours and days of their precious cpu-time spent executing my ridiculous programs. I always could count on those guys.

Contents

1

Introduction

1.1. Scheduling conflicting tasks with auxiliary requirements

We start with considering the following task scheduling setting. We are given a set of tasks which are needed to be scheduled. Each task takes exactly one unit of time to be completed. Some of the tasks can be run in parallel while some can not. We assume that we are given the set of conflicting pairs of tasks beforehand. Our goal is to schedule the tasks so that no two conflicting tasks are run simultaneously. To make things even more difficult, tasks have some auxiliary requirements. From our point of view it does not matter what these requirements are. All we need to know is which of the tasks are currently available to be scheduled and which are not.

The main problem here is "Which of the remaining unscheduled tasks that are currently available should be run in the next unit of time?" A subset of the available tasks must be chosen to run with respect to the conflicting pairs constraints. Execution of other (not chosen) tasks will be postponed and these tasks hopefully can be scheduled at some other time in the future. Working in a real-time scenario it is desirable to guarantee an upper bound on the number of times any task is postponed before it finally gets scheduled. In such scenario the problem is "Can one prepare a scheduling strategy which guarantees that no task is postponed more than a prescribed number of times?"

This thesis is about the last question. We show that in terms of the computational complexity it is a hard one. We also examine different variations of this question and most notably we solve the fractional version of the problem in which every task is available in a huge number of rounds and needs to be scheduled not once but in a constant (the bigger, the better) fraction of the rounds it is available in.

1.2. Notation

We set $\mathbb{N} = \{0, 1, 2, \ldots\}$ to be the set of non negative integers. Most of this thesis, with some exceptions in Chapters 2 and 4, deals with finite structures. For that reason we assume that all sets are finite unless we state otherwise.

For a given set A, a power set $\mathcal{P}(A)$ is the set of all subsets of A, while $\mathcal{P}^+(A)$ denotes the set of all nonempty subsets of A.

A graph G is a pair $G = (V, E)$ where V is a set of vertices and E is a set of edges – two element subsets of V. Occasionally we abuse the notation and refer to the vertex set of G just by G. For thorough introduction of graph theory terminology we refer the reader to the standard textbook by Diestel **[Die10]**. We expect the reader to be familiar with the standard notions and we promptly introduce the most important definitions to fix the notation.

The degree $\deg(v)$ of a vertex $v \in V$ is the number of edges incident to v. The maximum degree among the vertices of a graph G is denoted by $\Delta(G)$. A vertex with degree 0 is called an isolated vertex.

A graph $H = (V_H, E_H)$ is a subgraph $H \subseteq G$ of a graph $G = (V_G, E_G)$ when $V_H \subseteq V_G$ and $E_H \subseteq E_G$. A subgraph H is an induced subgraph of G, denoted $H = G[V_H]$, when $E_H = E_G \cap (V_H \times V_H)$. For a vertex $v \in V_G$ we denote the graph $G[V_G \setminus \{v\}]$ by $G - v$.

In a given graph $G = (V, E)$, a subset of vertices $C \subseteq V$ is an independent set when the graph $G[C]$ has no edges.

A proper colouring (or colouring for short) of a graph $G = (V, E)$ is a function $\Phi : V \to \mathbb{N}$ which assigns to each vertex a positive integer (a colour) in such a way that adjacent vertices receive different colours, i.e.:

$$\forall (u, w) \in E \;\; \Phi(u) \neq \Phi(w).$$

Equivalently, Φ is a proper colouring when for each $i \in \mathbb{N}$ the set $\Phi^{-1}(i)$ is independent in G.

A list assignment L is a function assigning a finite set $L(v) \subseteq \mathbb{N}$ to each vertex $v \in V$. Given an integer valued function $f : V \to \mathbb{N}$ we say that L is f-long if $|L(v)| \geqslant f(v)$ for each vertex $v \in V$. When $f \equiv a$ is a constant function ($f(v) = a$ for all $v \in V$) we simply say that L is a-long.

Given an integer valued function $g : V \to \mathbb{N}$, a function Φ assigning a subset $\Phi(v) \subseteq L(v)$ to each vertex $v \in V$ is an (L, g)-colouring if each vertex v receives at least $g(v)$ colours and adjacent vertices receive

different colours, i.e.:

$$\begin{aligned} \forall v \in V \quad & |\Phi(v)| \geqslant g(v), \\ \forall (u,w) \in E \quad & \Phi(u) \cap \Phi(w) = \emptyset. \end{aligned}$$

For $g \equiv 1$ we say that an (L,g)-colouring is an L-colouring. List colourings were introduced by Vizing **[Viz76]** and independently by Erdős, Rubin and Taylor **[ERT80]**.

Now, for two functions $f, g : V \to \mathbb{N}$ we say that

- G is (f,g)-choosable if there exists an (L,g)-colouring from any f-long list assignment L,
- G is (f,g)-colourable if there exists an (L_f,g)-colouring from the list assignment L_f in which $L_f(v) := \{0,\ldots,f(v)-1\}$.

Moreover, for $f \equiv a$ and $g \equiv b$ we say that the graph G is (a,b)-choosable, or (a,b)-colourable, respectively.

Now, the classical notions of the a-colourability and the a-choosability coincide with ours of the $(a,1)$-colourability and the $(a,1)$-choosability, respectively. This simply means that standard notions of the chromatic number $\chi(G)$ and the choice number $\mathrm{ch}(G)$ can be equivalently restated as follows:

$$\begin{aligned} \chi(G) &= \min\{a : G \text{ is } (a,1)\text{-colourable}\}, \\ \mathrm{ch}(G) &= \min\{a : G \text{ is } (a,1)\text{-choosable}\}. \end{aligned}$$

The following inequality between these chromatic parameters follows directly from their definitions:

$$\chi(G) \leqslant \mathrm{ch}(G).$$

Another important graph parameter is the colouring number of G, denoted $\mathrm{col}(G)$, which is the minimal integer k for which there exists a linear ordering R (colouring order) of the vertex set V such that each $v \in V$ has at most $k-1$ neighbours that are earlier than v in the order R. The existence of a colouring order R is used in Algorithm 1.1 to construct an L-colouring from any $\mathrm{col}(G)$-long list assignment L. This leads to:

$$\mathrm{ch}(G) \leqslant \mathrm{col}(G).$$

Algorithm 1.1 COLOURING-ORDER-GREEDY

inputs
  $G = (V, E)$ {*graph*}
  $R = (v_0, v_1, \ldots, v_{n-1})$ {*colouring order*}
  $L : V \to \mathbb{N}$ {*list assignment*}
outputs
  Φ {*L-colouring of* G}
do
  for $i = 0$ **to** $n - 1$ **do**
    $F_i = \emptyset$ {*set of forbidden colours*}
    for $j = 0$ **to** $i - 1$ **do**
      if $\{v_i, v_j\} \in E$ **then**
        $F_i = F_i \cup \{\Phi(v_j)\}$ {v_j *is a neighbour of* v_i *earlier in* R*, colour of* v_j *is forbidden for* v_i}
      end if
    end for{F_i *has at most* $\operatorname{col}(G) - 1$ *elements*}
    $\Phi(v_i) =$ any colour in $L(v_i) \setminus F_i$
  end for
  return Φ

We conclude this preparatory section with three more definitions.

An r-partite graph is simply an r-colourable graph. A graph G is complete r-partite if the vertices of G can be decomposed to r disjoint sets so that two vertices are adjacent if and only if they belong to different sets in this decomposition.

The Cartesian product $G \times H$ of two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ is a graph with vertex set $V_G \times V_H$. Two vertices (u_1, w_1) and (u_2, w_2) are adjacent in $G \times H$ if one of the following holds:

$$\begin{aligned} u_1 = u_2 \text{ and } \{w_1, w_2\} \in E_H, \\ \text{or } w_1 = w_2 \text{ and } \{u_1, u_2\} \in E_G. \end{aligned}$$

For a graph $G = (V, E)$, by its line graph $L(G)$ we mean a graph $L(G) = (E, F)$ in which $(e, e') \in F$ if and only if e and e' share an end point in G (i.e. $|e \cap e'| = 1$). Chromatic parameters have their edge counterparts – the chromatic index $\chi'(G)$ and the list chromatic index

$\mathrm{ch}'(G)$ defined by:

$$\chi'(G) = \chi(L(G)),$$
$$\mathrm{ch}'(G) = \mathrm{ch}(L(G)).$$

1.3. Mr. Paint and Mrs. Correct

Section 1.1 introduced a scheduling problem. It is a common practice to model such problems using on-line games. Usually in those games there are two players – one of them constructs the scheduling instance and the second one schedules the tasks. In our case, the role of the first player would be to decide the availability of tasks in consecutive rounds and the second player would choose the tasks to run. The game "Mr. Paint and Mrs. Correct", or PC-game for short, introduced by Shauz **[Sch09]**, perfectly models this scheduling problem.

The game "Mr. Paint and Mrs. Correct" is played in rounds on a graph $G = (V, E)$ which is known in advance to both players: Mr. Paint and Mrs. Correct. Moreover, two functions, a list-length-function $f : V \to \mathbb{N}$ and a colour-demand-function $g : V \to \mathbb{N}$, are fixed in advance. We say that the PC-game is played on the board (G, f, g). When $g \equiv b$ (or even additionally $f \equiv a$) we say that the game is played on the board (G, f, b) (or (G, a, b)).

In terms of the scheduling, Mr. Paint is responsible for creating the scheduling instance and the role of Mrs. Correct is to correctly schedule the tasks. Edges of the graph G encode pairs of conflicting tasks. The colour-demand-function g tells us how many times a task needs to be scheduled and in the most natural setting it is set to $g \equiv 1$. The list-length-function f gives the desired upper bound for the number of times a task is postponed.

At the very beginning of the game each vertex v has an empty list $L_0(v) = \emptyset$ assigned to it. These initially empty lists will eventually become f-long as Mr. Paint is going to saturate the lists during the game. The goal of Mrs. Correct is to incrementally construct an (L, g)-colouring φ from the resulting list assignment. Mr. Paint, by smartly filling the lists $L(v)$, tries to prevent the assignment φ, being built by Mrs. Correct, from resulting in an (L, g)-colouring. Initial colouring φ_0 is set to $\varphi_0(v) = \emptyset$ for all $v \in V$.

In the i-th round Mr. Paint chooses a nonempty subset P_i of vertices with unsaturated lists (i.e. $P_i \subseteq \{v \in V : |L_{i-1}(v)| < f(v)\}$) and

appends the new colour $i-1$ to the lists assigned to the vertices in P_i so that:

$$L_i(v) = \begin{cases} L_{i-1}(v) \cup \{i-1\}, & \text{if } v \in P_i, \\ L_{i-1}(v), & \text{otherwise.} \end{cases}$$

At this point Mrs. Correct chooses an independent subset $C_i \subseteq P_i$ and assigns the colour $i-1$ to the vertices in C_i to get:

$$\varphi_i(v) = \begin{cases} \varphi_{i-1}(v) \cup \{i-1\}, & \text{if } v \in C_i, \\ \varphi_{i-1}(v), & \text{otherwise.} \end{cases}$$

In terms of the scheduling, the lists L_i describe the rounds in which a particular task was available to be scheduled. The colouring φ_i describes the decisions of the scheduling strategy.

If after ℓ rounds all the lists are full, i.e. $|L_\ell(v)| = f(v)$ for every $v \in V$, the game ends with Mrs. Correct's move. Mrs. Correct wins if φ_ℓ is a valid (L_ℓ, g)-colouring of G that is, if $|\varphi_\ell(v)| \geqslant g(v)$ for all $v \in V$. Otherwise Mr. Paint wins. Observe that the requirement $P_i \neq \emptyset$ guarantees that in each round at least one list is extended and that the game ends after at most $\sum_{v \in V} f(v)$ rounds.

We say that a graph G is on-line (f, g)-choosable if Mrs. Correct has a winning strategy in the PC-game on the board (G, f, g). When $f \equiv a$ and $g \equiv b$ we say that graph G is on-line (a, b)-choosable. When a graph is on-line $(a, 1)$-choosable we simply call it on-line a-choosable.

For a graph G, the on-line choice number $\mathrm{ch}^{\mathrm{OL}}(G)$ is defined by:

$$\mathrm{ch}^{\mathrm{OL}}(G) = \min\{a : G \text{ is on-line } a\text{-choosable}\}.$$

The following inequality is straightforward:

$$\mathrm{ch}(G) \leqslant \mathrm{ch}^{\mathrm{OL}}(G).$$

The on-line list chromatic index $\mathrm{ch}^{\mathrm{OL}'}(G)$ of G is defined by:

$$\mathrm{ch}^{\mathrm{OL}'}(G) = \mathrm{ch}^{\mathrm{OL}}(L(G)).$$

Observe that the aforementioned question "Can one prepare a scheduling strategy, which guarantees that no task is postponed more than k times?" has now a simpler statement "$\mathrm{ch}^{\mathrm{OL}}(G) \leqslant k+1$?"

We will stick to the terminology of the PC-game for the rest of the thesis. We will of course remember that good strategies for Mrs. Correct describe desirable real time scheduling strategies and we will favour her over Mr. Paint.

We need to take a closer look into the structure of the gameplays in the PC-games and the boards they are being played on. Before that we present one more definition. For a given function $h : A \to \mathbb{N}$ and a subset $B \subseteq A$ we define a function $h_{\downarrow B} : A \to \mathbb{N}$ by:

$$h_{\downarrow B}(a) = \begin{cases} h(a) - 1, & \text{if } a \in B \text{ and } h(a) > 0, \\ h(a), & \text{otherwise.} \end{cases}$$

Suppose that the PC-game has started on the board (G, f, g). In the first round Mr. Paint presented a set P_1 and Mrs. Correct assigned the colour 0 to the vertices in an independent set $C_1 \subseteq P_1$. Observe that the rest of the game can be viewed as a PC-game on the board $(G, f_{\downarrow P_1}, g_{\downarrow C_1})$.

Now we argue that one can focus on strategies which do not track the history of the gameplay and make decisions in the i-th round based only on the state of the game, i.e. taking into consideration only the following information:

- $f^* : V \to \mathbb{N}$ with $f^*(v)$ giving the number of free positions on the list $L(v)$, i.e. $f^*(v) = f(v) - |L_{i-1}(v)|$,
- $g^* : V \to \mathbb{N}$ with $g^*(v)$ giving the number of colours missing in the colouring $\varphi(v)$, i.e. $g^*(v) = g(v) - |\varphi_{i-1}(v)|$.

Suppose $\mathcal{Q}$ is a winning strategy for Mr. Paint in PC-game on the board (G, f, g). Consider all possible gameplays of $\mathcal{Q}$ starting on the board (G, f, g). For any fixed $f^* \leqslant f$ and $g^* \leqslant g$ there are two possibilities. The first is that the state of the game (G, f^*, g^*) is never reached – we do not need to care about such f^* and g^*. The second possibility is that the strategy $\mathcal{Q}$ makes moves on the board (G, f^*, g^*) possibly depending on the history of the gameplay. We can simplify $\mathcal{Q}$ to always make one, arbitrarily chosen, of those moves independent of the history. If we do that for all choices of f^* and g^*, then the resulting strategy is also a winning strategy and it makes decisions based only on the state of the game.

Therefore for a fixed graph $G = (V, E)$, a strategy $\mathcal{Q}$ for Mr. Paint can be encoded in a (partial) function

$$\mathcal{Q} : \mathbb{N}^V \times \mathbb{N}^V \to \mathcal{P}^+(V)$$

satisfying the condition $\mathcal{Q}(f^*, g^*) \subseteq \{v \in V : f^*(v) > 0\}$. The strategy $\mathcal{Q}$ tells Mr. Paint to present a set $P = \mathcal{Q}(f^*, g^*)$ on the board (G, f^*, g^*).

We give a recursive definition of a winning position for the strategy $\mathcal{Q}$:

- (G, f^*, g^*) is a winning position for the strategy $\mathcal{Q}$ when $f^* \equiv 0$ and $g^* \not\equiv 0$,
- (G, f^*, g^*) is a winning position for the strategy $\mathcal{Q}$ if for each independent subset C of $P = \mathcal{Q}(f^*, g^*)$ the position $(G, f^*{}_{\downarrow P}, g^*{}_{\downarrow C})$ is a winning position for $\mathcal{Q}$.

Similarly, a strategy $\mathcal{S}$ for Mrs. Correct can be encoded in a (partial) function

$$\mathcal{S} : \mathbb{N}^V \times \mathbb{N}^V \times \mathcal{P}^+(V) \to \mathcal{P}(V)$$

satisfying the condition that $\mathcal{S}(f^*, g^*, P)$ is an independent subset of P. The strategy $\mathcal{S}$ tells Mrs. Correct to respond with a set $C = \mathcal{S}(f^*, g^*, P)$ to Mr. Paint's move P on the board (G, f^*, g^*). We give a recursive definition of a winning position for the strategy $\mathcal{S}$:

- (G, f^*, g^*) is a winning position for the strategy $\mathcal{S}$ when $g^* \equiv 0$,
- (G, f^*, g^*) is a winning position for the strategy $\mathcal{S}$ if for each nonempty P subset of $\{v \in V : f^*(v) > 0\}$ and $C = \mathcal{S}(f^*, g^*, P)$ the position $(G, f^*{}_{\downarrow P}, g^*{}_{\downarrow C})$ is a winning position for $\mathcal{S}$.

Observe that a graph G is on-line (f, g)-choosable if and only if (G, f, g) is a winning position for some strategy $\mathcal{S}$ for Mrs. Correct. On the other hand G is not on-line (f, g)-choosable if and only if (G, f, g) is a winning position for some strategy $\mathcal{Q}$ for Mr. Paint.

We are going to present a number of strategies for Mrs. Correct and Mr. Paint. We will present those strategies as algorithms written in pseudocode. Special instructions **read** and **print** denote the communication with the other player – presenting its own move (**print**) and waiting for the other player's move (**read**).

Algorithm 1.2 is an example of a strategy for Mrs. Correct. It is an on-line version of Algorithm 1.1 and it also exploits the colouring order R. If a vertex $v \in P$ is not included in the response C then one of its neighbours preceding v in R is in C. There are at most $\operatorname{col}(G) - 1$ such neighbours, so Algorithm 1.2 assigns colour to v after at most $\operatorname{col}(G)$ occurrences of v in Mr. Paint's moves. This leads to:

$$\operatorname{ch}^{\text{OL}}(G) \leqslant \operatorname{col}(G).$$

Algorithm 1.2 On-Line-Colouring-Order-Greedy

inputs
$G = (V, E)$ {*graph*}
$R = (v_0, v_1, \ldots, v_{n-1})$ {*colouring order*}
do
while true do
read P {*Mr. Paint's move*}
$C = \emptyset$
for $j = 0$ **to** $n - 1$ **do**
if $v_j \in P$ and $C \cup \{v_j\}$ is independent **then**
$C = C \cup \{v_j\}$ {*add vertex v_j to C if none of his neighbours preceding v_j in R is already in C*}
end if
end for
print C {*Mrs. Correct's move*}
end while

2

Warm up

The idea behind this warm up chapter is to give some intuition about the possible approaches to the game "Mr. Paint and Mrs. Correct". We present some simple observations followed by various results published by Schauz [**Sch09**] and Zhu [**Zhu09**]. Later, we present a modification, of the original PC-game, inspired by a scheduling setting.

2.1. Observations

Observation 2.1. If a graph $G = (V, E)$ is on-line (f, g)-choosable and

- $G' = (V', E')$ is a subgraph of G,
- $f' : V' \to \mathbb{N}$ satisfies $f'(v) \geqslant f(v)$ for all $v \in V'$,
- $g' : V' \to \mathbb{N}$ satisfies $g'(v) \leqslant g(v)$ for all $v \in V'$,

then G' is on-line (f', g')-choosable.

Proof. Given a winning strategy $\mathcal{S}$ for Mrs. Correct in the PC-game on the board (G, f, g) we construct a winning strategy $\mathcal{S}'$ for Mrs. Correct in the PC-game on the board (G', f', g'). We define two functions: $f''(v) = f'(v) - f(v)$ and $g''(v) = g(v) - g'(v)$ for $v \in V'$ and set

$$\mathcal{S}'(f^*, g^*, P) = \mathcal{S}(f^* - f'', g^* + g'', P).$$

A position (f^*, g^*) is a winning position for $\mathcal{S}'$ whenever $(f^* - f'', g^* - g'')$ is a winning position for $\mathcal{S}$. □

The proof of Observation 2.1 allows the construction of a strategy $\mathcal{S}'$ for Mrs. Correct in the PC-game on the board (G', f', g') using a winning strategy $\mathcal{S}$ for Mrs. Correct in the PC-game on the board (G, f, g). For its own purposes $\mathcal{S}'$ behaves as if the situation in the game was less favourable than it really is, and relies on the fact that $\mathcal{S}$ knows what to do in those less favourable situations. Later on, in arguments like this we will not specify the details of the function $\mathcal{S}'$ and simply say that the strategy $\mathcal{S}'$ uses the strategy $\mathcal{S}$.

Observation 2.2. Given a graph G and a vertex $v \in G$ we have

$$\mathrm{ch}^{\mathrm{OL}}(G - v) \geqslant \mathrm{ch}^{\mathrm{OL}}(G) - 1.$$

PROOF. Suppose that $\mathrm{ch}^{\mathrm{OL}}(G - v) = k$ is witnessed by Mrs. Correct's winning strategy $\mathcal{S}$ in the PC-game on the board $(G - v, k, 1)$. The following strategy is a winning strategy for her in the PC-game on the board $(G, k + 1, 1)$. For her own purposes, she is going to play a second PC-game B which starts on the board $(G - v, k, 1)$.

In the i-th round, when Mr. Paint presents P_i, Mrs. Correct either sets $C_i = \{v\}$ if $v \in P_i$, or copies the move P_i to the game B and responds with the same independent subset as the strategy $\mathcal{S}$ in the game B. The first case occurs only once during the game. Since the list-length-function on $G - v$ is one higher than required by the strategy $\mathcal{S}$, Mrs. Correct can safely use $\mathcal{S}$. □

The proof of Observation 2.2 allows the construction of a strategy $\mathcal{S}'$ for Mrs. Correct in the PC-game on the board $(G, k + 1, 1)$ using a winning strategy $\mathcal{S}$ for Mrs. Correct in the PC-game on the board $(G - v, k, 1)$. The strategy $\mathcal{S}$ plays on a subgraph of G and some of the moves from the real game are altered before they are presented to $\mathcal{S}$. We count the number of such alterations and if it does not exceed the difference between the list-length-function in the real game and the one in which the strategy $\mathcal{S}$ works then Mrs. Correct can safely use $\mathcal{S}$. The proof of the next lemma exploits the same technique.

Lemma 2.3. For any two graphs G and H, we have

$$\mathrm{ch}^{\mathrm{OL}}(G \times H) \leqslant \mathrm{ch}^{\mathrm{OL}}(G) + \mathrm{col}(H) - 1.$$

PROOF. We present a proof which is similar to the one presented by Borowiecki et al. [**BJKM06**] for the off-line case. We are going to use a winning strategy $\mathcal{S}$ for Mrs. Correct in the PC-game on the board $(G, \mathrm{ch}^{\mathrm{OL}}(G), 1)$ and a colouring order $(h_0, \ldots, h_{n-1})$ of the vertices of H to get a strategy for Mrs. Correct in the PC-game on the board $(G \times H, \mathrm{ch}^{\mathrm{OL}}(G) + \mathrm{col}(H) - 1, 1)$.

Mrs. Correct is going to play in n independent PC-games on boards $(G \times \{h_j\}, \mathrm{ch}^{\mathrm{OL}}(G), 1)$ for $j = 0, \ldots, n - 1$ using the strategy $\mathcal{S}$. In the i-th round, set P_i is presented to Mrs. Correct. She presents the set $P_i \cap (G \times \{h_0\})$ to the strategy $\mathcal{S}$ in the game on $G \times \{h_0\}$. The strategy $\mathcal{S}$ responds with a set S_0. Mrs. Correct adds the vertices in S_0 to her

response C_i. She also starts constructing the set F by putting into it all the neighbours of vertices in S_0 in the graph $G \times H$.

Then she presents the set $(P_i \setminus F) \cap (G \times \{h_1\})$ to the strategy $\mathcal{S}$ playing on $G \times \{h_1\}$. The strategy $\mathcal{S}$ responds with S_1. Mrs. Correct adds the vertices in S_1 to her response C_i. She also updates the set F by adding the neighbours of the vertices in S_1 in the graph $G \times H$ to the set F.

She continues this procedure for $G \times \{h_2\}, \ldots, G \times \{h_{n-1}\}$. A vertex in $G \times \{h_j\}$ has at most $\mathrm{col}(H) - 1$ neighbours among the vertices in graphs $G \times \{h_k\}$ for $k = 0, \ldots, j-1$. Thus, a vertex is present in at most $\mathrm{col}(H) - 1$ many of the sets P_i without being presented to the strategy $\mathcal{S}$ in the game on $G \times \{h_j\}$. The strategy $\mathcal{S}$ assigns the colour to any vertex after at most $\mathrm{ch}^{\mathrm{OL}}(G)$ occurrences. Combining these two facts, any vertex has a colour assigned after at most $\mathrm{ch}^{\mathrm{OL}}(G) + \mathrm{col}(H) - 1$ occurrences. □

2.2. Previous results

The most intriguing result in the on-line choosability field is the adoption of Alon's algebraic method of Combinatorial Nullstellensatz. Schauz **[Sch10a, Sch10b]** showed that all results concerning choosability that can be obtained using Combinatorial Nullstellensatz hold for on-line choosability as well. We refer the interested reader to see the Alon-Tarsi type theorems **[Sch10a]**, adoption of Combinatorial Nullstellensatz **[Sch10b]** and Brooks' type theorem **[HKS10]**. Other methods commonly used in the list colouring field include the colouring extension method, the kernel method and various random methods. The first two can be adapted to the on-line choosability and we present some of the results of these types. The results in Chapter 4 have some random flavour but random methods used in the proofs of various upper bounds for choosability do not easily adapt to the on-line case.

Theorem 2.4. **[Sch09]** For a planar graph G we have:

$$\mathrm{ch}^{\mathrm{OL}}(G) \leqslant 5.$$

The proof of Theorem 2.4 uses the colouring extension technique used in the proof of Thomassen's Theorem **[Tho94]** which needs to be carefully adjusted to the on-line setting.

Theorem 2.5. [**Sch09**] For a bipartite graph G we have:

$$\mathrm{ch}^{\mathrm{OL}'}(G) = \chi'(G).$$

Galvin [**Gal95**] originally proved that for a bipartite graph G we have $\mathrm{ch}'(G) = \chi'(G)$. Upon close examination of his argument it is clear that it also shows $\mathrm{ch}^{\mathrm{OL}'}(G) = \chi'(G)$.

To state the next two theorems we need a couple of definitions. The core of a connected graph G is the graph obtained from G by successively deleting vertices of degree 1. A graph $\Theta_{a,b,c}$ is a graph consisting of two vertices of degree 3 joined by three internally disjoint paths of respective lengths a, b and c (see Figure 2.6).

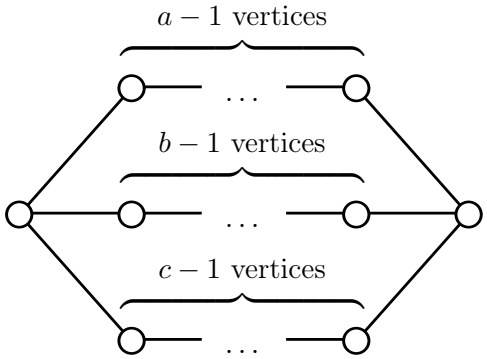


FIGURE 2.6. Graph $\Theta_{a,b,c}$

Theorem 2.7. [**ERT80**] A connected graph G is 2-choosable if and only if the core of G is one of the following graphs: K_1, C_{2n} or $\Theta_{2n,2,2}$.

Theorem 2.8. [**Zhu09**] A connected graph G is on-line 2-choosable if and only if the core of G is one of the following graphs: K_1, C_{2n} or $\Theta_{2,2,2} = K_{2,3}$.

Theorem 2.9. [**Zhu09**] For a graph G with n vertices we have:

$$\mathrm{ch}^{\mathrm{OL}}(G) \leqslant \chi(G) \cdot \ln n + 1.$$

This bound follows from the investigation of a clever algorithm described by Zhu and improves all previously known bounds for $\mathrm{ch}(G)$.

2.3. Respawning vertices

In this section we consider a modification of the PC-game, "Mr. Paint and Mrs. Correct with vertex respawning" (PC$^{\text{vr}}$-game for short), inspired by the following scheduling scenario.

To introduce the new scheduling scenario, suppose that each task needs to be scheduled not once but possibly infinitely many times. In the scheduling scenario modelled by "Mr. Paint and Mrs. Correct" we wish to guarantee that each task gets scheduled after at most k rounds it is available in. In the new scenario for each task t we wish to guarantee that if $[i, \ldots, j]$ is an interval of rounds such that the task t was available for scheduling k times in the rounds from i to j then the task t is scheduled at least once in those rounds. This scheduling scenario is modelled by the following on-line game.

The PC$^{\text{vr}}$-game is being played on a graph G with a fixed list-length-function f. It is like a PC-game on the board $(G, f, 1)$ but in PC$^{\text{vr}}$-game the game is possibly infinite and Mrs. Correct assigns colours to vertices many times during the game. Whenever Mrs. Correct assigns a colour to a vertex v, it gets immediately respawned with an empty list and no colour assigned. More precisely, when Mrs. Correct responds with an independent set C to Mr. Paint's move P on the board $(G, f^*, 1)$ then the game continues on the board $(G, f^{**}, 1)$ where list-length-function f^{**} is defined by:

$$f^{**}(v) = \begin{cases} f(v), & v \in C, \\ f^*(v) - 1, & v \in P \setminus C, \\ f^*(v), & v \in G \setminus P. \end{cases}$$

If $f^{**}(v) = 0$ then the game ends and Mr. Paint wins. A winning strategy for Mrs. Correct is a strategy which guarantees an infinite gameplay.

The following theorem gives a simple complete characterisation of boards for which Mrs. Correct has winning strategies in PC$^{\text{vr}}$-games.

Theorem 2.10. Mrs. Correct has a winning strategy in the PC$^{\text{vr}}$-game on the board $(G, f, 1)$ if and only if $f(v) > \deg(v)$ for each vertex v.

Proof. We start with a construction of a strategy for Mrs. Correct when $f(v) > \deg(v)$. The strategy is described by Algorithm 2.11. It starts with arranging vertices in an arbitrary order R. The order R is going to change from round to round. In each round Mrs. Correct returns an independent subset C selected greedily according to the current order

R and moves the vertices in C to the end of R. In the i-th round we consider a vertex $v \in P_i$ which was not selected for $\deg(v)$ times since the last respawn of v. Each time v was not selected, some neighbour of v was selected and moved to the end of R. This means that v is now before all of its neighbours in the order R and therefore v will be selected this time.

Algorithm 2.11 GREEDY-ROUND-ROBIN

```
inputs
  G = (V, E) {graph}
do
  R = V {any ordering of V}
  while true do
    read P {Mr. Paint's move}
    C = ∅
    for j = 0 to n − 1 do
      if R[j] ∈ P and C ∪ {R[j]} is independent then
        C = C ∪ {R[j]} {add vertex to C}
      end if
    end for
    print C {Mrs. Correct's move}
    for each v in C do
      REMOVE(R, v) {remove v from R}
      APPEND(R, v) {append v to the end of R}
    end for{move the set C to the end of R}
  end while
```

On the other hand, Algorithm 2.12 brings victory to Mr. Paint whenever there exists a vertex v with $f(v) \leqslant \deg(v)$. First observe that the counter c counts the number of times the vertex v was not selected since its last respawn. If c gets to $\deg(v) + 1 > f(v)$, Mr. Paint wins.

Now set m to be $\max_{i=0,\ldots,\deg(v)-1} f(w_i)$ and consider counters c_i for $i = 0, \ldots, \deg(v) - 1$ counting the number of times the vertex w_i was not selected since its last respawn. We let $M = \sum_{i=0,\ldots,\deg(v)-1} c_i \cdot m^i$ and we examine the values of M in the moments when the counter c gets zeroed. At the very beginning of the game all of the c_i's are zero and $M = 0$ as well. We are going to show that at some point the value of one of the c_i's exceeds the value of m. Now look at a moment when the counter c gets zeroed. The last move of Mr. Paint was $\{v, w_j\}$ for some j. Notice that since the last time the counter c got zeroed, counters $c_0, \ldots, c_{j-1}$

Algorithm 2.12 FORCE-DELTA

inputs
$G = (V, E)$ {*graph*}
$v \in V$ {*any vertex with* $f(v) \leqslant \deg(v)$}
$w_0, \ldots, w_{\deg(v)-1}$ {*neighbours of* v}
do
$c = 0$
while true do
$P = \{v, w_c\}$
print P {*Mr. Paint's move*}
read C {*Mrs. Correct's move*}
if $v \in C$ **then**
$c = 0$
else
$c = c + 1$
end if
end while

got zeroed as well, but the counter c_j increased. Thus, the value of M increases. This means that the value of M is unbounded and at some point one of the counters c_i will be greater than $m \geqslant f(n_i)$ and that Mr. Paint will win the game. □

2.4. Small complete bipartite graphs

The choice number of a complete bipartite graph $K_{n,n}$ is roughly $(1 + o(1)) \log n$ [**ERT80**]. Nevertheless, finding the exact values of choice number, even for small complete bipartite graphs, is not an easy task and received much attention in literature. A characterisation of 3-choosable complete bipartite graphs is available by combining results of many authors.

Theorem 2.13. [**ERT80, MRS91, ST95, O'D95**] Graph $K_{p,q}$ with $p \leqslant q$ is 3-choosable if and only if one of the following holds:

- $p \leqslant 2$
- $p = 3$ and $q \leqslant 26$ [**ERT80**],
- $p = 4$ and $q \leqslant 18$ [**MRS91**],
- $p = 5$ and $q \leqslant 12$ [**ST95**],
- $p = 6$ and $q \leqslant 10$ [**O'D95**].

In his paper Zhu [**Zhu09**] calculated the on-line choice number for some small complete bipartite graphs. Using computer methods, similar to those which we describe in Chapter 3 and Appendix A, we have obtained a complete characterisation of on-line 3-choosable and on-line 4-choosable graphs $K_{p,q}$. We present this characterisation without a proof.

Fact 2.14. Graph $K_{p,q}$ with $p \leqslant q$ is on-line 3-choosable if and only if one of the following holds:

- $p \leqslant 2$,
- $p = 3$ and $q \leqslant 26$,
- $p = 4$ and $q \leqslant 11$,
- $p = 5$ and $q \leqslant 8$,
- $p = 6$ and $q \leqslant 7$.

Fact 2.15. Graph $K_{p,q}$ with $p \leqslant q$ is on-line 4-choosable if and only if one of the following holds:

- $p \leqslant 3$
- $p = 4$ and $q \leqslant 255$,
- $p = 5$ and $q \leqslant 87$,
- $p = 6$ and $q \leqslant 50$,
- $p = 7$ and $q \leqslant 34$,
- $p = 8$ and $q \leqslant 27$,
- $p = 9$ and $q \leqslant 23$,
- $p = 10$ and $q \leqslant 19$,
- $p = 11$ and $q \leqslant 17$,
- $p = 12$ and $q \leqslant 16$,
- $p = 13$ and $q \leqslant 14$.

3

On-line choosability is NP-hard

In this chapter we shed some light on the computational complexity of determining the on-line choice number. Our main result is NP-hardness of the recognition of on-line 3-choosable bipartite graphs. In the short introduction 3.1 we present some basic notions required in the rest of the chapter. Section 3.2 gives an old-school proof that a carefully selected modification of the standard 3SAT problem remains NP-complete. Section 3.4 introduces a few necessary gadgets. Computer methods used to check some properties of those gadgets are presented in detail in Appendix A. In Section 3.5 we present a proof of NP-hardness for on-line choosability problems.

3.1. Introduction

Readers without experience in the standard computational complexity are referred to the classical guides such as [**Pap94**] or [**Sip06**]. The rest of the chapter requires understanding of the notions of complexity classes, and in particular NP and PSPACE, reductions and completeness. The class Π_2^{P} denotes the second level of the polynomial hierarchy ($\mathsf{coNP}^{\mathsf{NP}}$).

The complexity of the (off-line) choosability had been established by Rubin [**ERT80**] and was later examined in more detail by Gutner [**Gut92, Gut96, GT09**].

Problem 3.1. CHOOSABILITY:
Input: A graph $G = (V, E)$ and a number $k \in \mathbb{N}$.
Question: Is G k-choosable?

Theorem 3.2. [**ERT80, Gut92**] CHOOSABILITY is Π_2^{P}-complete.

It is rather easy to see that CHOOSABILITY belongs to the class Π_2^{P}. Indeed, the statement $\mathrm{ch}\,(G) \leqslant k$ can be, using the standard tool set,

transformed to the formula:

$$\forall \bar{x} \ \exists \bar{y} \ \varphi_{G,k}(\bar{x}, \bar{y}),$$

where $\varphi_{G,k}(\bar{x}, \bar{y})$ is a quantifier-free formula checking that $\bar{y}$ encodes a proper colouring of G from $\bar{x}$ whenever $\bar{x}$ encodes a k-long list assignment. The proof of Π_2^{P}-hardness is more difficult. ChoosABILITY remains Π_2^{P}-complete even for very restricted classes of graphs and small values of k [**Gut92**].

It is natural to expect that the on-line version of ChoosABILITY is harder. We define the problem as follows:

Problem 3.3. OnLineChoosability:
Input: A graph $G = (V, E)$ and a number $k \in \mathbb{N}$.
Question: Is G on-line k-choosable?

We start our investigation of the complexity with an easy upper bound.

Lemma 3.4. OnLineChoosability is in PSPACE.

Proof. Consider a graph G on n vertices and a number k given on the input. We are going to bound the memory space needed by the algorithm in terms of a polynomial of n. If $k \geqslant n$ then G is trivially on-line k-choosable and we are going to assume that $k \leqslant n$.

Observe, that a state of the PC-game (f^*, g^*) can be stored using $2n$ integers (with values bounded by $k \leqslant n$). The game ends after at most kn rounds so that we can bound the depth of recursion in Recursive-On-Line-Choosability (Algorithm 3.5) by n^2. Thus the total space requirement is bounded by a polynomial of n. □

We are going to prove NP-hardness of the following restriction of OnLineChoosability:

Problem 3.6. OnLineBipartite3Choosability:
Input: A bipartite graph $G = (V, E)$.
Question: Is G on-line 3-choosable?

3.2. 2Pos2Neg3SAT

In this section we introduce a restriction of the standard SAT problem and prove that it remains NP-complete.

Algorithm 3.5 Recursive-On-Line-Choosability

inputs
$G = (V, E)$ {*graph*}
$f^* : V \to \mathbb{N}$ {*list-length-function*}
$g^* : V \to \mathbb{N}$ {*colour-demand-function*}
outputs
value: **true** if Mrs. Correct has a winning strategy on the board (G, f^*, g^*) and **false** otherwise
do
if $g^* \equiv 0$ **then**
value = **true** {*nothing left to colour*}
return *value*
else if $f^* \equiv 0$ **then**
value = **false** {*something left to colour, but game finished*}
return *value*
end if
value = **true**
for all $P \subseteq \{v \in V : f^*(v) > 0\}$ **do**
value = **false**
for all independent $C \subseteq P$ **do**
if Recursive-On-Line-Choosability $(G, f^*{}_{\downarrow P}, g^*{}_{\downarrow C})$ **then**
value = **true**
end if
end for
if *value* = **false then**
return *value*
end if
end for
return *value*

Problem 3.7. 2Pos2Neg3SAT:
Input: A Boolean formula φ in the conjunctive normal form with each clause containing 2 or 3 literals and each variable occurring in exactly 4 different clauses, 2 times positively and 2 times negatively.
Question: Is φ satisfiable?

S similar problem (3,4)SAT, in which each clause contains exactly 3 literals and each variable occurs in exactly 4 different clauses, was examined by Tovey [**Tov84**] and was established to be NP-complete. However, in (3,4)SAT there are no constraints on the number of positive/negative

occurrences of variables. Those constraints play important role in our final proof of NP-hardness of OnLineBipartite3Choosability.

Lemma 3.8. 2Pos2Neg3SAT is NP-complete.

Proof. Given a 3SAT formula ψ we are going to transform it into a 2Pos2Neg3SAT formula φ without altering the satisfiability. We achieve this by applying the following transformations.

First we decrease the number of occurrences of the variables. Given a variable x that has $r > 2$ occurrences we introduce r new variables $x_0, \ldots, x_{r-1}$. We replace each occurrence of x in ψ with an occurrence of x_i so that each x_i is used only once. The variable x is no longer used in the formula. Then we add the following clause for each $i = 0, \ldots, r-1$:

$$\neg x_i \vee x_{(i+1) \bmod r}.$$

This set of clauses is equivalent to $x_0 \Rightarrow x_1 \Rightarrow \ldots \Rightarrow x_{r-1} \Rightarrow x_0$ and secures that any truth assignment satisfying those clauses gives the same value to each of the variables $x_0, \ldots, x_{r-1}$.

Observe that after applying the previous transformation to each variable in ψ, the new formula is satisfiable if and only if ψ is satisfiable. In this new formula each variable occurs at most 3 times and variables with 3 occurrences (these are the x_i's) have both positive and negative occurrences.

The only thing left to do is to add the missing occurrences. Consider any variable y with less than 4 occurrences. It has some occurrences missing – positive, negative or possibly both. Let y^* be a missing positive or negative occurrence. We introduce 5 new variables a, b, c, d, e and the following 7 new clauses:

$$\begin{aligned} y^* \vee \quad a \vee \quad b, \\ a \vee \neg d \vee \neg e, \\ b \vee \neg d \vee \neg e, \\ c \vee \quad d \vee \neg a, \\ c \vee \quad e \vee \neg a, \\ d \vee \neg b \vee \neg c, \\ e \vee \neg b \vee \neg c. \end{aligned}$$

New clauses are satisfiable independent of the valuation of y^* (one can set the value of the variables a, b, c, d, e to **true**) and each of the new

variables has 4 occurrences – two positive and two negative. After applying the last transformation to all variables with less than 4 occurrences, the new formula φ is a 2Pos2Neg3SAT formula which is satisfiable if and only if ψ is satisfiable. □

3.3. Power moves

Before proceeding further we introduce a new, quite technical, modification of the PC-game. We call it "Mr. Paint and Mrs. Correct with power moves" ($\mathrm{PC}^{\mathrm{pm}}$-game for short). In this game, the board (G, f, g, S, U) consists of a graph $G = (V, E)$, a list-length-function f, a colour-demand-function g, and two nonempty, disjoint subsets of vertices $S, U \subset V$.

Vertices in the subset S are called solid and vertices in the second subset U are called vulnerable. During the game the subset S does not change, but some vertices may be removed from the subset U. The game continues as in a regular PC-game but in the i-th round, after Mrs. Correct has responded with C_i, Mr. Paint may decide to use a power move. The rules of power moves are as follows:

- Mr. Paint chooses a subset $T \subseteq U$ of vulnerable vertices.
- Mrs. Correct changes her response from C_i to C_i' with respect to the following conditions:
  - Mrs. Correct must remove vertices in T from her response. $C_i' \cap T = \emptyset$.
  - Mrs. Correct can't change her response on solid vertices. $C_i' \cap S = C_i \cap S$.
- Vertices in T are no longer vulnerable. $U = U \setminus T$.
- The game continues as if Mrs. Correct has responded with C_i'.

The power moves allow Mr. Paint to "uncolour" each vulnerable vertex once during the game. Mrs. Correct needs to be extra careful when including vulnerable vertices in her responses – she might be forced to modify her move and she will not be able to change her mind about solid vertices when doing so.

3.4. Gadgets

In this section we present three gadgets crucial for the final proof of NP-hardness. Each of those graphs have very specific characteristic of possible strategies for Mr. Paint and for Mrs. Correct. We are going to

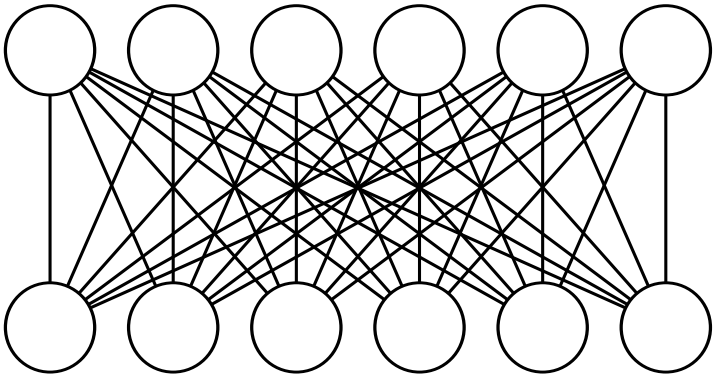


FIGURE 3.9. Gadget G_{crit}

claim those properties in this section. For some of those properties we have obtained combinatorial case analysis proofs, but some of them are simply too complex to analyse by hand. To avoid all of that case analysis we have decided to present a computer assisted proof of all of the claimed properties.

We have developed a computer program which analyses all possible strategies of Mr. Paint and Mrs. Correct in the PC-games and PC^{pm}-games on gadgets and checks that the stated properties hold. The program is based on Algorithm 3.5 used in the proof of Lemma 3.4. In Appendix A we discuss an effective implementation and present a C++ code of the program.

Fact 3.10. A gadget G_{crit} shown on Figure 3.9 is a complete bipartite graph $K_{6,6}$. It has the following properties:

- $\text{ch}^{\text{OL}}(G_{\text{crit}}) = 3$.
- For each vertex a and a list-length-function f defined by:

$$f(v) = \begin{cases} 2, & \text{for } v = a, \\ 3, & \text{otherwise,} \end{cases}$$

G_{crit} is not on-line f-choosable.

PROOF. See Section A.1 for instructions on running the proof on a computer. □

Corollary 3.11. For a given graph $G = (V, E)$ and a vertex $b \in V$ we construct a graph H by adding a copy of G_{crit} and one additional edge joining one vertex a in G_{crit} and the vertex b. Let h be a list-length-function for the graph H satisfying

$$h(v) = 3, \text{ when } v \in G_{\text{crit}},$$

and f be a list-length-function for the graph G defined by:

$$f(v) = \begin{cases} h(v) - 1, & \text{for } v = b, \\ h(v), & \text{otherwise.} \end{cases}$$

The graph G is on-line f-choosable if and only if the graph H is on-line h-choosable.

PROOF. We start with a proof that H is on-line h-choosable whenever G is on-line f-choosable. The strategy for Mrs. Correct is as follows. She is going to use a winning strategy $\mathcal{S}$ in the PC-game on the board $(G_{\text{crit}}, 3, 1)$, supplied by Fact 3.10, and a winning strategy $\mathcal{T}$ in the PC-game on the board $(G, f, 1)$. In the i-th round when Mr. Paint reveals P_i, she presents $P_i \cap G_{\text{crit}}$ to $\mathcal{S}$. This strategy tells Mrs. Correct to respond with a set S_i. If $a \notin S_i$ then she simply presents $P_i \cap G$ to $\mathcal{T}$. If $a \in S_i$ then she presents $(P_i \cap G) \setminus \{b\}$ to $\mathcal{T}$. In either case she sets her response C_i to be the union of the responses returned by $\mathcal{S}$ and $\mathcal{T}$. Observe that the second case ($a \in S_i$) happens only once in the whole game and the condition $h(b) = f(b) + 1$ allows Mrs. Correct to safely use the strategy $\mathcal{T}$.

When Mr. Paint has a winning strategy in the PC-game on the board $(G, f, 1)$, then the following strategy brings him victory in the PC-game on the board $(H, h, 1)$. In his first move Mr. Paint presents the set $\{a, b\}$. Mrs. Correct may respond with at most one of the vertices a or b. If she does not include b in her response then Mr. Paint can win on the board $(G, f, 1)$. If she does not include a in her response then, by Fact 3.10, Mr. Paint can win on G_{crit}. □

Corollary 3.11 gives us a way to "lower" the value of a list-length-function on any vertex by attaching a copy of G_{crit} to it. Given a graph G and a list-length-function f with values not exceeding 3 we can extend G to a graph H by attaching $3 - f(v)$ copies of G_{crit} to each vertex v of G. The graph G is on-line f-choosable if and only if the graph H is on-line

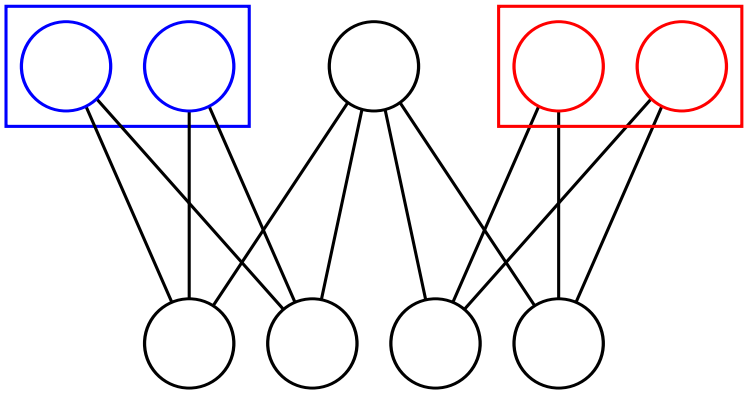


FIGURE 3.12. Gadget G_{cons}

3-choosable. Thus in the proof of NP-hardness of ONLINEBIPARTITE-3CHOOSABILITY we may use gadgets with list-length-functions not limited to taking the value 3.

Fact 3.13. A gadget G_{cons} shown on Figure 3.12 is a bipartite graph on 9 vertices with two two-element subsets of vertices – red and blue. It has the following properties:

- $\text{ch}^{\text{OL}}(G_{\text{cons}}) = 3$.
- For a list-length-function f defined either by:

$$f(v) = \begin{cases} 3, & \text{when } v \text{ is blue,} \\ 2, & \text{otherwise,} \end{cases}$$

  or by:

$$f(v) = \begin{cases} 3, & \text{when } v \text{ is red,} \\ 2, & \text{otherwise,} \end{cases}$$

  G_{cons} is on-line f-choosable.
- For any red vertex a and any blue vertex b and a list-length-function f defined by:

$$f(v) = \begin{cases} 3, & \text{when } v = a \text{ or } v = b, \\ 2, & \text{otherwise,} \end{cases}$$

  G_{cons} is not on-line f-choosable.

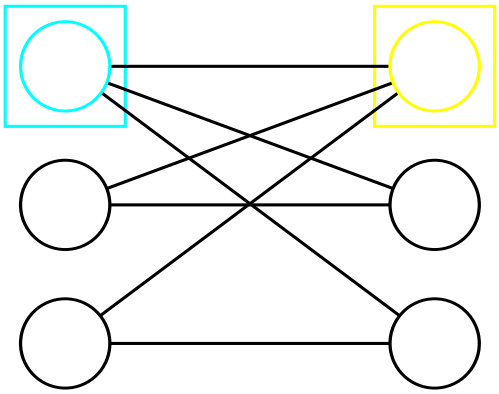


FIGURE 3.14. Gadget G_{choice}

PROOF. See Section A.2 for instructions on running the proof on a computer. □

Fact 3.15. A gadget G_{choice} shown on Figure 3.14 is a bipartite graph on 6 vertices with two one-element subsets of vertices – cyan X and yellow Y. It has the following properties:

- $\text{ch}^{\text{OL}}(G_{\text{choice}}) = 3$, in particular Mr. Paint has a winning strategy in the PC-game on the board $(G_{\text{choice}}, 2, 1)$.
- For a list-length-function f defined by:

$$f(v) = \begin{cases} 3, & \text{when } v \text{ is cyan or yellow,} \\ 2, & \text{otherwise,} \end{cases}$$

  Mrs. Correct has winning strategies in the PC^{pm}-games on the boards $(G_{\text{choice}}, f, 1, X, Y)$ and $(G_{\text{choice}}, f, 1, Y, X)$.

PROOF. See Section A.3 for instructions on running the proof on a computer. □

Corollary 3.16. For a given graph $G = (V, E)$ and its two vertices $a, b \in V$ we construct a graph H by adding a copy of G_{choice} and two additional edges: one joining a with the cyan vertex in G_{choice} and the

other one joining b with the yellow vertex in G_{choice}. Let h be a list-length-function for the graph H satisfying

$$h(v) = \begin{cases} 3, & \text{when } v \text{ is yellow or cyan,} \\ 2, & \text{for other vertices in } G_{\text{choice}}, \end{cases}$$

and f_1,f_2 be two list-length-functions for the graph G defined by:

$$f_1(v) = \begin{cases} h(v) - 1, & \text{when } v = a, \\ h(v), & \text{for other vertices in } G, \end{cases}$$

$$f_2(v) = \begin{cases} h(v) - 1, & \text{when } v = b, \\ h(v), & \text{for other vertices in } G. \end{cases}$$

The graph H is on-line h-choosable if and only if the graph G is on-line f_1-choosable or on-line f_2-choosable.

PROOF. First we show a strategy for Mr. Paint in the PC-game on the board $(H, h, 1)$ when G is neither on-line f_1-choosable nor on-line f_2-choosable. Let $\mathcal{Q}$ be a winning strategy for Mr. Paint in the PC-game on the board $(G_{\text{choice}}, 2, 1)$ supplied by Fact 3.15. The list-length-function h gives 3 to the cyan and the yellow vertex, nevertheless Mr. Paint still follows the strategy $\mathcal{Q}$ (as if $h \equiv 2$) up to the point when one of the cyan or the yellow vertex has been presented twice by Mr. Paint and still has no colour assigned by Mrs. Correct. If this does not happen then Mrs. Correct is playing as if h on both the cyan and the yellow vertex was 2 and she is going to loose the game on G_{choice} and therefore on H.

Thus, we are left with the cyan or the yellow vertex already presented twice by Mr. Paint and remaining unpainted by Mrs. Correct. Suppose first that this is the cyan vertex. Mr. Paint presents now the set that consists of the cyan vertex and the vertex a. Mrs. Correct must choose to assign colour to the cyan vertex and a can receive no colour this time. After this move the situation on G is described by the list-length-function f_1 and Mr. Paint has a winning strategy.

In the second case, when the yellow vertex was presented twice by Mr. Paint and has no colour assigned, Mr. Paint plays the yellow vertex and the vertex b leading to a situation on G described by the list-length-function f_2.

Now we show a strategy for Mrs. Correct in the PC-game on the board $(H, h, 1)$ when G is on-line f_1-choosable. The case of G being on-line f_2-choosable is symmetric with respect to the role played by the cyan and the yellow vertex.

The list-length-function h on G_{choice} is exactly as needed in Fact 3.15. Let $\mathcal{S}$ be a winning strategy for Mrs. Correct in the PC^{pm}-game on G_{choice} with the yellow vertex being vulnerable and the cyan vertex being solid. Let $\mathcal{T}$ be a winning strategy for Mrs. Correct in the PC-game on the board $(G, f_1, 1)$.

In the i-th round of the PC-game on H Mr. Paint presents his move P_i. Mrs. Correct uses the strategy $\mathcal{S}$ on G_{choice} as if Mr. Paint played $P_i \cap G_{\text{choice}}$. The strategy $\mathcal{S}$ tells Mrs. Correct to respond with some set S. If the cyan vertex is not in S then Mrs. Correct uses the strategy $\mathcal{T}$ on G as if Mr. Paint played $P_i \cap G$. Otherwise, she uses the strategy $\mathcal{T}$ on G as if Mr. Paint played $(P_i \setminus \{a\}) \cap G$. The second case will occur only once during the whole game and the condition $f_1(a) = h(a) - 1$ allows Mrs. Correct to safely use the strategy $\mathcal{T}$.

In either case it may happen that the strategy $\mathcal{T}$ tells Mrs. Correct to include the vertex b in her response. This is the time to use a power move on the yellow vertex, which guarantees that the yellow vertex and the vertex b are not chosen simultaneously. The strategy $\mathcal{S}$ tells how to respond to the power move which results in changing the decision about some vertices in G_{choice}. It is important that the decision about the solid cyan vertex does not change, as it would possibly alter the set presented to the strategy $\mathcal{T}$. □

3.5. NP-hardness

Finally, we are in a position to prove our main result of this chapter.

Theorem 3.17. OnLineBipartite3Choosability is NP-hard.

Proof. We present a reduction from 2Pos2Neg3SAT to OnLine-Bipartite3Choosability. We start with a 2Pos2Neg3SAT formula φ and construct a graph G_φ in the following way. For each variable x we introduce a copy of G_{cons} and call it G_x. For each clause C we introduce a single vertex v_C. Now, when a variable x occurs in a clause C then we introduce a copy of G_{choice} and call it $G_{x,C}$. We join the yellow vertex of $G_{x,C}$ with the vertex v_C. To express the fact that the variable x occurs positively (negatively) in C we pick $v_{x,C}$ to be one of the red

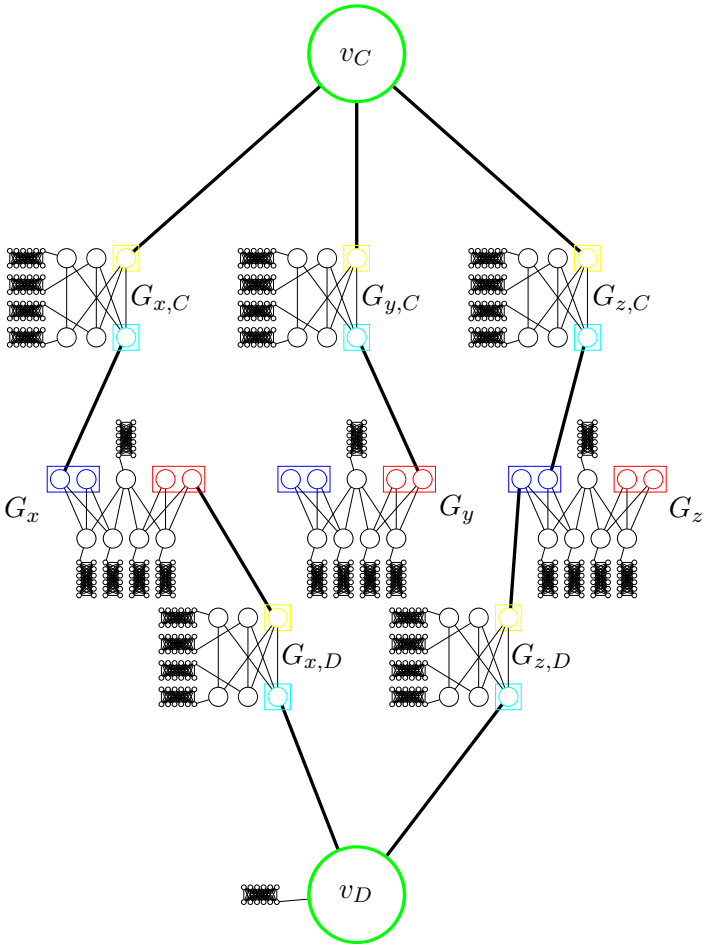


FIGURE 3.18. Transformation of the clauses $C = (\neg x, y, \neg z)$ and $D = (x, \neg z)$.

(blue) vertices of G_x and join it with the cyan vertex of $G_{x,C}$. The fact that φ is a 2POS2NEG3SAT formula allows us to use each of the red and blue vertices in the copies of G_{cons} exactly once. Finally, we add copies of G_{crit} and link them to each of the vertices of all the G_x's and all the $G_{x,C}$'s that are neither red, blue, cyan, nor yellow. Furthermore, for each clause C with 2 literals we link a copy of G_{crit} to the vertex v_C. A sample transformation is shown in Figure 3.18.

We claim that the resulting graph G_φ is on-line 3-choosable if and only if the formula φ is satisfiable. We start the analysis of on-line 3-choosability of G_φ by removing all of the gadgets G_{crit} from the scene. Using Corollary 3.11 we remove them and lower the list-length-function from 3 to 2 on all vertices in the gadgets but red, blue, cyan and yellow, as well as on the vertices v_C for each clause C with 2 literals.

Consider removing the gadget of the form $G_{x,C}$. Corollary 3.16 allows a removal of $G_{x,C}$ in two possible ways. The first one lowers the list-length-function on v_C and the second one lowers the list-length-function on $v_{x,C}$. Think of the second one as "x points at C."

If there are total ℓ occurrences of the variables in the clauses then there are 2^ℓ different ways to remove all the $G_{x,C}$'s. Consider one of those ways and let H be the remaining graph and h be the resulting list-length-function. The graph H, consisting of the isolated vertices v_C and the graphs G_x, is on-line h-choosable if and only if the following conditions are met:

- For each clause C we have $h(v_C) \geqslant 1$, meaning that "at least one variable points at C."
- For each variable x, by Fact 3.13, the function h is not decreased simultaneously from 3 to 2 on a blue and a red vertex, meaning that "the variable x does not simultaneously point at a clause with a positive occurrence of x and at a clause with a negative occurrence of x."

Assume that all of the above conditions are met and consider the following valuation ξ of variables:

- $\xi(x) = \textbf{true}$, if a variable x points at clauses in which it occurs positively,
- $\xi(x) = \textbf{false}$, if a variable x points at clauses in which it occurs negatively,
- $\xi(x)$ is set arbitrarily to **true** or **false**, if a variable x does not point at any clause.

Observe that the formula φ is satisfied by the valuation ξ. Indeed, any clause C is satisfied by one of the variables that points at C.

On the other hand, if φ is satisfiable, given a satisfying valuation of variables, one can remove the gadgets $G_{x,C}$ so that variables point at clauses in which their occurrences evaluate to **true**. This removal leads to the conclusion that G_φ is on-line 3-choosable. □

4

Fractional on-line choosability[1]

For a graph G we define the following fractional chromatic parameters: the fractional chromatic number $\chi_{\mathrm{F}}(G)$, the choice ratio $\mathrm{ch}_{\mathrm{F}}(G)$ and the on-line choice ratio $\mathrm{ch}_{\mathrm{F}}^{\mathrm{OL}}(G)$.

$$\chi_{\mathrm{F}}(G) = \inf\left\{\frac{a}{b} : G \text{ is } (a,b)\text{-colourable}\right\},$$
$$\mathrm{ch}_{\mathrm{F}}(G) = \inf\left\{\frac{a}{b} : G \text{ is } (a,b)\text{-choosable}\right\},$$
$$\mathrm{ch}_{\mathrm{F}}^{\mathrm{OL}}(G) = \inf\left\{\frac{a}{b} : G \text{ is on-line } (a,b)\text{-choosable}\right\}.$$

The following inequalities between the six chromatic parameters follow directly from their definitions:

$$\begin{array}{ccccc} \chi_{\mathrm{F}}(G) & \leqslant & \mathrm{ch}_{\mathrm{F}}(G) & \leqslant & \mathrm{ch}_{\mathrm{F}}^{\mathrm{OL}}(G) \\ \wedge\!\!\backslash & & \wedge\!\!\backslash & & \wedge\!\!\backslash \\ \chi(G) & \leqslant & \mathrm{ch}(G) & \leqslant & \mathrm{ch}^{\mathrm{OL}}(G) \end{array}$$

In this chapter we answer positively the question of Zhu [**Zhu09**] by proving that for any given graph the choice ratio and the on-line choice ratio coincide. On the other hand it is known from the paper of Alon et al. [**ATV97**] that the choice ratio equals the fractional chromatic number.

Lemma 4.1. [**ATV97**] For any graph G we have $\chi_{\mathrm{F}}(G) = \mathrm{ch}_{\mathrm{F}}(G)$.

Lemma 4.2. [**ATV97**] For any graph G, the values of the infimum in the definitions of $\chi_{\mathrm{F}}(G)$ and $\mathrm{ch}_{\mathrm{F}}(G)$ are attained.

An important consequence of Lemma 4.2 is that the infimum can be replaced by the minimum in the definitions of the fractional chromatic number and the choice ratio. We show that this is not the case for the on-line choice ratio – there are graphs for which the infimum in the definition of $\mathrm{ch}_{\mathrm{F}}^{\mathrm{OL}}$ is not attained. Both our results are obtained by exploring the

[1] Work presented in this chapter was published by the author [**Gut11**].

strong links between the on-line choice ratio, and a new on-line game with probabilistic flavour which we introduce.

4.1. Balanced Distribution Game

For a better understanding of possible strategies for Mr. Paint and for Mrs. Correct in PC-games we introduce a new game. It is again a two person game, this time between Nominator and Distributor, and we call it the Balanced Distribution Game, or the BD-game for short. Informally speaking, in this game Nominator constructs a sequence (N_i) of subsets of a set Π of participants – one subset at a time. Distributor consecutively assigns goods to the participants in the new subset N_i. More precisely, he chooses a commodity δ_i out of a finite set Γ of commodities, and then distributes only goods of this commodity to the participants in N_i, one good for each of the participants at a time. The number of goods of each commodity is unlimited. Distributor's goal is to ensure that at the end of the game each single participant got a well balanced spectrum of goods, that is, about equally many goods of each commodity.

In the preparatory phase of the game players establish some rules for the next phase of the game. Nominator chooses a finite set Π of participants and a finite set Γ of commodities of goods. Then he chooses a real number $0 \leqslant \varepsilon < 1$ – an acceptable deviation from the perfectly balanced distribution. Distributor responds by carefully picking a number $k > 0$. This number determines how many goods each of the participants will receive or, equivalently, how many times each participant will occur in the subsets N_i constructed by Nominator.

After the preparatory phase the game is played in rounds. In the i-th round Nominator nominates a nonempty subset $N_i \subseteq \Pi$. Distributor responds by choosing any $\delta_i \in \Gamma$ and by giving one good of the commodity δ_i to each participant $\pi \in N_i$. The following two parameters are necessary to determine the winner in the BD-game.

- $\operatorname{occ}_i(\pi) = |\{j : \pi \in N_j \text{ and } j \leqslant i\}|$ – the number of goods given to the participant $\pi \in \Pi$ in the first i rounds,
- $\operatorname{gds}_i(\pi, \gamma) = |\{j : \pi \in N_j \text{ and } \gamma = \delta_j \text{ and } j \leqslant i\}|$ – the number of goods of the commodity $\gamma \in \Gamma$ given to the participant $\pi \in \Pi$ in the first i rounds.

We will omit the subscripts and use the parameters $\operatorname{occ}(\cdot)$ and $\operatorname{gds}(\cdot, \cdot)$ which change their values as the game is played.

Nominator is not allowed to construct sequences with $\operatorname{occ}(\pi) > k$. He instantly loses the game if he decides to break this rule. The game actually ends when each of the participants from Π was nominated exactly k times, i.e. $\operatorname{occ}(\pi) = k$ for all $\pi \in \Pi$. As Nominator presents only nonempty subsets, the game comes to an end after at most $k \cdot |\Pi|$ rounds. Distributor wins if the inequality

$$\operatorname{gds}(\pi, \gamma) \geqslant (1 - \varepsilon) \frac{k}{|\Gamma|}$$

holds for each π and γ. Otherwise Nominator wins.

We will also consider a variant of the BD-game in which values of Π, Γ and ε are fixed instead of being chosen by Nominator. We will see that the most important parameter is ε and the situation splits dramatically between $\varepsilon > 0$ and $\varepsilon = 0$.

One natural strategy for Distributor is to choose k "big enough" and then pick each δ_i "uniformly at random from Γ." We omit the somewhat troublesome details, as they will not be needed, and assure the reader that for $\varepsilon > 0$ this randomised strategy brings victory to Distributor with probability tending to 1 as k tends to infinity. The following lemma gives a straightforward derandomisation of this idea.

Lemma 4.3. In a BD-game with $\varepsilon > 0$, Distributor has a winning strategy.

Proof. We describe a strategy that leads to the defeat of Nominator. In the preparatory phase Distributor, knowing Π, Γ and $\varepsilon > 0$, fixes an arbitrary numbering of $\Gamma = \{\gamma_0, \ldots, \gamma_{r-1}\}$ and sets

$$k = \left\lceil \frac{2^{|\Pi|}\,|\Gamma|}{\varepsilon} \right\rceil.$$

In the i-th round, Distributor needs to select δ_i for the participants in N_i nominated by Nominator. If this is the first time the subset N_i appears in the sequence $(N_1, \ldots, N_i)$ then Distributor simply chooses $\delta_i = \gamma_0$. Otherwise, let $j < i$ be the biggest number such that $N_j = N_i$. Distributor finds q such that $\delta_j = \gamma_q$ and chooses δ_i to be the next one in Γ, i.e. $\delta_i = \gamma_{((q+1) \bmod r)}$.

Suppose the game ends after round ℓ. Consider the possible values of $\operatorname{gds}_\ell(\pi, \gamma)$ at the end of the game. For a fixed participant π let γ_{min}, γ_{max} be such that $\operatorname{gds}_\ell(\pi, \gamma_{min}) = \min_{\gamma \in \Gamma} \operatorname{gds}_\ell(\pi, \gamma)$ and $\operatorname{gds}_\ell(\pi, \gamma_{max}) = \max_{\gamma \in \Gamma} \operatorname{gds}_\ell(\pi, \gamma)$.

Now, suppose that a subset $N \subseteq \Pi$ occurs s times in the sequence $(N_1, \ldots, N_\ell)$. Observe that Distributor responded to those s occurrences of the subset N in a round robin manner, and γ_j was chosen to be the response exactly $\left\lceil \frac{s-j}{r} \right\rceil$ times. This means that the subset N contributes at most 1 to the value of the difference $\mathrm{gds}_\ell(\pi, \gamma_{max}) - \mathrm{gds}_\ell(\pi, \gamma_{min})$. This, together with the fact that $\mathrm{gds}_\ell(\pi, \gamma_{max})$ is not smaller than the average value $\frac{k}{|\Gamma|}$ of $\mathrm{gds}_\ell(\pi, \gamma)$, allows us to state

$$\mathrm{gds}_\ell(\pi, \gamma_{min}) \geqslant \mathrm{gds}_\ell(\pi, \gamma_{max}) - 2^{|\Pi|} \geqslant \frac{k}{|\Gamma|} - 2^{|\Pi|} \geqslant \frac{k}{|\Gamma|}(1 - \varepsilon).$$

Since this holds for all participants $\pi \in \Pi$, Distributor has won. □

From the results of **[ATV97]** one can infer that in the off-line version of Balanced Distribution Game (i.e. when all sets N_i are given to Distributor in a single batch) Distributor has a winning strategy even for $\varepsilon = 0$. In the BD-games with $|\Pi| = 1$ or $|\Gamma| = 1$ Distributor has an obvious winning strategy. In the BD-games with $\varepsilon = 0$ and $|\Pi| = 2$, a winning strategy for Distributor also exists:

- set $k = |\Gamma|$ (meaning that each of the two participants should receive one good of each commodity),
- before the i-th round renumber Π to $\{\pi_1, \pi_2\} = \Pi$ so that $\mathrm{occ}(\pi_1) \geqslant \mathrm{occ}(\pi_2)$,
- if $\pi_1 \in N_i$, set δ_i to be one of the goods not received yet by π_1,
- if $\pi_1 \notin N_i$, set δ_i to be one of the goods received by π_1 but not by π_2.

The next lemma shows that the situation changes dramatically in the remaining cases.

Lemma 4.4. In a BD-game with fixed $\varepsilon = 0$, $|\Pi| \geqslant 3$ and $|\Gamma| \geqslant 2$, Nominator has a winning strategy.

Proof. Assume that ε, Π and Γ are given as in the Lemma and that Distributor has already set k. If $k = 1$ then Nominator plays $N_1 = \Pi$ and wins instantly. We will assume $k \geqslant 2$ for the rest of the proof.

We call a configuration of the two participants (π_1, π_2) insecure if $\mathrm{occ}(\pi_1) = \mathrm{occ}(\pi_2)$ and $\mathrm{gds}(\pi_1, \gamma_1) \neq \mathrm{gds}(\pi_2, \gamma_1)$ for some γ_1. Observe that if at any point of the game Nominator finds an insecure configuration (π_1, π_2) then he can win the game using the following approach. Nominator simply plays $k - \mathrm{occ}(\pi_1)$ times the subset $\{\pi_1, \pi_2\}$. No matter how Distributor responds to those moves, $\mathrm{occ}(\pi_1) = \mathrm{occ}(\pi_2) = k$ and

$\operatorname{gds}(\pi_1, \gamma_1) \neq \operatorname{gds}(\pi_2, \gamma_1)$ still holds. Without the loss of generality this allows us to assume that $\operatorname{gds}(\pi_1, \gamma_1) \neq \frac{k}{|\Gamma|}$. This means that the goods given to π_1 are not in a balance, and therefore $\operatorname{gds}(\pi_1, \gamma) < \frac{k}{|\Gamma|}$ for some $\gamma \in \Gamma$. Nominator still needs to finish the game and he will do so by playing singletons $N_i = \{\pi\}$ as long as there exists a π with $\operatorname{occ}(\pi) < k$. Distributor loses the game, for he failed to balance the distribution of the goods received by π_1.

Now we show how Nominator can win the game. He starts with fixing three different participants $\pi_1, \pi_2, \pi_3 \in \Pi$. In the first round Nominator plays $N_1 = \{\pi_1, \pi_2\}$ and after Distributor's response δ_1 he plays $N_2 = \{\pi_1, \pi_3\}$. Distributor must respond with $\delta_2 = \delta_1$, for otherwise the configuration (π_2, π_3) becomes insecure. Nominator continues with $N_3 = \{\pi_2\}$ forcing $\delta_3 = \delta_1$. Indeed, for $\delta_3 \neq \delta_2 = \delta_1$ the configuration (π_1, π_2) is insecure. The following table presents the state of the game during the first rounds:

i	N_i	δ_i	$\operatorname{occ}(\pi_1)$	$\operatorname{occ}(\pi_2)$	$\operatorname{occ}(\pi_3)$
1	$\{\pi_1, \pi_2\}$	δ_1	1	1	0
2	$\{\pi_1, \pi_3\}$	δ_1	2	1	1
3	$\{\pi_2\}$	δ_1	2	2	1

Nominator may continue alternating the moves $\{\pi_1, \pi_3\}$ and $\{\pi_2\}$ and Distributor must always respond with the same δ_1 or create an insecure configuration. When Nominator can no longer continue the simple alternations, the situation is as follows:

i	N_i	δ_i	$\operatorname{occ}(\pi_1)$	$\operatorname{occ}(\pi_2)$	$\operatorname{occ}(\pi_3)$
$2k-2$	$\{\pi_1, \pi_3\}$	δ_1	k	$k-1$	$k-1$
$2k-1$	$\{\pi_2\}$	δ_1	k	k	$k-1$

Again Nominator can finish the game by playing singletons as long as needed. He wins, for the distributions of the goods owned by the participants π_1 and π_2 are as far from being balanced as one can get – all of the goods received by them are of the single commodity δ_1. This is unacceptable for $|\Gamma| \geqslant 2$. □

4.2. On-line choice ratio

Now that we understand the basic mechanics of the BD-game, we will use this knowledge to prove the main results concerning the on-line choice ratio $\mathrm{ch}_{\mathrm{F}}^{\mathrm{OL}}(G)$.

Theorem 4.5. For a graph G we have $\mathrm{ch}_{\mathrm{F}}^{\mathrm{OL}}(G) = \chi_{\mathrm{F}}(G)$.

Proof. Recall that inequality $\mathrm{ch}_{\mathrm{F}}^{\mathrm{OL}}(G) \geqslant \chi_{\mathrm{F}}(G)$ follows immediately from the definitions of those parameters. We are going to prove the converse inequality by reducing a PC-game to a BD-game. For a graph $G = (V, E)$, let L_a be the list assignment with $L_a(v) = \{0, \ldots, a-1\}$ for all $v \in V$. Lemma 4.2 supplies us with a, b such that $\chi_{\mathrm{F}}(G) = \frac{a}{b}$ and an (L_a, b)-colouring Φ. Given any real number $\tau > 0$ we will find a' and b' such that G is on-line (a', b')-choosable and $\frac{a'}{b'} \leqslant \frac{a}{b}(1+\tau)$. This will allow us to conclude that $\mathrm{ch}_{\mathrm{F}}^{\mathrm{OL}}(G) = \frac{a}{b}$.

To find a' and b' and construct a strategy for Mrs. Correct, we are going to use Distributor's winning strategy in a Balanced Distribution Game and the colouring Φ. We imagine that Mrs. Correct plays simultaneously as Nominator in the BD-game (with $\Pi = V$, $\Gamma = \{0, \ldots, a-1\}$ and $\varepsilon = \frac{\tau}{1+\tau}$) against Distributor using a winning strategy. She translates P_i played by Mr. Paint to $N_i := P_i$ for the BD-game, and then takes Distributor's response δ_i to calculate her response in the PC-game. The idea is that, at the end of the game, for each vertex v, Distributor partitioned the occurrences of v in the sets P_i into a different types of about equal size. For each occurrence of v in P_i, Mrs. Correct is going to include v in her response C_i if the type chosen by Distributor coincides with one of the colours assigned to v in the colouring Φ. Mrs. Correct's strategy grants that a fraction of about $\frac{a}{b}$ of the occurrences will be included in her responses. In detail, we proceed as follows.

By Lemma 4.3 we know that Distributor has a winning strategy in the BD-game with $\Pi = V$, $\Gamma = \{0, \ldots, a-1\}$ and $\varepsilon = \frac{\tau}{1+\tau}$. In particular this strategy tells Distributor a value of k with which to respond initially. Now we are ready to set $a' = k$ and $b' = \left\lceil (1-\varepsilon)\frac{b}{a}k \right\rceil$ and start the PC-game. Note that

$$\frac{a'}{b'} \leqslant \frac{a}{b}\frac{1}{1-\varepsilon} = \frac{a}{b}(1+\tau).$$

In the i-th round, Mr. Paint presents a set P_i. In order to find her response $C_i \subseteq P_i$, Mrs. Correct plays $N_i = P_i$ as Nominator in the BD-game. Distributor responds with $\delta_i \in \{0, \ldots, a-1\}$. Mrs. Correct includes a vertex v into her response $C_i \subseteq P_i$ if and only if δ_i is one of the colours assigned to v in the colouring Φ, i.e.

$$C_i = P_i \cap \{v \in V : \delta_i \in \Phi(v)\}.$$

This is a valid response in the PC-game, as $\{v \in V : \delta_i \in \Phi(v)\}$ is an independent set.

The PC-game ends when each list contains $a' = k$ colours. This means that every participant was nominated exactly k times and the BD-game finishes in the very same moment. Distributor wins the BD-game, thus $\operatorname{gds}(v, d) \geqslant (1-\varepsilon)\frac{k}{a}$ for each vertex v and $d \in \{0, \ldots, a-1\}$. Since $|\Phi(v)| = b$ we know that each vertex was given a set $\varphi(v)$ of at least b' colours in the colouring constructed by Mrs. Correct, for

$$|\varphi(v)| = \sum_{d \in \Phi(v)} \operatorname{gds}(v, d) \geqslant \left\lceil b(1-\varepsilon)\frac{k}{a} \right\rceil = b'.$$

□

The next theorem shows that, in contrast to Lemma 4.2, the infimum in the definition of the on-line choice ratio is not always reached.

Theorem 4.6. There is an infinite family $\mathcal{G}$ of graphs, such that for every graph $G \in \mathcal{G}$ we have

$$\operatorname{ch}_{\mathrm{F}}^{\mathrm{OL}}(G) \notin \left\{ \frac{a}{b} : G \text{ is on-line } (a,b)\text{-choosable} \right\}.$$

Proof. For the finite sets Π, Γ with $|\Pi| \geqslant 3$ and $|\Gamma| \geqslant 2$ let the graph $G_{\Pi,\Gamma} = (V, E)$ be a complete $|\Gamma|$-partite graph with $|\Pi|$ vertices in each partite set. For the further reference we simply put:

- $V = \Pi \times \Gamma$,
- $E = \{\{(\pi_1, \gamma_1), (\pi_2, \gamma_2)\} \subseteq V : \gamma_1 \neq \gamma_2\}$.

The graph $G_{\Pi,\Gamma}$ is obviously $(|\Gamma|, 1)$-colourable – one can assign a different colour to each partite set $\Pi \times \{\gamma\}$. Observe that the graph $G_{\Pi,\Gamma}$ contains $|\Gamma|$-cliques, for example $\{\pi\} \times \Gamma$ for any $\pi \in \Pi$. Assume that $G_{\Pi,\Gamma}$ is (a,b)-colourable. Colours assigned to the vertices of any $|\Gamma|$-clique need to be different, thus $a \geqslant b \cdot |\Gamma|$ holds. It follows that $\chi_{\mathrm{F}}(G_{\Pi,\Gamma}) = \chi(G_{\Pi,\Gamma}) = |\Gamma|$. Theorem 4.5 allows us to conclude that $\operatorname{ch}_{\mathrm{F}}^{\mathrm{OL}}(G_{\Pi,\Gamma}) = |\Gamma|$.

Put $\mathcal{G} = \{G_{\Pi,\Gamma} : |\Pi| \geqslant 3 \wedge |\Gamma| \geqslant 2\}$. Now suppose, to the contrary, that some $G_{\Pi,\Gamma} \in \mathcal{G}$ is on-line (a, b)-choosable for some a, b with $\frac{a}{b} = |\Gamma|$. This means that Mrs. Correct has a winning strategy $\mathcal{S}$ in the PC-game on the board $(G_{\Pi,\Gamma}, a, b)$.

Leading to a contradiction with Lemma 4.4 we will find a winning strategy for Distributor in the BD-game with Π, Γ and $\varepsilon = 0$ using a reduction to the PC-game on the graph $G_{\Pi,\Gamma}$ so that strategy $\mathcal{S}$ can be used.

Distributor's strategy is as follows. In the preparatory phase of the BD-game Distributor responds with $k = a$. In the i-th round, when Nominator nominates $N_i \subseteq \Pi$, Distributor plays $P_i = N_i \times \Gamma$ in the PC-game as Mr. Paint. Strategy $\mathcal{S}$ tells Mrs. Correct to respond with an independent set $C_i \subseteq P_i$. The set C_i contains vertices from at most one copy $\Pi \times \{\gamma\}$ of Π. Distributor sets his response δ_i in the BD-game to be γ such that $C_i \subseteq N_i \times \{\gamma\}$. Observe that the participant π receives one good of the commodity γ each time the vertex (π, γ) is present in the set C_i.

The BD-game ends when each participant was nominated a times. This means that each vertex in G has a list of length a attached to it. The PC-game is over and Mrs. Correct has won using the strategy $\mathcal{S}$, so each vertex is present in at least b of the sets C_i. This means that $\operatorname{gds}(\pi, \gamma) \geqslant b = \frac{k}{|\Gamma|}$, and that Distributor has managed to construct a perfectly balanced distribution. This contradiction with Lemma 4.4 ends the proof. □

5

Conclusions and open problems

In Chapter 3 we show that from the computational complexity point of view the notion of on-line choosability is a difficult one. This result is not a surprise as intuitively the on-line choosability should be more difficult than the standard list colouring. We hope that our techniques can be developed further and our lower bound for the computational complexity of OnLineChoosability can be improved.

In Chapter 4 we show that the fractional versions of the on-line choosability and the off-line choosability are similar in nature. The subtle difference between these two notions lies in the realisation of the limits used for their definition. Perhaps, the strong connection between "Mr. Paint and Mrs. Correct" and Balanced Distribution Game can be exploited further to show some other results.

In his paper Zhu [**Zhu09**] gives a list of 11 interesting questions in the field of on-line choosability (this thesis gives an answer to just one of them). We conclude our thesis with some problems we would like to see solved.

Complexity. Following the work presented in Chapter 3 we conjecture the following:

Conjecture 5.1. OnLineChoosability is PSPACE-complete.

We have developed some tools to prove this conjecture, but some pieces are still missing. The main difficulty is that we lack any gadgets which would force Mr. Paint to play in different parts of the graph in some prescribed order.

On-line (a, b)-choosability.

Question 5.2. Is it true that every on-line a-choosable graph is on-line (ak, k)-choosable for all $k \in \mathbb{N}$?

This is an on-line variant of the famous conjecture by Erdős, Rubin and Taylor **[ERT80]** about (ak, k)-choosability. Perhaps, one can answer Question 5.2 negatively.

Competitivity. Using the fact that the choice number grows with the colouring number **[Alo93]**, we know that there exists a very fast growing function f such that for each graph G we have:

$$\mathrm{ch}^{\mathrm{OL}}(G) \leqslant \mathrm{col}(G) \leqslant f(\mathrm{ch}(G)).$$

Question 5.3. Is there a slow growing (polynomial, linear) function f such that for each graph G we have:

$$\mathrm{ch}^{\mathrm{OL}}(G) \leqslant f(\mathrm{ch}(G))\,?$$

To support a possible positive answer to Question 5.3 note that there is no known example of a graph G with $\mathrm{ch}^{\mathrm{OL}}(G) > \mathrm{ch}(G) + 1$. However, we conjecture that the difference between the on-line choice number and the off-line choice number can be arbitrarily large.

Conjecture 5.4. For each constant $c \in \mathbb{N}$ there exists a graph G such that:

$$\mathrm{ch}^{\mathrm{OL}}(G) > \mathrm{ch}(G) + c.$$

On-line choice index. The famous list edge colouring conjecture states that for any graph G the list chromatic index $\mathrm{ch}'(G)$ equals the chromatic index $\chi'(G)$. This motivates our next question.

Question 5.5. Is it true that for each graph G we have:

$$\mathrm{ch}^{\mathrm{OL}'}(G) = \mathrm{ch}'(G) = \chi'(G)\,?$$

The famous Vizing's Theorem **[Viz64]** gives $\chi'(G) \leqslant \Delta(G) + 1$. Perhaps, one can answer negatively the previous question by answering positively the following:

Question 5.6. Is there a graph G with:

$$\mathrm{ch}^{\mathrm{OL}'}(G) \geqslant \Delta(G) + 2?$$

Bipartite graphs. Johansson [**Joh96**] proved, using random methods, that for a triangle-free graph G with $\Delta(G) = \Delta$ we have an $O\left(\frac{\Delta}{\log \Delta}\right)$ upper bound for $\mathrm{ch}(G)$.

Question 5.7. What is the maximum $\mathrm{ch}^{\mathrm{OL}}(G)$ for a bipartite graph G with bounded $\Delta(G)$?

The random method used by Johansson [**Joh96**] (see [**MR01**, Chapter 13]) seems inapplicable in the on-line setting.

Products. Lemma 2.3 gives an upper bound for $\mathrm{ch}^{\mathrm{OL}}(G \times H)$ in terms of $\mathrm{ch}^{\mathrm{OL}}(G)$ and $\mathrm{col}(H)$.

Question 5.8. For any graphs G and H, is $\mathrm{ch}^{\mathrm{OL}}(G \times H)$ bounded by $\mathrm{ch}^{\mathrm{OL}}(G) + \mathrm{ch}^{\mathrm{OL}}(H)$?

Bibliography

[Alo93] Noga Alon. Restricted colorings of graphs. In *Proceedings, 14th British Combinatorial Conference*, volume 187 of *Surveys in Combinatorics, London Mathematical Society Lecture Notes Series*, pages 1–33, 1993.

[ATV97] Noga Alon, Zsolt Tuza, and Margit Voigt. Choosability and fractional chromatic numbers. *Discrete Mathematics*, 165–166:31–38, 1997.

[BJKM06] Mieczysław Borowiecki, Stanislav Jendrol', Daniel Král', and Jozef Miškuf. List coloring of cartesian products of graphs. *Discrete Mathematics*, 306(16):1955–1958, 2006.

[Die10] Reinhard Diestel. *Graph Theory*. Springer, fourth edition, 2010.

[ERT80] Paul Erdős, Arthur L. Rubin, and Herbert Taylor. Choosability in graphs. In *Proceedings, West Coast Conference on Combinatorics, Graph Theory and Computing*, volume 26 of *Congressus Numerantium*, pages 125–157, 1980.

[Gal95] Fred Galvin. The list chromatic index of a bipartite multigraph. *Journal of Combinatorial Theory, Series B*, 63(1):153–158, 1995.

[GT09] Shai Gutner and Michael Tarsi. Some results on $(a{:}b)$-choosability. *Discrete Mathematics*, 309(8):2260–2270, 2009.

[Gut92] Shai Gutner. Choice numbers of graphs. Master's thesis, Tel-Aviv University, 1992.

[Gut96] Shai Gutner. The complexity of planar graph choosability. *Discrete Mathematics*, 159(1–3):119–130, 1996.

[Gut11] Grzegorz Gutowski. Mr. Paint and Mrs. Correct go fractional. *Electronic Journal of Combinatorics*, 18(1):Research Paper 140, 8 pp. (electronic), 2011.

[HKS10] Jan Hladký, Daniel Král', and Uwe Schauz. Brooks' Theorem via the Alon-Tarsi Theorem. *Discrete Mathematics*, 310(23):3426–3428, 2010.

[Joh96] Anders Johansson. Asymptotic choice number for triangle free graphs. Technical report, DIMACS, 1996.

[MR01] Michael Molloy and Bruce Reed. *Graph Colouring and the Probabilistic Method*, volume 23 of *Algorithms and Combinatorics*. Springer, 2001.

[MRS91] Nadimpalli V. R. Mahadev, Fred S. Roberts, and Prakash Santhanakrishnan. 3-choosable complete bipartite graphs. Technical Report 91-62, DIMACS, 1991.

[O'D95] Paul O'Donnell. The choice number of $K_{6,q}$. Preprint, 1995.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[Sch09] Uwe Schauz. Mr. Paint and Mrs. Correct. *Electronic Journal of Combinatorics*, 16(1):Research Paper 77, 18 pp. (electronic), 2009.

[Sch10a] Uwe Schauz. Flexible color lists in Alon and Tarsi's Theorem, and time scheduling with unreliable participants. *Electronic Journal of Combinatorics*, 17:Research Paper 13, 18 pp. (electronic), 2010.

[Sch10b] Uwe Schauz. A paintability version of the combinatorial nullstellensatz, and list colorings of k-partite k-uniform hypergraphs. *Electronic Journal of Combinatorics*, 17:Research Paper 176, 13 pp. (electronic), 2010.

[Sip06] Michael Sipser. *Introduction to the Theory of Computation*. Thompson, second edition, 2006.

[ST95] Anil M. Shende and Barry A. Tesman. 3-choosability of $K_{5,q}$. In *Proceedings, 26th Southeastern International Conference on Combinatorics, Graph Theory and Computing*, volume 111 of *Congressus Numerantium*, pages 193–221, 1995.

[Tho94] Carsten Thomassen. Every planar graph is 5-choosable. *Journal of Combinatorial Theory, Series B*, 62(1):180–181, 1994.

[Tov84] Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–90, 1984.

[Viz64] Vadim G. Vizing. On an estimate of the chromatic class of a p-graph, «Об оценке хроматического класса p-графа». *Дискретный анализ*, 3:25–30, 1964. in Russian.

[Viz76] Vadim G. Vizing. Colouring the vertices of a graph in prescribed colours, «Раскраска вершин графа в предписанные цвета». In *Методы дискретного анализа в теории кодов и схем*, volume 29 of *Дискретный анализ*, pages 3–10. 1976. in Russian.

[Zhu09] Xuding Zhu. On-line list colouring of graphs. *Electronic Journal of Combinatorics*, 16(1):Research Paper 127, 16 pp. (electronic), 2009.

APPENDIX A

Computer assisted proofs

In this appendix we present a source code of a program which is used to prove the properties of the gadgets from Section 3.4. In Sections A.1, A.2 and A.3 we present input files, which allow to execute proofs of Facts 3.10, 3.13 and 3.15, respectively. All of the source code and the input files presented in this appendix are available on a CD attached to this thesis.

We start with a description of the program. It takes on input a specification of a graph and a number cases to check. Each case describes a $\mathrm{PC}^{\mathrm{pm}}$-game board and a player (Mr. Paint or Mrs. Correct) which is expected to have a winning strategy on this board. The program checks that in each case the specified player has a winning strategy in $\mathrm{PC}^{\mathrm{pm}}$-game on the specified board. If all checks are successful then the program prints `OK` and if any of the checks fails then the program prints `FAIL`.

Most of the proofs we need to execute deal with PC-games, not $\mathrm{PC}^{\mathrm{pm}}$-games. We handle PC-games as $\mathrm{PC}^{\mathrm{pm}}$-games with empty sets of vulnerable and solid vertices.

The program is an effective implementation of Algorithm 3.5. It performs an exhaustive discovery of all possible gameplays and marks winning positions for Mrs. Correct – all other positions are winning for Mr. Paint. There are two important optimisations which speed up the execution of the proofs. First is the memoization of the winning player in each position. Once the winning player is found for a certain position, it is stored in the memory. Next time the program needs to know the winner, it simply checks it in the memory.

Second important optimisation is limiting the number of Mrs. Correct moves taken into consideration at each position. If Mr. Paint plays a set P then the program checks only responses $C \subseteq P$ which are the maximal independent subsets of P. This optimisation considerably limits the number of analysed responses. The problem is, that it may possibly lead to wrong results. It is obvious that in a PC-game there is no need

to analyse any responses other than the maximal independent subsets – responding with a bigger set is always beneficial for Mrs. Correct. For $\mathrm{PC}^{\mathrm{pm}}$-games this optimisation may possibly (although we are not aware of any example) lead to wrongly calculating that Mr. Paint is a winner in some situations. For our purposes it is not a problem, as we need to analyse only one $\mathrm{PC}^{\mathrm{pm}}$-game for proof of Fact 3.15 and in that proof we need to certify that Mrs. Correct is a winner. If our program does that, then Mrs. Correct must be a true winner.

Before proceeding further we give a strict description of the input files used by the program. In the first line of the input there are two numbers n, the number of vertices in the graph, and m, the number of edges. The vertices of the graph are labelled 0 to $n-1$. In the next m lines there are pairs of numbers $0 \leqslant a, b \leqslant n-1$ describing the edge $\{a, b\}$. It is followed by a line with single number c of the cases to check. The data for each case begins with a line containing the number v of vulnerable vertices followed by v labels of vulnerable vertices. In the second line of the data for each case there is the number s of solid vertices followed by s labels of solid vertices. The last line of data for each case contains n numbers – values of the list-length-function on the vertices 0 to $n-1$, and the name of a player (`Paint` or `Correct`) which is expected to have a winning strategy.

Before running the proofs, one needs to compile the provided C++ source code in the file `gadget_check.cpp`. To perform a specific proof, one needs to run the compiled program and give one of the files:

- `gadget_crit.in` for G_{crit},
- `gadget_cons.in` for G_{cons},
- `gadget_choice.in` for G_{choice},

on the standard input. Once all the checks are done, the program will produce `OK` on the standard output. There is a makefile provided on the CD that compiles the program and executes all the proofs.

Now we present the C++ source code of our program. Some implementation notes can be found in the comments inside the source code.

```
#include <iostream>
#include <set>
#include <vector>
using namespace std;
typedef unsigned int uint;

class SmallSet {
  // small set stored as a bit string
  private:
  uint set;

  static inline uint bit (uint i) {
    return ((uint)1 << i);
  }

  SmallSet (uint _set) :
    set(_set) {
  }

  public:
  SmallSet () :
    // construct empty set
    set(0) {
  }

  static SmallSet one_element (uint i) {
    // construct a set i
    return SmallSet(bit(i));
  }
  static SmallSet all_elements (uint i) {
    // construct a set 0,...,i-1
    return SmallSet(bit(i)-1);
  }

  bool operator== (const SmallSet& o) const {
    return set == o.set;
  }
  bool operator!= (const SmallSet& o) const {
    return set != o.set;
  }

  bool empty() const {
    return set == 0;
  }

  // check, add, remove element
  bool operator[] (uint i) const {
    return (set & bit(i)) != 0;
  }
```

```
SmallSet operator+ (uint i) const {
  return SmallSet(set | bit(i));
}
SmallSet operator- (uint i) const {
  return SmallSet(set & ~bit(i));
}
SmallSet& operator+= (uint i) {
  set |= bit(i);
  return *this;
}
SmallSet& operator-= (uint i) {
  set &= ~bit(i);
  return *this;
}

// union, difference and intersection of sets
SmallSet operator+ (const SmallSet& o) const {
  return SmallSet(set | o.set);
}
SmallSet operator- (const SmallSet& o) const {
  return SmallSet(set & ~o.set);
}
SmallSet operator* (const SmallSet& o) const {
  return SmallSet(set & o.set);
}
SmallSet& operator+= (const SmallSet& o) {
  set |= o.set;
  return *this;
}
SmallSet& operator-= (const SmallSet& o) {
  set &= ~o.set;
  return *this;
}
SmallSet& operator*= (const SmallSet& o) {
  set &= o.set;
  return *this;
}

// inclusion order operators
bool operator<= (const SmallSet& o) const {
  return (set & ~o.set) == 0;
}
bool operator< (const SmallSet& o) const {
  return *this <= o && *this != o;
}
bool operator>= (const SmallSet& o) const {
  return (o.set & ~set) == 0;
}
bool operator> (const SmallSet& o) const {
```

```
    return *this >= o && *this != o;
  }

  vector<SmallSet> subsets () const {
    // return all subsets of the set
    vector<SmallSet> subs;
    for(uint i=0;i<=set;i++)
      if((i|set) == set)
        subs.push_back(SmallSet(i));
    return subs;
  }

  struct Compare {
    // linear order on sets
    bool operator() (const SmallSet& a, const SmallSet& b) const {
      return a.set < b.set;
    }
  };
};

class GameBoard {
  // description of the graph, vulnerable set and solid set
  public:
  vector<SmallSet> graph;
  SmallSet vulnerable;
  SmallSet solid;
  set<SmallSet, SmallSet::Compare> maximal_independent_sets;

  void misSearch (SmallSet is) {
    // calculate and store all maximal independent sets
    SmallSet neighbourhood(is);
    for(uint i=0;i<graph.size();i++)
      if(is[i])
        neighbourhood += graph[i];
    if(neighbourhood == SmallSet::all_elements(graph.size()))
      maximal_independent_sets.insert(is);
    else
      for(uint i=0;i<graph.size();i++)
        if(!neighbourhood[i])
          misSearch(is + i);
  }

  GameBoard (uint size, const vector<pair<uint, uint> >& edges,
      const vector<uint>& vul, const vector<uint>& sol) :
    graph(size) {
    // construct board based on list of edges, vulnerable/solid vertices
    for(uint i=0;i<edges.size();i++)
    {
      graph[edges[i].first] += edges[i].second;
```

```
      graph[edges[i].second] += edges[i].first;
    }
    for(uint i=0;i<vul.size();i++)
      vulnerable += vul[i];
    for(uint i=0;i<sol.size();i++)
      solid += sol[i];
    misSearch(SmallSet());
  }
};

class GameState {
  // description of the state of the game
  public:
  const GameBoard& board;
  vector<uint> paint_count;
  SmallSet vulnerable;

  GameState (const GameBoard& _board) :
    board(_board),
    paint_count(_board.graph.size(), 0),
    vulnerable(_board.vulnerable) {
  }

  // bit coding
  inline static void push (uint& cd, uint c, uint b) {
    cd <<= b;
    cd += c;
  }
  inline static void pop (uint& cd, uint& c, uint b) {
    c = cd & (((uint)1 << b)-1);
    cd >>= b;
  }

  // encode and decode in uint
  uint code () const {
    uint cd = 0;
    for(uint i=0;i<board.graph.size();i++)
      if(board.vulnerable[i])
        push(cd, vulnerable[i]?1:0, 1);
    for(uint i=0;i<board.graph.size();i++) {
      push(cd, paint_count[i], 2);
    }
    return cd;
  }
  void decode (uint cd) {
    for(uint i=1;i<=board.graph.size();i++) {
      uint b;
      pop(cd, b, 2);
      paint_count[board.graph.size()-i] = b;
```

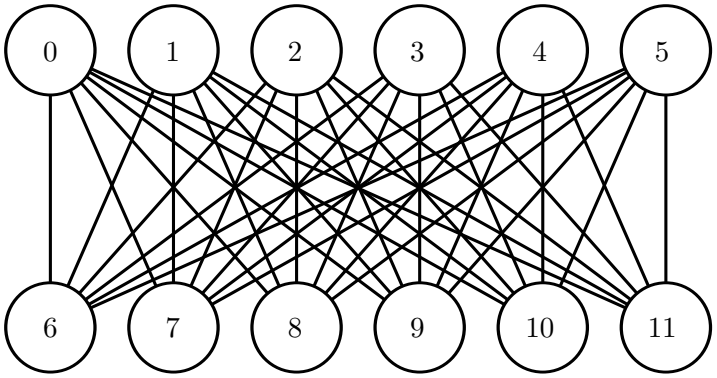
```
  }
  vulnerable = SmallSet();
  for(uint i=1;i<=board.graph.size();i++)
    if(board.vulnerable[board.graph.size()-i]) {
      uint b;
      pop(cd, b, 1);
      if(b!=0)
        vulnerable += board.graph.size()-i;
    }
}

vector<SmallSet> paintMoves () const {
  // enumerate all Mr.Paint moves
  SmallSet active;
  for(uint i=0;i<board.graph.size();i++)
    if(paint_count[i] < 3)
      active += i;
  vector<SmallSet> subs = active.subsets();
  for(uint i=0;i<subs.size();i++)
    if(subs[i].empty()) {
      swap(subs[i],subs.back());
      subs.pop_back();
    }
  return subs;
}

vector<SmallSet> correctResponses (SmallSet move) const {
  // enumerate maximal Mrs.Correct responses
  set<SmallSet, SmallSet::Compare> resp;
  for(set<SmallSet, SmallSet::Compare>::const_iterator
      i=board.maximal_independent_sets.begin();
      i!=board.maximal_independent_sets.end();i++)
    resp.insert(*i * move);
  vector<SmallSet> res;
  for(set<SmallSet, SmallSet::Compare>::const_iterator
      i=resp.begin();
      i!=resp.end();i++)
    if(!i->empty())
      res.push_back(*i);
  return res;
}

vector<SmallSet> powerMoves (SmallSet move, SmallSet response) const {
  // enumerate power moves for Mr.Paint
  SmallSet active;
  for(uint i=0;i<board.graph.size();i++)
    if(vulnerable[i] && response[i])
      active += i;
  vector<SmallSet> subs = active.subsets();
```

```
    for(uint i=0;i<subs.size();i++)
      if(subs[i].empty()) {
        swap(subs[i],subs.back());
        subs.pop_back();
      }
    return subs;
  }

  bool oneRound (SmallSet move, SmallSet response, GameState& state) const {
    // calculate state after move and response
    for(uint i=0;i<board.graph.size();i++)
      if(move[i] && !response[i] && paint_count[i] == 2)
        return false;
    state.vulnerable = vulnerable;
    for(uint i=0;i<board.graph.size();i++)
      if(response[i] || paint_count[i] == 3)
        state.paint_count[i] = 3;
      else if(move[i])
        state.paint_count[i] = paint_count[i] + 1;
      else
        state.paint_count[i] = paint_count[i];
    return true;
  }
};

class GameSolution {
  // calculation and memoization of winner
  public:
  const GameBoard& board;
  vector<bool> _solved;
  vector<bool> _solution;

  GameSolution(const GameBoard& _board) :
    board(_board) {
  }

  bool solution (uint i);
  bool solution (const GameState& state) {
    return solution(state.code());
  }

  bool solve (const GameState& state) {
    // establish winner
    vector<SmallSet> moves = state.paintMoves();
    for(uint i=0;i<moves.size();i++) {
      // iterate over Mr.Paint moves
      SmallSet move = moves[i];
      bool danger=true;
      vector<SmallSet> responses = state.correctResponses(move);
```

```
      for(uint j=0;j<responses.size();j++) {
        // iterate over Mrs.Correct responses
        SmallSet response = responses[j];
        GameState result(board);
        if(!state.oneRound(move, response, result))
          continue;
        if(!solution(result))
          continue;
        danger = false;
        vector<SmallSet> power_moves = state.powerMoves(move, response);
        for(uint k=0;k<power_moves.size();k++) {
          // iterate over Mr.Paint power moves
          SmallSet power = power_moves[k];
          SmallSet forbidden = board.solid - response;
          SmallSet obligatory = board.solid * response;
          danger = true;
          for(uint l=0;l<responses.size();l++) {
            // iterate over Mrs.Correct responses
            if(! (obligatory <= responses[l]))
              continue;
            SmallSet changed = responses[l] - power - forbidden;
            GameState result2(board);
            if(!state.oneRound(move, changed, result2))
              continue;
            result2.vulnerable -= power;
            if(!solution(result2))
              continue;
            danger = false;
          }
          if(danger)
            break;
        }
        if(!danger)
          break;
      }
      if(danger)
        return false;
    }
    return true;
  }
};

bool GameSolution::solution (uint i) {
  if(_solved.size() <= i || !_solved[i]) {
    if(_solved.size() <= i) {
      _solved.resize(max(i,(uint)_solved.size()*2),false);
      _solution.resize(_solved.size());
    }
    GameState state(board);
```

```
    state.decode(i);
    bool res = solve(state);
    _solution[i] = res;
    _solved[i] = true;
  }
  return _solution[i];
}

bool gadget_check () {
  // read graph description
  uint N,M;
  cin >> N >> M;
  if (N > 16) {
    cerr << "Wrong data" << endl;
    return false;
  }
  vector<pair<uint, uint> > E;
  for(uint i=0;i<M;i++) {
    uint a,b;
    cin >> a >> b;
    if (a >= N || b >= N) {
      cerr << "Wrong data" << endl;
      return false;
    }
    E.push_back(make_pair(a,b));
  }

  uint Z;
  cin >> Z;

  for(uint i=0;i<Z;i++) {
    // single case
    vector<uint> A, B;
    cin >> M;
    for(uint j=0;j<M;j++) {
      uint a;
      cin >> a;
      if (a >= N) {
        cerr << "Wrong data" << endl;
        return false;
      }
      A.push_back(a);
    }
    cin >> M;
    for(uint j=0;j<M;j++) {
      uint b;
      cin >> b;
      if (b >= N) {
        cerr << "Wrong data" << endl;
```

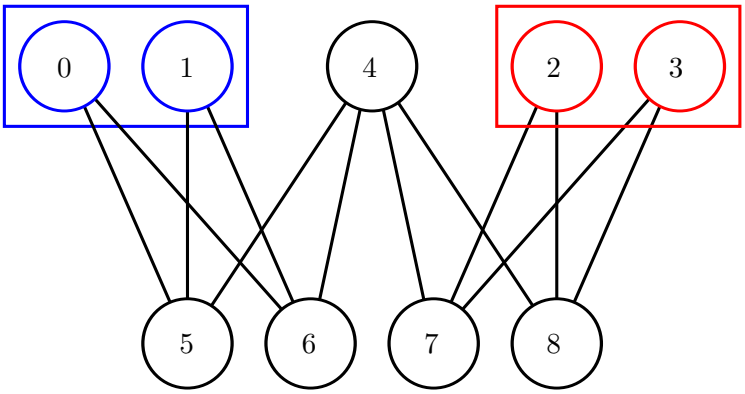
```
        return false;
      }
      B.push_back(b);
    }

    GameBoard board(N, E, A, B);
    GameSolution sol(board);
    GameState state(board);

    for(uint j=0;j<N;j++) {
      uint v;
      cin >> v;
      if (v > 3) {
        cerr << "Wrong data" << endl;
        return false;
      }
      state.paint_count[j] = 3-v;
    }
    string W;
    cin >> W;
    if(W == "Paint") {
      if (sol.solution(state)) {
        cerr << "Mr. Paint can't win in situtation " << i << endl;
        return false;
      }
    } else if (W == "Correct") {
      if (!sol.solution(state)) {
        cerr << "Mrs. Correct can't win in situation " << i << endl;
        return false;
      }
    } else {
      cerr << "Wrong data" << endl;
      return false;
    }
  }
  return true;
}

int main () {
  if (!gadget_check()) {
    cout << "FAIL" << endl;
    return 1;
  }
  cout << "OK" << endl;
  return 0;
}
```



A.1. Gadget G_{crit}

Below is the description of G_{crit} and Fact 3.10 in the format suitable for the program. There are two cases in the input file. The first shows the on-line 3-choosability of G_{crit}. In the second case the list-length-function is lowered to 2 on one of the vertices and the program proves that G_{crit} is no longer on-line choosable with the new list-length-function.

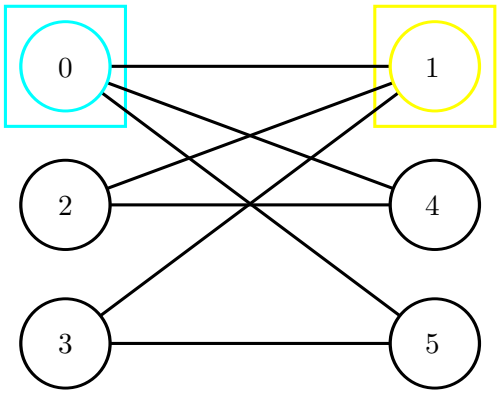
```
12
36
0 6
0 7
0 8
0 9
0 10
0 11
1 6
1 7
1 8
1 9
1 10
1 11
2 6
2 7
2 8
2 9
2 10
2 11
3 6
3 7
3 8
3 9
3 10
3 11
4 6
4 7
4 8
4 9
4 10
4 11
5 6
5 7
5 8
5 9
5 10
5 11

2

0
0
3 3 3 3 3 3 3 3 3 3 3 3 Correct

0
0
2 3 3 3 3 3 3 3 3 3 3 3 Paint
```



A.2. Gadget G_{cons}

Below is the description of G_{cons} and Fact 3.13 in the format suitable for the program. Observe that the role played by the blue and the red vertices is fully symmetric, so all the cases from Fact 3.13 are captured by the two cases in the input file. The first case handles all the situations in which Mrs. Correct is supposed to win, and the second case handles all the situations in which Mr. Paint is supposed to win.

```
9
12
0 5
0 6
1 5
1 6
2 7
2 8
3 7
3 8
4 5
4 6
4 7
4 8

2

0
0
2 2 3 3 2 2 2 2 2 Correct

0
0
2 3 2 3 2 2 2 2 2 Paint
```



A.3. Gadget G_{choice}

Below is the description of G_{choice} and Fact 3.15 in the format suitable for the program. Observe that the role played by the cyan and the yellow vertex is fully symmetric, so all the cases from Fact 3.15 are captured by the two cases in the input file. The first case handles all the situations in which Mrs. Correct is supposed to win, and the second case handles all the situations in which Mr. Paint is supposed to win. The first case is the only one with a PC$^{\text{pm}}$-game, not a PC-game.

```
6                                   2
7
0 1                                 1 0
0 4                                 1 1
0 5                                 3 3 2 2 2 2 Correct
1 2
1 3                                 0
2 4                                 0
3 5                                 2 2 2 2 2 2 Paint
```