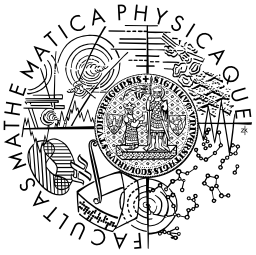


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Petr Illner

Návrh meta-algoritmu pro problém barvení grafu

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. Ing. Otakar Trunda
Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2019

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Rád bych poděkoval svému vedoucímu RNDr. Ing. Otakaru Trundovi za odborné vedení práce, cenné rady, připomínky a ochotu projevenou při konzultaci. Dále bych poděkoval své rodině za podporu a trpělivost při mém studiu. Mé díky patří i všem dalším učitelům za získané vědomosti v průběhu studia.

Název práce: Návrh meta-algoritmu pro problém barvení grafu

Autor: Petr Illner

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. Ing. Otakar Trunda, Katedra teoretické informatiky a matematické logiky

Abstrakt: Barvení grafů je jednou z nejdůležitějších oblastí teorie grafů a velmi využívaná v reálných situacích, zmiňme například: rozvrhovací problémy a alokace registrů. Cílem je obarvit vrcholy barvami tak, že žádné dva sousední vrcholy nesdílejí stejnou barvu. Jedná se o jeden z NP problémů. Existuje řada heuristických algoritmů, které dávají přibližné řešení. Hlavním cílem této práce je vytvoření umělé inteligence, která pro daný graf odhadne nejvhodnější barvící algoritmus.
Dalším cílem této práce je vylepšení metody interchange.
V této práci je také popsán nový algoritmus, který je pojmenovaný CLF (connected largest first). Jedná se o modifikaci LF (largest first) algoritmu.

Klíčová slova: Barvení grafu, algorithm selection, meta-algoritmy, kombinatorická optimalizace, grafové algoritmy, klasifikace grafů

Title: Design of a Meta-algorithm for the Graph Coloring Problem

Author: Petr Illner

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. Ing. Otakar Trunda, Department of Theoretical Computer Science and Mathematical Logic

Abstract: Graph coloring problem (GCP) is one of the most important concepts in graph theory and is used in many real-world problems such as time scheduling and register allocation. The aim of GCP is to color vertices of any given graph such that the colors on adjacent vertices are different. This problem is NP-hard. There are many heuristic algorithms that can be used to find an approximate solution. The main aim of this paper is to implement an artificial intelligence that tries to choose the most appropriate algorithm for graphs.
The next aim of this paper is to improve the interchange method.
In this thesis, we have designed a new algorithm for GCP called CLF (connected largest first). It is a modification of the largest first algorithm.

Keywords: Graph coloring, algorithm selection, meta-algorithms, combinatorial optimization, graph algorithms, classification of graphs

Obsah

Úvod **4**

1 Teorie grafů **6**
1.1 Graf . . . 6
1.1.1 Definice . . . 7
1.1.2 Reprezentace grafů v praxi . . . 9
1.1.3 Důležité grafy . . . 9
1.1.4 Grafový isomorfismus . . . 12
1.1.5 Grafové modifikace . . . 13
1.2 Grafová souvislost . . . 17
1.2.1 Most a artikulace . . . 17
1.2.2 Vrcholová a hranová souvislost . . . 18
1.3 Reprezentace grafu . . . 19
1.3.1 Seznam hran . . . 19
1.3.2 Matice sousednosti . . . 19
1.3.3 Seznam sousedů . . . 20
1.4 Stromy . . . 21
1.4.1 Kostra grafu . . . 22
1.5 Chordální grafy . . . 23
1.6 Náhodný graf . . . 25
1.6.1 Erdős–Rényi model . . . 25

2 Barvení grafů **28**
2.1 Vrcholové barvení grafů . . . 28
2.1.1 Aplikace barvení grafů . . . 31
2.2 Hranové barvení grafů . . . 32

3 Algoritmy **33**
3.1 Hladové obarvení . . . 33
3.2 Optimální obarvení . . . 34
3.2.1 Barvení chordálních grafů . . . 35
3.3 Sekvenční algoritmy . . . 35
3.3.1 Random sequence metoda . . . 35
3.3.2 Largest first metoda . . . 36
3.3.3 Smallest last metoda . . . 37
3.3.4 Connected sequential metoda . . . 39
3.3.5 Interchange metoda . . . 41
3.4 Saturation largest first metoda . . . 44
3.5 Greedy independent sets metoda . . . 45
3.6 Kombinační metoda . . . 46
3.7 Genetický algoritmus . . . 46
3.8 Connected largest first metoda . . . 48
3.9 Rozšíření metody interchange . . . 49
3.10 Benchmark a weak benchmark . . . 55

4 Meta-algoritmus **57**

5 Uživatelská dokumentace **58**
5.1 Program - GraphColoring . . . 58
5.1.1 Hlavní panel . . . 59
5.1.2 Panel pro zobrazení grafových vlastností . . . 60
5.1.3 Panel pro modifikace grafu . . . 60
5.1.4 Panel pro grafové operace . . . 62
5.1.5 Komponenta pro zobrazení grafu . . . 62
5.1.6 Soubory s příponou *.graph* . . . 63
5.2 Program - GraphColoringConsole . . . 66
5.2.1 Generování náhodných grafů . . . 66
5.2.2 Vytvoření modelů pro meta-algoritmus . . . 69
5.2.3 Vložení grafů ze souborů do databáze . . . 69
5.2.4 Převod z *.col* na *.graph* . . . 70
5.2.5 Měření časových složitostí algoritmů . . . 70
5.2.6 Obarvování grafů . . . 71

6 Vývojová dokumentace **72**
6.1 Databáze . . . 73
6.1.1 Tabulka GraphColoringGraph . . . 74
6.1.2 Tabulka GraphColoringCore . . . 74
6.1.3 Tabulka GraphColoringBest . . . 74
6.2 Backend - GraphColoring . . . 75
6.2.1 Graph . . . 75
6.2.2 GraphColoringAlgorithm . . . 81
6.2.3 GenerateGraph . . . 83
6.2.4 GraphVisualization . . . 84
6.2.5 ReaderWriter . . . 85
6.2.6 Tests . . . 86
6.3 Frontend - GraphColoring . . . 88
6.4 Backend - GraphColoringConsole . . . 89
6.4.1 Database . . . 89
6.4.2 ML . . . 89

7 Experimenty **92**
7.1 Obarvování grafů z DIMACS kolekce grafů . . . 92
7.1.1 Tabulky s obarveními . . . 95
7.1.2 Grafy . . . 103
7.2 Měření doby trvání algoritmů . . . 105
7.2.1 Tabulky . . . 105
7.3 Obarvování náhodných grafů . . . 111
7.3.1 Grafy . . . 111
7.4 Meta-algoritmus . . . 118
7.4.1 Grafy . . . 118

Závěr **120**

Seznam použité literatury **122**

Seznam obrázků **123**

Seznam tabulek **126**

Seznam použitých zkratek **127**

A Příloha - Ohodnocení AI modelů **129**

B Příloha - Obsah CD **130**

Úvod

Barvení grafů je jeden z nejstarších (počátky v 19. století) a často zkoumaných oblastí *teorie grafů*. Existuje celá řada variant barvení grafů, tato práce se zabývá tzv. *klasickým barvením* (classical coloring). Cílem je vrcholům grafu G přiřadit barvy tak, že každému vrcholu je přiřazena právě jedna barva a navíc musí platit, že každé dva sousední vrcholy (tedy vrcholy spojeny hranou) nemají stejnou barvu. Obarvení grafu G k barvami se říká *k-obarvení* (k-coloring). Nejmenší k, pro které existuje obarvení grafu G se nazývá *chromatické číslo* (chromatic number), které se značí $\chi(G)$. Cílem je tedy pro daný graf nalézt jeho chromatické číslo a odpovídající obarvení vrcholů.

Barvení grafů se využívá v praxi velmi často, zmiňme například plánování procesů (rozvrh hodin, alokace registrů při překladu programu apod.).

Barvení grafů je jeden z nejpopulárnějších *NP problém*ů. Jinak řečeno, na zjištění chromatického čísla pro libovolný graf se nezná efektivní algoritmus (tj. algoritmus s polynomiální časovou složitostí). Existuje ale celá řada algoritmů využívající různé typy heuristik, díky kterým se snaží co nejlépe obarvit grafy. Barvící algoritmy fungují různě dobře na jednotlivých grafech, a proto má smysl vytvořit nějaký meta-algoritmus, který pro daný graf vybere nejvhodnější algoritmus.

Cíle této práce jsou:

- **vytvoření aplikace na barvení grafů,**
- **vytvoření meta-algoritmu,**
- **sepsání skript zaměřující se na barvení grafů,**
- **vymyšlení nového barvícího algoritmu a nějaké metody, která by vylepšila stávající barvící algoritmy.**

Aplikace bude naimplementována v jazyku C# a bude mít následující funkcionality:

- načítání a generování grafů,
- obarvování grafů,
- zjišťování vlastností grafů,
- vykreslování (obarvených) grafů a jejich vlastností.

Tato aplikace může být díky vykreslování (obarvených) grafů a jejich vlastností použita jako výuková pomůcka. Aplikace navíc bude obsahovat meta-algoritmus, který na základě vlastností grafu vybere nejvhodnější barvící algoritmus.

Kapitoly 1, 2 a 3 mohou sloužit jako rozšíření knížky *Kapitoly z diskrétní matematiky*[1], která se barvením grafů věnuje jen okrajově. Toto je i jeden z důvodů, proč se v celé práci používá stejná terminologie jako ve zmiňované knížce.

Kapitola 1 se zabývá teorií grafů, kde se vyskytují důležité definice a věty, které jsou spjaty s barvením grafů (většina je potřebná pro pochopení funkcionality aplikace, zbytek je zmíněn pouze pro úplnost). Kapitola 2 se zaměřuje pouze

na barvení grafů (věty v této kapitole jsou uvedeny pouze pro úplnost a nejsou nijak potřebné k pochopení chování algoritmů a funkčnosti aplikace). V podkapitole 2.2 navíc ukážeme, že aplikace může sloužit i pro hranové obarvení. Tyto dvě kapitoly jsou psány pro čtenáře, který nemá žádné zkušenosti s teorií grafů, a proto tyto kapitoly obsahují spoustu poznámek, které mají čtenáři propojit řadu na první pohled nesouvisejících věcí. Čtenář znalý teorie grafů může tyto dvě kapitoly směle přeskočit. Třetí kapitola se věnuje barvícím algoritmům. Většina informací (popis algoritmů, SHC a HC grafy pro jednotlivé algoritmy) v této kapitole byla získána z publikace *Classical Coloring of Graphs*[2]. Kapitola navíc popisuje nový algoritmus *CLF* a dvě metody (*rozšířený interchange a rozšířený interchange s přebarvením K3*), které se dají využít k vylepšení stávajících barvících algoritmů. Kapitola 4 obsahuje návrh meta-algoritmu. Kapitola 5 obsahuje uživatelskou dokumentaci k aplikaci, kde jsou popsány funkcionality aplikace podrobněji včetně popsání uživatelského rozhraní. Kapitola 6 obsahuje vývojovou dokumentaci, kde je popis činnosti aplikace, popis jednotlivých rozhraní apod. Poslední kapitola 7 se věnuje experimentům, kde budeme porovnávat algoritmus *CLF* s ostatními algoritmy a otestujeme rozšířenou metodu interchnage a rozšířenou metodu interchange s přebarvením K3 na již existujících algoritmech. Navíc změříme efektivitu meta-algoritmu.

1. Teorie grafů

1.1 Graf

Velké množství situací v informatice a v matematice lze zpravidla popsat pomocí dvou věcí:

- množinou objektů (bodů)
- a vztahy mezi objekty (spojnicemi mezi body).

Neformálně, graf je množina bodů, z nichž některé jsou spojeny čárami. Bohužel tato neformální definice je nedostačující.

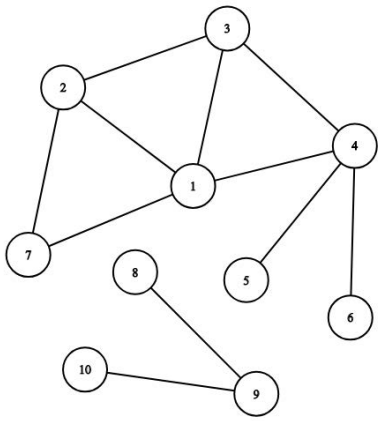
Definice 1. Graf *(graph) je uspořádaná dvojice (V, E), kde*

- *V je neprázdná množina, jejíž elementy se nazývají* vrcholy *(vertices)*
- *a E je množina dvouprvkových podmnožin množiny V, prvky množiny E se nazývají* hrany *(edges).*

Poznámka. V naší definici tedy graf musí mít alespoň jeden vrchol. Někteří povolují i grafy bez vrcholů, což je hloupý protipříklad pro řadu vět a lemmat, a proto tento typ grafu zakazujeme.

Poznámka. Existuje více variant grafů. Grafu z definice 1 se říká *neorientovaný (prostý) graf* (undirected graph). Dalším typem grafu je *orientovaný graf*, kde hrana, označovaná jako orientovaná hrana (directed edge, arc), je uspořádaná dvojice vrcholů. Lze si pod tím představit, že hrana směřuje z nějakého vrcholu do jiného vrcholu.

Příklad. Grafy se znázorňují kreslením do roviny, kde vrcholy odpovídají bodům a hrany odpovídají čárám. Obrázek 1.1 tedy reprezentuje graf (V, E), kde V = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} a E = {{1, 2}, {1, 3}, {1, 4}, {1, 7}, {2, 3}, {2, 7}, {3, 4}, {4, 5}, {4, 6}, {8, 9}, {9, 10}}.



Obrázek 1.1: Graf

1.1.1 Definice

Definice 2. *Řekněme, že dva vrcholy jsou* sousední *(adjacent), pokud jsou spojeny hranou a říkáme, že tato hrana je* incidentní *(incident) k těmto dvěma vrcholům.*

Definice 3. *Nechť G je graf a v je jeho vrchol.* Sousedství *(neighborhood) vrcholu v je množina vrcholů, které jsou sousední s vrcholem v. Tuto množinu budeme značit* $N_G(v)$.

Příklad. Pro graf na obr. 1.1 jsou vrcholy *1* a *2* sousední, neboť jsou spojeny hranou a tedy hrana $\{1, 2\}$ je incidentní k vrcholům *1* a *2*. Sousedství vrcholu *1* je $N_G(1) = \{2, 3, 4, 7\}$.

Definice 4. *Nechť G je graf a v je jeho vrchol.* Stupeň vrcholu *(degree of vertex) v, který značíme* $deg_G(v)$*, definujeme jako počet hran incidentních s vrcholem v, což je ekvivalentní počtu sousedů vrcholu v, tj.* $|N_G(v)|$.

Definice 5. *Nechť G je graf.* Maximální stupeň *(maximum degree) grafu G, který značíme* $\Delta(G)$*, je největší stupeň ze všech jeho vrcholů. Obdobně,* minimální stupeň *(minimum degree) grafu G, který značíme* $\delta(G)$*, je nejmenší stupeň ze všech jeho vrcholů. Číslu* $g(G) = \frac{2 \times |E|}{|V| \times (|V| - 1)}$ *se říká* hustota grafu *(density of graph).*

Příklad. Pro graf na obr. 1.1 stupeň vrcholu *2* je $deg_G(2) = 3$. Maximální stupeň grafu je $\Delta(\mathrm{G}) = 4$ a minimální stupeň grafu je $\delta(\mathrm{G}) = 1$.

Definice 6. *Nechť G je graf a* v_1, v_2, ..., v_n *jsou jeho vrcholy. Potom posloupností* $(deg_G(v_1), deg_G(v_2), ..., deg_G(v_n))$ *označujeme* skóre grafu *(degree sequence). Skóre grafu nemusí být nutně setříděné.*

Příklad. Skóre grafu pro graf na obr. 1.1 je (1, 1, 1, 1, 2, 2, 3, 3, 4, 4).

Věta 1 (Princip sudosti (handshaking lemma))**.** *Pro každý graf G = (V, E) platí*

$$\sum_{v \in V} deg_G(v) = 2 \ * \ |E|.$$

Jinými slovy, počet účastníků večírku, kteří si potřásli rukou s lichým počtem jiných účastníků, je sudé číslo.

Důkaz. Stupeň vrcholu *v* grafu G určuje počet hran incidentních s vrcholem *v*. Každá hrana je incidentní s právě dvěma vrcholy. Proto po sečtení všech stupňů vrcholů dostáváme dvojnásobek počtu hran.

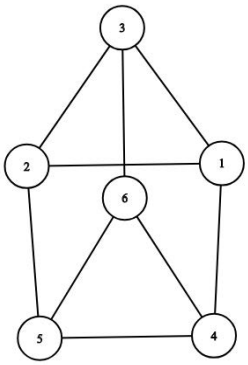
□

Důsledek. Počet vrcholů lichého stupně je sudé číslo pro každý graf.

Definice 7. *Graf G se nazývá* k-regulární *(k-regular graph), pokud jsou stupně všech vrcholů rovny přesně k.*

Poznámka. Pro k-regulární graf platí, že $\Delta(\mathrm{G}) = \delta(\mathrm{G}) = \mathrm{k}$.

Příklad. Na obr. 1.2 je příklad 3-regulárního grafu na 6 vrcholech.



Obrázek 1.2: 3-regulární graf

Poznámka. Nechť G = (V, E) je graf. Potom E(G) budeme označovat množinu hran grafu G a V(G) budeme označovat množinu vrcholů grafu G.

Definice 8. *Nechť G je graf. Řekneme, že graf H je* podgrafem *(subgraph) grafu G, jestliže V(H) ⊆ V(G) a E(H) ⊆ E(G).*

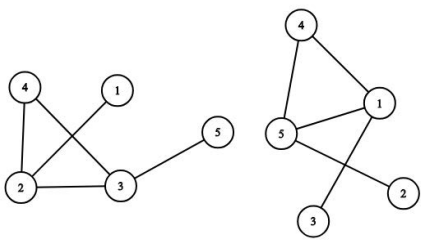
Definice 9. *Nechť G = (V, E) je graf. Řekneme, že graf H je* indukovaným podgrafem *(induced subragph) grafu G, značeno H ≤ G, jestliže V(H) ⊆ V(G) a* $E(H) = E(G) \cap \binom{V(H)}{2}$.

Definice 10. *Nechť G = (V, E) je graf.* Kliknou *(clique) grafu G nazveme množinu V' ⊆ V takovou, že* $\forall v_1, v_2 \in V'$ *platí* $\{v_1, v_2\} \in E$. *Výraz klika se také často používá pro indukovaný podgraf grafu G s množinou vrcholů V'. Klika V' v grafu G se nazývá maximální, pokud neexistuje žádná větší klika (do počtu vrcholů). Velikost maximální kliky v grafu G se značí* $\omega(G)$.

Definice 11. *Nechť G = (V, E) je graf.* Nezávislou množinou *(independent set) grafu G nazveme množinu V' ⊆ V takovou, že* $\forall v_1, v_2 \in V'$ *platí* $\{v_1, v_2\} \notin E$. *Nezávislá množina V' v grafu G se nazývá maximální, pokud neexistuje větší nezávislá množina (do počtu vrcholů). Velikost maximální nezávislé množiny v grafu G se značí* $\alpha(G)$.

Definice 12. *Nechť G = (V, E) je graf. Pak definujeme* komplementární graf *(complement graph) ke grafu G, označovaný jako* $\bar{G} = (V, \bar{E})$, *kde* $\forall v_1, v_2 \in V$ *platí* $\{v_1, v_2\} \in E$ *právě tehdy, když* $\{v_1, v_2\} \notin \bar{E}$.

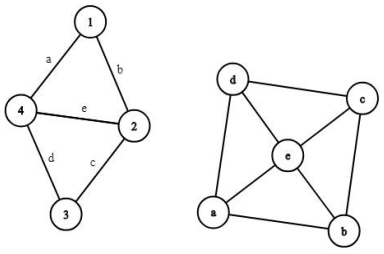
Příklad. Na obr. 1.3 je příklad dvou komplementárních grafů.



Obrázek 1.3: Komplementární grafy

Definice 13. *Nechť G = (V, E) je graf. Pak ke grafu G definujeme* hranový graf *(line graph), označovaný jako L(G), tak, že hrany grafu G budou vrcholy grafu L(G) a budou spojeny hranou, pokud v grafu G měly ony hrany společný vrchol. Tedy V(L(G)) = E(G) a* $(e_1, e_2) \in E(L(G)) \Leftrightarrow |e_1 \cap e_2| = 1$.

Příklad. Na obr. 1.4 je graf G a k němu hranový graf L(G).



Obrázek 1.4: Hranový graf

Definice 14. *Graf G je* d-degenerovaný *(d-degenerate) právě tehdy, když každý jeho neprázdný podgraf H grafu G obsahuje vrchol stupně* $\leq d$ *(vzhledem k H).*

Poznámka. Každý graf G je $\Delta(G)$-degenerovaný.

1.1.2 Reprezentace grafů v praxi

Existuje celá řada reálných situací, ve kterých se používají grafy. Tady jsou uvedeny některé praktické situace s jejich grafickými reprezentacemi:

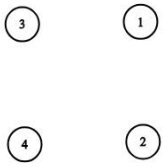
- *internet* - každá stránka je reprezentována jedním vrcholem. Orientovaná hrana mezi dvěma vrcholy (stránkami) reprezentuje hyperlink.
- *datové struktury* - každý vrchol reprezentuje nějaký datový objekt. Dva vrcholy (objekty) jsou spojeny orientovanou hranou, pokud první objekt obsahuje pointer (ukazatel) na druhý objekt.
- *letecké spoje* - letiště jsou reprezentované vrcholy. Dvě letiště jsou spojena orientovanou hranou, pokud existuje nějaký let mezi nimi.

1.1.3 Důležité grafy

Některé grafy se používají tak často, že se pro ně ujalo standardní názvosloví a označení.

Diskrétní graf (empty graph) je graf $\bar{K_n}$ = (V, E), kde V = $\{v_1, \ldots, v_n\}$ a E = $\emptyset$, kde n $\geq$ 1. Tedy prázdný graf nemá žádné hrany.

Příklad. Příklad diskrétního grafu $\bar{K_4}$ je na obr. 1.5.



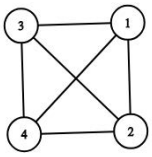
Obrázek 1.5: Diskrétní graf $\bar{K_4}$

Úplný graf (complete graph) je graf K_n = (V, E), kde V = $\{v_1, \ldots, v_n\}$ a E = $\binom{V}{2}$, kde n $\geq$ 1. Úplný graf má tedy hranu mezi každou dvojicí vrcholů.

Poznámka. Úplný graf K_n je $(n - 1)$-regulární graf.

Poznámka. Úplný graf K_n je komplementárním grafem k prázdnému grafu $\bar{K_n}$.

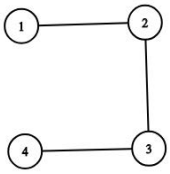
Příklad. Příklad úplného grafu K_4 je na obr. 1.6.



Obrázek 1.6: Úplný graf K_4

Cesta (path) je graf P_n = (V, E), kde V = $\{v_1, \ldots, v_n\}$ a E = $\{\{v_{i-1}, v_i\}$; i = 1, ..., n$\}$, kde n $\geq$ 1. Vrcholy v_1 a v_n se nazývají koncové body cesty.

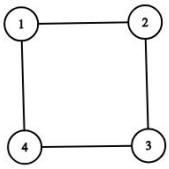
Příklad. Příklad cesty P_4 je na obr. 1.7.



Obrázek 1.7: Cesta P_4

Kružnice (cycle) je graf C_n = (V, E), kde V = $\{v_1, \ldots, v_n\}$ a E = $\{\{v_i, v_{i+1}\}$; i = 1, ..., $n - 1\} \cup \{\{v_1, v_n\}\}$, kde n $\geq$ 3.

Příklad. Příklad kružnice C_4 je na obr. 1.8.



Obrázek 1.8: Kružnice C_4

Bipartitní graf (bipartite graph) je graf, jehož množinu vrcholů lze rozdělit na dvě disjunktní podmnožiny V_1 a V_2 takové, že každá hrana spojuje vrchol z V_1 s vrcholem z V_2. Tedy bipartitní graf je graf G = (V, E), kde V = $V_1 \cup V_2$ a E $\subseteq \{\{v_1, v_2\}; v_1 \in V_1, v_2 \in V_2\}$.

Lemma 2. *Jestliže graf je bipartitní, pak neobsahuje kružnici liché délky.*

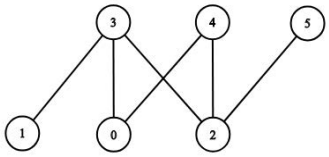
Důkaz. Nechť G je bipartitní graf s partitami V_1 a V_2. Každá hrana v grafu je incidentní s jedním vrcholem z partity V_1 a s druhým vrcholem z partity V_2. Tedy libovolná cesta, která začíná a končí ve stejné partitě musí mít sudý počet hran a tedy každá kružnice v grafu musí mít sudý počet hran.

□

Poznámka. V předchozím lemmatu dokonce platí i obrácená implikace.

Poznámka. Pojem bipartitnosti lze rozšířit pro libovolné k ≥ 2. Graf G = (V, E) nazveme *k-partitním* grafem, pokud množinu V lze rozdělit na k disjunktních množin takových, že dva vrcholy ze stejné množiny nejsou spojeny hranou. Významnost těchto grafů vyplyne v kapitole 2.

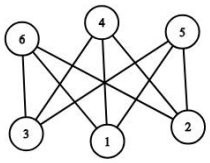
Příklad. Příklad bipartitního grafu je na obr. 1.9.



Obrázek 1.9: Bipartitní graf

Úplný bipartitní graf (complete bipartite graph) je graf $K_{n,m}$, kde V = $\{u_1, \ldots, u_n\} \cup \{v_1, \ldots, v_m\}$ a E = $\{\{u_i, v_j\}$; i = 1, …, n, j = 1, …, m$\}$.

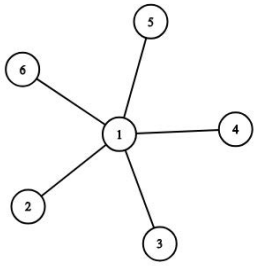
Příklad. Příklad úplného bipartitního grafu $K_{3,3}$ je na obr. 1.10.



Obrázek 1.10: Úplný bipartitní graf $K_{3,3}$

Hvězda (star) je úplný bipartitní graf označovaný S_n, kde jedna partita obsahuje pouze jeden vrchol.

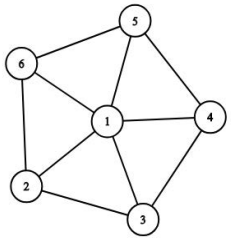
Příklad. Příklad hvězdy S_6 je na obr. 1.11.



Obrázek 1.11: Hvězda S_6

Kolo (wheel) je graf W_n obsahující kružnici a vrchol, který neleží na dané kružnici, spojený hranami se všemi vrcholy kružnice.

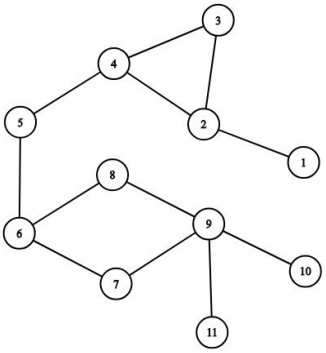
Příklad. Příklad kola W_6 je na obr. 1.13.



Obrázek 1.12: Kolo W_6

Kaktus (cactus) je graf, ve kterém každé dva libovolné cykly mají nejvýše jeden společný vrchol.

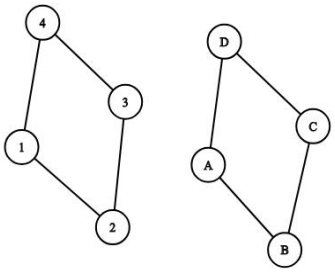
Příklad. Příklad kaktusu je na obr. 1.13.



Obrázek 1.13: Kaktus

1.1.4 Grafový isomorfismus

Dva grafy, které vypadají stejně, mohou být po formální stránce naprosto odlišné. Jako například grafy na obr. 1.14. Protože první graf má množinu vrcholů {1, 2, 3, 4} a druhý graf má množinu vrcholů {A, B, C, D}, tak se jedná o rozdílné matematické objekty. Naštěstí umíme definovat, že dva grafy "vypadají stejně".

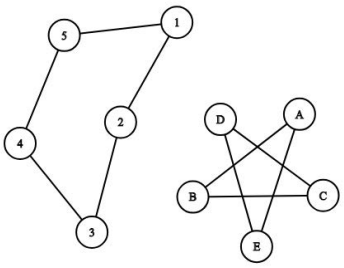


Obrázek 1.14: Dva stejně vypadající grafy

Definice 15. *Nechť $G_1 = (V_1, E_1)$ a $G_2 = (V_2, E_2)$ jsou grafy. Grafy G_1 a G_2 nazveme* isomorfní *(isomorphic), jestliže existuje vzájemně jednoznačné (bijektivní) zobrazení $f : V_1 \hookrightarrow\!\!\!\!\rightarrow V_2$ takové, že $\{x, y\} \in E_1$ právě když $\{f(x), f(y)\} \in E_2$. Fakt, že grafy G_1 a G_2 jsou isomorfní, vyznačujeme zápisem $G_1 \cong G_2$.*

Poznámka. Isomorfismus je tedy "přejmenování vrcholů" grafu. Tudíž dva isomorfní grafy mají stejné vlastnosti jako počet vrcholů, počet hran, skóre grafu apod. Pokud dva grafy nemají stejné vlastnosti, tak nemohou být isomorfní.

Příklad. Grafy z obr. 1.15 jsou isomorfní. Jeden z isomorfismů může například být $1 \mapsto$ A, $2 \mapsto$ B, $3 \mapsto$ C, $4 \mapsto$ D, $5 \mapsto$ E. Je vidět, že dva isomorfní grafy lze nakreslit tak, že na první pohled nevypadají isomorfně.



Obrázek 1.15: Isomorfní grafy

1.1.5 Grafové modifikace

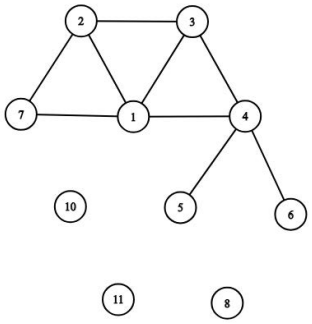
Zavedeme několik grafových operací.

Definice 16. *Nechť $G = (V, E)$ je graf.*

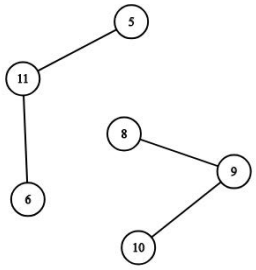
- Přidání vrcholu *(adding vertex) $v \notin V(G)$ do grafu G, označované jako $G + v$, vytvoří nový graf $G' = (V \cup \{v\}, E)$. Nově přidaný vrchol tedy není incidentní s žádnou hranou.*
- Odebrání vrcholu *(deleting vertex) $v \in V(G)$ v grafu G, označované jako $G - v$, vytvoří nový graf $G' = (V \setminus \{v\}, \{e \in E; v \notin e\})$.*
- Kontrakce vrcholu *(vertex contraction) $v \in V(G)$ v grafu G sloučí všechny sousedy vrcholu v včetně vrcholu v do nového vrcholu.*
- Potlačení vrcholu *(vertex suppression) $v \in V(G)$, kde $deg_G(v) = 2$, v grafu G odstraní vrchol v a jeho sousední vrcholy spojí hranou.*

- Expanze vrcholu *(vertex expansion) $v \in V(G)$ v grafu G vytvoří nový vrchol v_k, který se propojí hranou se všemi sousedy vrcholu v včetně vrcholu v.*
- Přidání hrany *(adding edge) $e = \{v_1, v_2\}$, kde $e \notin E(G)$ a $v_1, v_2 \in V(G)$, do grafu G, označované jako $G + e$, vytvoří nový graf G' = (V, $E \cup \{e\}$).*
- Odebrání hrany *(deleting edge) $e = \{v_1, v_2\}$, kde $e \in E(G)$, v grafu G, označované jako $G - e$, vytvoří nový graf G' = (V, $E \setminus \{e\}$).*
- Kontrakce hrany *(edge contraction) $e = \{v_1, v_2\}$, kde $e \in E(G)$, označovaná jako $G.e$, v grafu G odstraní hranu e a sloučí jeho dva incidentní vrcholy v_1 a v_2.*
- Dělení hrany *(edge subdivision) $e = \{v_1, v_2\}$, kde $e \in E(G)$, v grafu G, označované jako G % e, nahradí hranu e cestou délky 3, kde koncové vrcholy jsou v_1, v_2 a vnitřní vrchol je nový vrchol. Tedy G % e = ($V \cup \{v\}$, $E \setminus \{e\} \cup \{\{v_1, v\}, \{v_2, v\}\}$).*

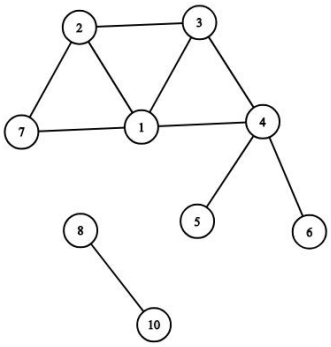
Příklad. Příklady grafových modifikací aplikovaných na graf na obr. 1.1 jsou na obrázcích 1.16 - 1.23.



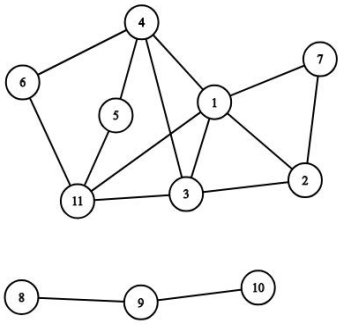
Obrázek 1.16: Přidání vrcholu *11* a odebrání vrcholu *9* v grafu na obr. 1.1



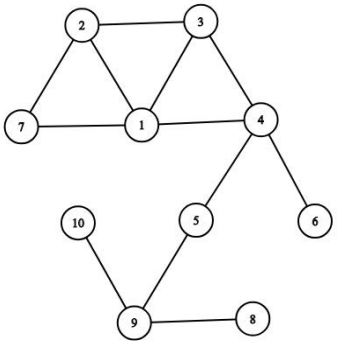
Obrázek 1.17: Kontrakce vrcholu *1* v grafu na obr. 1.1



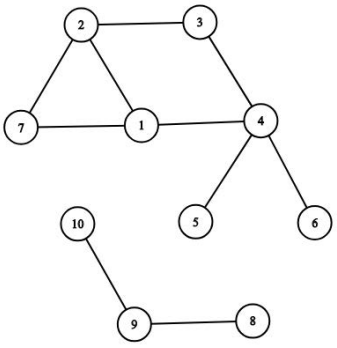
Obrázek 1.18: Potlačení vrcholu *9* v grafu na obr. 1.1



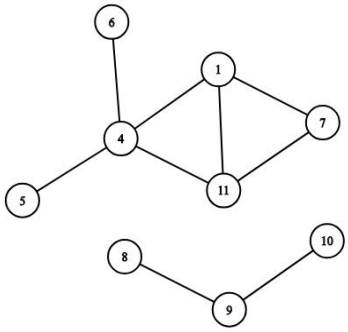
Obrázek 1.19: Expanze vrcholu *4* v grafu na obr. 1.1



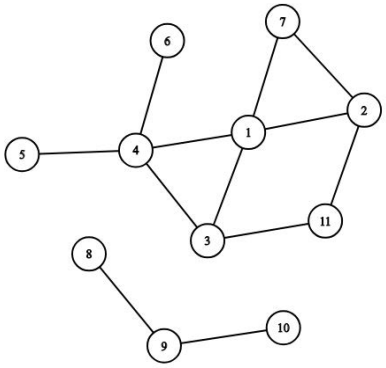
Obrázek 1.20: Přidání hrany $e = \{9, 5\}$ do grafu na obr. 1.1



Obrázek 1.21: Odebrání hrany $e = \{1, 3\}$ z grafu na obr. 1.1



Obrázek 1.22: Kontrakce hrany $e = \{1, 3\}$ v grafu na obr. 1.1



Obrázek 1.23: Dělení hrany $e = \{2, 3\}$ v grafu na obr. 1.1

1.2 Grafová souvislost

Definice 17. *Podgraf grafu G = (V, E) isomorfní nějaké cestě P_t se nazývá* cesta *(path) v grafu G. Cestu v grafu G můžeme též chápat jako posloupnost (v_0, e_1, v_1, ..., e_t, v_t), kde $v_i \in V$ jsou po dvojicích různé a $e_i = \{v_{i-1}, v_i\} \in E(G)$ pro každé i = 1, ..., t.*

Definice 18. *Podgraf grafu G = (V, E) isomorfní nějaké kružnici C_t ($t \geq 3$) se nazývá* kružnice *(cycle) v grafu G. Kružnici v grafu G můžeme taky chápat jako posloupnost (v_0, e_1, v_1, ..., e_t, v_0), kde $v_i \in V$ jsou po dvojicích různé a $e_i = \{v_{i-1}, v_i\} \in E(G)$ pro každé i = 1, ..., t - 1 a $e_t = \{v_{t-1}, v_0\} \in E(G)$.*

Definice 19. *Graf G je* souvislý *(connected), pokud pro každou dvojici vrcholů v_1 a v_2 v něm existuje cesta z v_1 do v_2. Jinak říkáme, že graf je* nesouvislý *(disconnected).*
Maximální souvislý podgraf se nazývá souvislá komponenta *(connected component).*

Věta 3. *Každý graf G = (V, E) má alespoň $|V| - |E|$ souvislých komponent.*

Důkaz. Důkaz provedeme indukcí podle počtu hran v grafu. Nechť indukční předpoklad je, že každý graf G = (V, E) s n hrany má alespoň $|V| - n$ souvislých komponent.
Základní případ: v grafu s 0 hrany je každý vrchol souvislá komponenta. Tedy graf má přesně $|V| - 0 = |V|$ souvislých komponent.
Indukční krok: předpokládejme, že indukční předpoklad platí pro každý graf s n hranami a chceme dokázat, že platí i pro grafy s $(n + 1)$ hranami, kde $n \geq 0$. Nechť G = (V, E) je graf s $(n + 1)$ hranami. Z grafu G odstraníme libovolnou hranu $\{v_1, v_2\}$ a nově vzniklý graf označíme G'. Z indukčního předpokladu má G' alespoň $|V| - n$ souvislých komponent. Nyní přidáme zpátky hranu $\{v_1, v_2\}$, abychom získali graf G. Pokud vrcholy v_1 a v_2 byly ve stejné souvislé komponentě G', tak G má stejný počet souvislých komponent jako G', což je alespoň $|V| - n$. Jinak, pokud vrcholy v_1 a v_2 byly v různých komponentách souvislosti G', tak došlo ke spojení těchto dvou souvislých komponent v G, ale všechny ostatní souvislé komponenty zůstaly. Tedy G má alespoň $|V| - n - 1 = |V| - (n + 1)$ souvislých komponent.

□

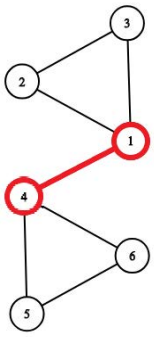
Důsledek. Každý souvislý graf s n vrcholy má alespoň $(n - 1)$ hran.

1.2.1 Most a artikulace

Definice 20. *Nechť G = (V, E) je graf.* Most *(edge-cut, bridge) je taková hrana v G, jejímž odstraněním se zvýší počet komponent grafu G.*

Definice 21. *Nechť G = (V, E) je graf.* Artikulace *(vertex-cut) je takový vrchol v G, jehož odstraněním (spolu s incidenčními hranami) se zvýší počet komponent alespoň o jedna.*

Příklad. Pro graf na obr. 1.24 jsou vrcholy *1* a *4* artikulace a hrana *{1, 4}* je most.



Obrázek 1.24: Graf s artikulacemi a mosty

1.2.2 Vrcholová a hranová souvislost

Definice 22. *Nechť $G = (V, E)$ je graf a $\kappa > 1$. Řekněme, že graf G je* vrcholově κ-souvislý *(κ-vertex-connected), pokud po odebrání libovolných nejvýše $\kappa - 1$ vrcholů z grafu G, zůstane výsledný graf souvislý.*

Definice 23. *Nechť $G = (V, E)$ je graf a $\kappa' > 1$. Řekněme, že graf G je* hranově κ'-souvislý *(κ'-edge-connected), pokud po odebrání libovolných nejvýše $\kappa' - 1$ hran z grafu G, zůstane výsledný graf souvislý.*

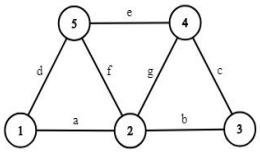
1.3 Reprezentace grafu

Jakým způsobem ale ukládat grafy v počítači? Ukládat grafy pomocí geometrických prostředků je nedostačující. Naštěstí máme i další způsoby, jak reprezentovat graf, které jsou pro tento účel vhodnější. Obvyklé jsou následující reprezentace:

- *seznam hran* (edge list),
- matice sousednosti (adjacency matrix) a
- seznam sousedů (adjacency list).

Každá reprezentace má zpravidla jiné časové složitosti pro operace nad grafy. Tedy danou reprezentaci vybíráme na základě toho, co s grafy budeme dělat.

Nechť máme graf G = (V, E), kde V = $\{v_1, \ldots, v_n\}$ a E = $\{e_1, \ldots, e_m\}$.



Obrázek 1.25: Graf

1.3.1 Seznam hran

Graf je reprezentovaný seznamem, který obsahuje všechny hrany grafu, tj. dvojice vrcholů. Seznam je tedy tvaru $e_1, e_2, \ldots, e_m$.

Zjištění, zda dva vrcholy jsou spojeny hranou trvá $\mathcal{O}(m)$. Stejnou časovou složitost má i získání všech sousedů nějakého vrcholu, protože se musí projít celý seznam hran, jehož délka je m.

Paměťová složitost této reprezentace je $\mathcal{O}(m)$.

Příklad. Graf z obr. 1.25 je reprezentován pomocí seznamu hran následovně: (1, 2), (2, 3), (3, 4), (1, 5), (4, 5), (2, 5), (2, 4).

1.3.2 Matice sousednosti

Matice sousednosti je čtvercová matice typu $n \times n$, jejíž řádky a sloupce jsou indexovány vrcholy, definovaná předpisem:

$$a_{ij} = \begin{cases} 1 & \{v_i, v_j\} \in \mathrm{E}, \\ 0 & \text{jinak.} \end{cases}$$

Výhoda této reprezentace je, že můžeme v konstantním čase zjistit, zda jsou dva vrcholy spojeny hranou, neboť pro dva vrcholy v_i a v_j stačí zjistit, zda v matici sousednosti je na pozici (i, j) jednička, nebo nula. Získání všech sousedů nějakého vrcholu v_i má časovou složitost $\mathcal{O}(n)$, neboť musíme projít celý i-tý řádek / sloupec matice sousednosti.

Paměťová složitost je $\mathcal{O}(n^2)$. Pro řídké grafy, tj. grafy s podstatně menším než kvadratickým počtem hran, je tato reprezentace paměťově neoptimální, neboť většina prvků matice budou nulové.

Poznámka. Matice sousednosti je pro neorientované grafy vždy symetrická a na diagonále se vyskytují pouze nuly.

Příklad. Matice sousednosti pro graf z 1.25 vypadá následovně:

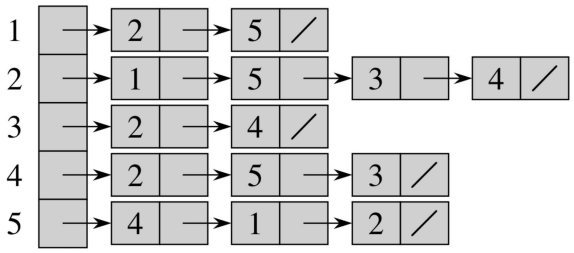
$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}.$$

1.3.3 Seznam sousedů

Pro každý vrchol je vytvořen seznam jeho sousedů. Cílem této reprezentace je zamezit plýtváním paměti u řídkých grafů.

Časová složitost na nalezení všech sousedů je $\mathcal{O}(\Delta(G))$, neboť se jedná pouze o průchod seznamu sousedů daného vrcholu. Zjištění, zda dva vrcholy jsou spojeny hranou, má stejnou časovou složitost, neboť také u daného vrcholu musíme projít v nejhorším případě celý seznam jeho sousedů.

Paměťová složitost této reprezentace je $\mathcal{O}(n + m)$.

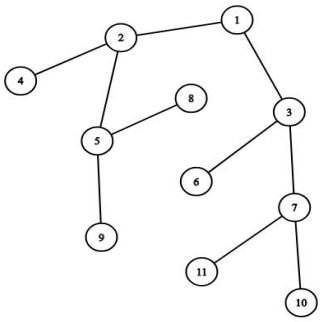


Obrázek 1.26: Seznam sousedů pro graf z 1.25

1.4 Stromy

Definice 24. *Souvislý, acyklický graf se nazývá* strom *(tree). (Graf je acyklický právě tehdy, když neobsahuje žádnou kružnici.)*

Příklad. Příklad stromu je na obr. 1.27.



Obrázek 1.27: Strom

Definice 25. *Vrchol stromu se stupněm 1 se nazývá* list *(leaf). Ostatním vrcholům se říká* vnitřní vrcholy *(inner vertices).*

Příklad. Listy stromu z obr. 1.27 jsou {4, 6, 8, 9, 10, 11}.

Pokud by byla ze stromu odstraněná libovolná hrana nebo vnitřní vrchol, tak by se již nejednalo o strom, neboť by byl nesouvislý. Pokud bychom naopak do grafu přidali libovolnou hranu, tak by se zase nejednalo o strom, neboť by obsahoval cyklus.

Existuje ještě takzvaný *zakořeněný strom* (rooted tree), který má jeden významný vrchol označovaný jako *kořen* (root). Často se místo zakořeněného stromu říká pouze strom.

Vrcholy stromu můžeme rozdělovat do takzvaných *vrstev* (levels), kdy do *i-té vrstvy* patří všechny vrcholy, které jsou spojeny s kořenem přes právě i hran.

Příklad. Kdyby strom z obr. 1.27 měl kořen *1*, pak do první vrstvy patří vrcholy *{2, 3}*, do druhé vrstvy patří vrcholy *{4, 5, 6, 7}* apod.

Pojem stromu se dá popsat dalšími způsoby, my si vystačíme pouze s jedním.

Věta 4 (Charakterizace stromů)**.** *Pro graf G = (V, E) jsou následující podmínky ekvivalentní:*

1. *G je strom.*
2. *G je souvislý a |V| = |E| + 1.*

Důkaz. Implikace z jedničky do dvojky je zřejmá.
Opačnou implikaci provedeme indukcí podle počtu vrcholů. Nechť indukční předpoklad je, že pro každý strom G = (V, E) s n vrcholy platí |V| = |E| + 1.
Základní případ: pro strom s jedním vrcholem tvrzení platí, protože |E| + 1 = = 0 + 1 = 1.
Indukční krok: předpokládejme, že indukční předpoklad platí pro každý strom s n vrcholy a chceme dokázat, že platí i pro stromy s $(n + 1)$ vrcholy. Nechť v je

list stromu s $n + 1$ vrcholy. Odstraněním vrcholu v včetně jeho incidentní hrany dostaneme menší strom s n vrcholy, pro které tvrzení $|V| = |E| + 1$ platí. Nyní přidáme zpátky vrchol v s jeho incidentní hranou. Potom tvrzení $|V| = |E| + 1$ stále platí, protože počet vrcholů a počet hran se zvětšily o jedna.

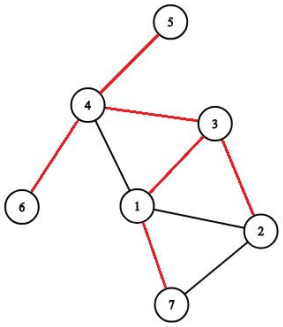
□

Poznámka. Stromy jsou bipartitní grafy.

1.4.1 Kostra grafu

Definice 26. *Nechť G = (V, E) je souvislý graf. Libovolný strom tvaru (V, E'), kde E' ⊆ E, nazveme* kostra *(spanning tree) grafu G.*

Příklad. Příklad kostry grafu je na obr. 1.28.



Obrázek 1.28: Kostra grafu (červené hrany)

Věta 5. *Každý souvislý graf G = (V, E) obsahuje kostru.*

Důkaz. Důkaz provedeme sporem. Nechť T = (V, E') je souvislý podgraf G s nejmenším počtem hran. Ukážeme, že T je acyklický. Předpokládejme, že T má cyklus: v_0–v_1, v_1–v_2, ..., v_n–v_0. Odstraňme poslední hranu cyklu v_n–v_0. Pokud libovolné dva vrcholy x a y byly spojeny cestou, která neobsahovala hranu v_n–v_0, tak zůstanou spojeny i po odebrání hrany. Naopak, pokud vrcholy x a y byly spojeny cestou, která obsahovala hranu v_n–v_0, tak zůstanou spojeny i po odstranění hrany díky cestě obsahující zbytek cyklu. Což je spor s tím, že T byl definovaný jako souvislý podgraf s nejmenším počtem hran. Proto T je acyklický.

□

Věta 6 (Cayleyho formule)**.** *Pro každé $n \geq 3$ je počet stromů na daných n vrcholech roven n^{n-2}.*

1.5 Chordální grafy

Definice 27. *Nechť G je graf. Řekneme, že G je* chordální *(chordal), pokud neobsahuje kružnici délky ≥ 4 jako indukovaný podgraf.*

Chordální grafy lze definovat i následujícím způsobem.

Definice 28. *Nechť G je graf. Řekneme, že G je chordální, pokud každá kružnice délky ≥ 4 obsahuje* tětivu *(chord), což je hrana, která spojuje dva nesousední vrcholy dané kružnice.*

Definice 29. *Nechť G je graf a x, y jsou dva nesousední vrcholy grafu G. Potom* x, y - řez *v G je množina vrcholů R grafu G taková, že x a y patří do různých komponent souvislosti po odstranění všech vrcholů R z G (označované $G - R$).*

Tvrzení 7. *Jestliže graf G je chordální, pak pro každé dva nesousední vrcholy x a y ($x \neq y$) v G existuje x, y - řez v G, který tvoří kliku.*

Důkaz. Důkaz provedeme sporem. Nechť G je chordální graf. Nechť x a y jsou dva nesousední vrcholy grafu G a nechť R je x, y - řez s nejmenším počtem vrcholů. Tvrdíme, že R tvoří kliku. Označme G_x a G_y komponenty $G - R$ obsahující x, resp. y. Pro spor předpokládejme, že R netvoří kliku, tedy R obsahuje dva vrcholy u a v, které nejsou spojeny hranou. Díky tomu, že R je co nejmenší, tak každý vrchol v řezu musí mít sousedy ve všech komponentách $G - R$. Tedy vrcholy u a v mají souseda jak v G_x, tak i v G_y. Nechť P_x je co nejkratší cesta z u do v, jejíž vnitřní vrcholy jsou v G_x a nechť P_y je co nejkratší cesta z u do v, jejíž vnitřní vrcholy jsou v G_y. Nechť $C = P_x \cup P_y$. Ukážeme, že C je indukovaná kružnice délky ≥ 4. Žádný vnitřní vrchol cesty P_x nemůže být totožný s vnitřním vrcholem cesty P_y, tudíž sjednocením dostaneme kružnici. Protože P_x a P_y musí mít alespoň jeden vrchol, tak se jedná o kružnici délky ≥ 4. Musíme ještě ověřit, že C je indukovaná kružnice v G. Díky předpokladu nevede mezi vrcholy u a v hrana. Z vnitřního vrcholu P_x nevede žádná hrana do vnitřního vrcholu P_y, neboť by R nebyl řez. A díky tomu, že P_x a P_y jsou co nejmenší, tak vnitřní vrcholy z P_x, resp. P_y nemohou být spojeny hranou, kromě těch hran, které patří do dané cesty, jinak bychom dostali kratší cestu. Tedy C je indukovaná kružnice délky ≥ 4, což je spor s tím, že G je chordální.

□

Poznámka. V tvrzení 7 platí dokonce ekvivalence.

Definice 30. *Nechť G je graf. Vrchol x v grafu G je* simpliciální vrchol *(simplicial vertex), pokud $N_G(x)$ tvoří kliku v G.*

Lemma 8. *Každý chordální graf G obsahuje simpliciální vrchol.*

Důkaz. Dokážeme, že každý chordální graf je buď úplný, nebo obsahuje dva nesousední simpliciální vrcholy.
Důkaz provedeme indukcí podle počtu vrcholů.
Základní případ: pro graf s jedním vrcholem tvrzení platí, neboť se jedná o úplný a chordální graf, který má jeden simpliciální vrchol.

Indukční krok: předpokládejme, že máme chordální graf G, který má aspoň dva vrcholy. Pokud G je úplný graf, tak jsme hotovi. Předpokládejme tedy, že G má dva nesousední vrcholy x a y. Díky tvrzení 7 víme, že existuje *x, y - řez R*, který v G tvoří kliku. Označme G_x (resp. G_y) komponentu $G - R$ obsahující x (resp. y). Navíc označme G_x^R (resp. G_y^R) podgraf G indukovaný $G_x \cup \mathrm{R}$ (resp. $G_y \cup \mathrm{R}$). Indukované podgrafy chordálního grafu jsou zase chordální grafy, tedy G_x^R a G_y^R jsou chordální. Z indukčního předpokladu víme, že G_x^R obsahuje simpliciální vrchol s_x nepatřící do R, neboť G_x^R je buď úplný, nebo má dva nesousední vrcholy. Pokud G_x^R je úplný, tak za simpliciální vrchol s_x volíme libovolný vrchol z G_x^R mimo R. Pokud G_x^R není úplný, tak za simpliciální vrchol s_x volíme ten ze dvou simpliciálních vrcholů, který nepatří do R. Analogicky pro G_y^R a simpliciální vrchol s_y. Protože s_x (resp. s_y) není v R, tak $N_{G_x^R}(s_x) = N_G(s_x)$ (resp. $N_{G_y^R}(s_y) = N_G(s_y)$). A tedy vrcholy s_x a s_y jsou nesousední simpliciální vrcholy grafu G.

□

Definice 31. *Nechť G = (V, E) je graf.* Perfektní eliminační schéma *(perfect elimination ordering) grafu G je posloupnost vrcholů* $(v_1, v_2, \ldots, v_n)$ *taková, že* $\forall i = 1, \ldots, n$ *platí, že sousedi* v_i *mezi vrcholy* $v_{i+1}, \ldots, v_n$ *tvoří kliku.*

Věta 9. *Graf G = (V, E) je chordální* $\iff$ *G má perfektní eliminační schéma.*

Důkaz. Implikaci zleva doprava dokážeme obměnou. Ukážeme tedy, že pokud G není chordální, tak nemůže mít perfektní eliminační schéma. Nechť G není chordální a nechť C je jeho libovolná indukovaná kružnice v grafu G délky ≥ 4. Pro každé uspořádání $v_1, v_2, \ldots, v_n$ vrcholů G platí, že nejlevější vrchol z C v této posloupnosti má dva nesousední pravé vrcholy (patřící do C). Tedy G nemá perfektní eliminační schéma.

K důkazu druhé implikace využijeme lemma 8. Nechť G je chordální graf, pak jeho perfektní eliminační schéma se zrekonstruuje následovně. Nechť v_1 je simpliciální vrchol grafu G, v_2 je simpliciální vrchol grafu $G - \{v_1\}$, v_3 je simpliciální vrchol grafu $G - \{v_1, v_2\}$ atd. Pak posloupnost vrcholů $(v_1, v_2, \ldots, v_n)$ je perfektní eliminační schéma grafu G.

□

1.6 Náhodný graf

Poznámka. Tato podkapitola je více než inspirována skripty *Erdős–Rényi random graphs*[3].

Pod *náhodným grafem* (random graph) si lze představit diskrétní graf, do kterého se náhodně přidají hrany.

Na vytváření náhodných grafů existuje celá řada modelů, my se zaměříme na běžně používaný model nazvaný *Erdős–Rényi model*.

1.6.1 Erdős–Rényi model

Definice 32. *Nechť $n \in \mathbb{N}$ a $0 \leq p \leq 1$, pak definujeme* Erdős–Rényiho náhodný graf *(ve zbytku textu pouze náhodný graf) G(n, p) jako graf s počtem vrcholů n, kde každé dva vrcholy jsou spojeny hranou s pravděpodobností p (nezávisle na ostatních hranách).*

Poznámka. Náhodným grafem se může rozumět i *G(n, m)*, kde se daný graf s n vrcholy a m hranami vybere náhodně z množiny grafů {G = (V, E), kde |V| = n a |E| = m}, kde každý graf má stejnou pravděpodobnost výběru.

Střední hodnota počtu hran náhodného grafu G(n, p) je $\binom{n}{2} \times p$.

Poznámka. Pravděpodobnost, že náhodný graf G(n, p) má k hran je dán binomickým rozdělením $B(\binom{n}{2}, p)$, kde nechť X je náhodná veličina popisující počet hran, pak

$$P(X = k) = \binom{\binom{n}{2}}{k} \times p^k \times (1 - p)^{\binom{n}{2}-k}, \text{ kde } 0 \leq k \leq \binom{n}{2}.$$

Střední hodnota stupně vrcholu v náhodném grafu G(n, p) je $(n - 1) \times p$.

Poznámka. Pravděpodobnost, že daný vrchol v náhodném grafu s n vrcholy má stupeň k je dán binomickým rozdělením $B(n - 1, p)$, kde nechť X_i je náhodná veličina popisující stupeň i-tého vrcholu, pak

$$P(X_i = k) = \binom{n - 1}{k} \times p^k \times (1 - p)^{k-1}, \text{ kde } 0 \leq k \leq n \text{ - } 1.$$

Poznamenejme, že dvě různé náhodné veličiny X_i a X_j nejsou úplně nezávislé (např. pokud $X_i = (n - 1)$, pak je zřejmé, že $X_j \neq 0$).

Definice 33. *Pravděpodobnost p_t nazveme* perkolační práh *(percolation threshold) pro nějakou vlastnost grafu, pokud pro všechny $p < p_t$ téměř všechny náhodné grafy G(n, p) nemají danou vlastnost a zároveň pokud pro všechny $p > p_t$ téměř všechny náhodné grafy G(n, p) mají danou vlastnost.*

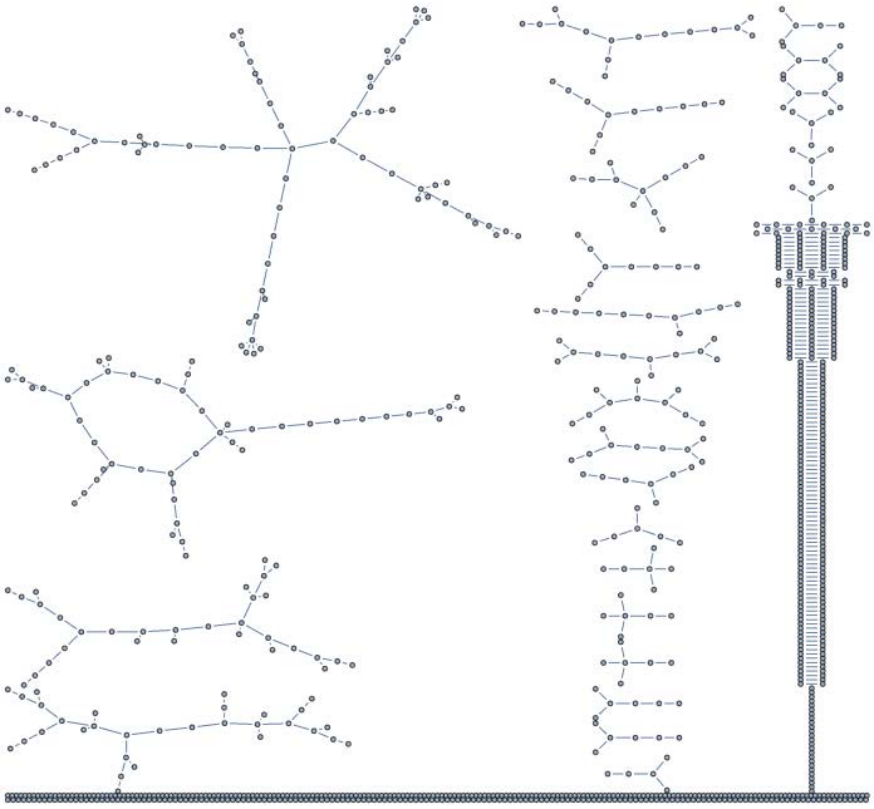
Souvislost náhodných grafů

Nyní uvedeme několik perkolačních prahů a jejich vliv na souvislost náhodného grafu.

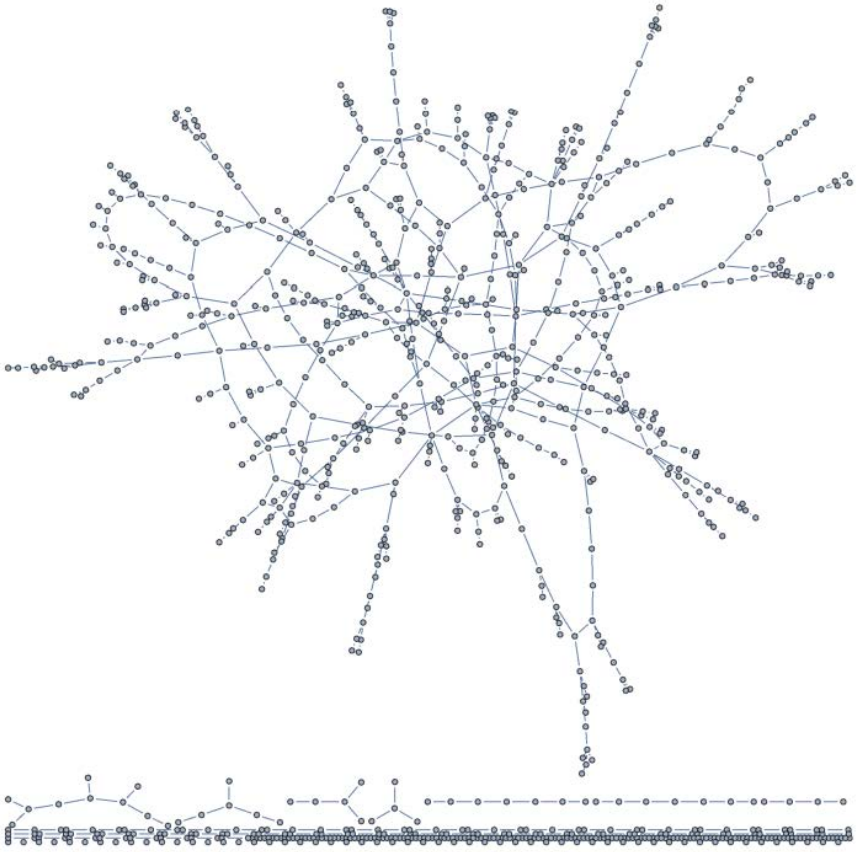
- p = 0, pak náhodný graf G(n, p) je diskrétní graf,

- p = 1, pak náhodný graf G(n, p) je úplný graf,
- p ∼ c/n, kde 0 < c < 1, pak komponenty náhodného grafu G(n, p) jsou buď stromy nebo komponenty obsahující právě jeden cyklus a střední hodnota počtu souvislých komponent je $n - p \times \binom{n}{2} + \mathcal{O}(1)$,
- p ∼ c/n, kde c > 1, pak náhodný graf G(n, p) obsahuje jednu obrovskou komponentu souvislosti a zbylé komponenty souvislosti jsou zpravidla stromy,
- p = c × log(n)/n, kde c ≥ 1, pak náhodný graf G(n, p) je souvislý a pro c = 1 náhodný graf obsahuje obrovskou komponentu souvislosti a izolované vrcholy.

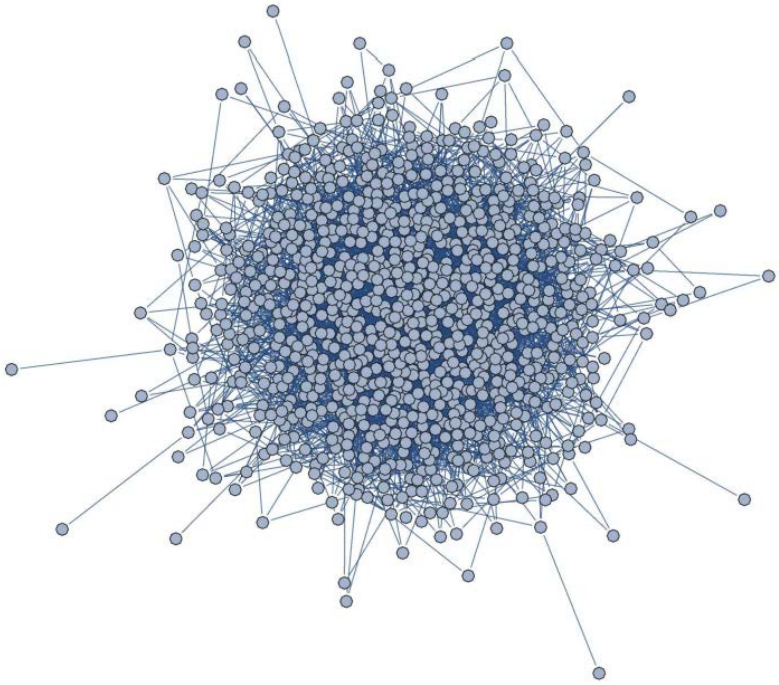
Příklad. Na obrázcích 1.29 - 1.31 jsou příklady náhodných grafů s pevným počtem vrcholů 1 000 a s různými pravděpodobnostmi p.



Obrázek 1.29: Příklad náhodného grafu G(1 000; 0,00095), kde p = $\frac{0{,}95}{n}$[3]



Obrázek 1.30: Příklad náhodného grafu G(1 000; 0,0015), kde p = $\frac{1,5}{n}$[3]



Obrázek 1.31: Příklad náhodného grafu G(1 000; 0,007), kde p = $2,\overline{33} \times \frac{log(n)}{n}$[3]

2. Barvení grafů

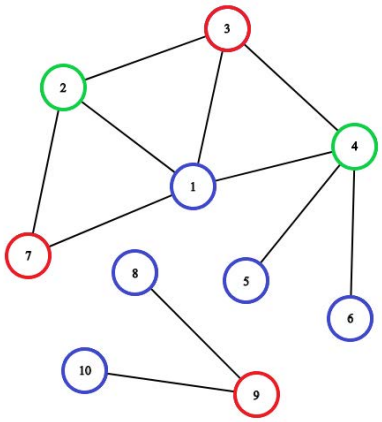
Nyní se zaměříme na takzvané barvení grafů. V grafu můžeme barvit dvě věci, a to vrcholy a hrany.

2.1 Vrcholové barvení grafů

Definice 34. Vrcholové obarvení *(vertex coloring) grafu* $G = (V, E)$ *je funkce* $c: V \to N$, *kde pro každou hranu* $\{x, y\} \in E$ *platí* $c(x) \neq c(y)$. *Graf* G *nazveme* k-obarvitelným *(k-colorable), pokud lze obarvit* k *barvami. Pod obarvením grafu si lze také představit rozdělení množiny vrcholů do disjunktních podmnožin* V_1, V_2, ..., V_k *takových, že žádné dva sousední vrcholy nejsou ve stejné množině,* $V_1 \cup V_2 \cup \ldots \cup V_k = V$ *a* $V_i \cap V_j = \emptyset$ *pro* $i \neq j$.

Definice 35. *Nechť* $G = (V, E)$ *je graf. Nejmenší* k, *pro které existuje* k*-obarvení grafu* G, *se nazývá chromatické číslo, které se značí* $\chi(G)$. *Každé obarvení grafu* G, *které požaduje* $\chi(G)$ *barev se nazývá optimální (chromatické) obarvení.*

Příklad. Příklad optimálního vrcholového obarvení grafu z obr. 1.1 je na obr. 2.1.



Obrázek 2.1: Příklad optimálního vrcholového obarvení grafu s $\chi(G) = 3$

Nalezení chromatického čísla pro libovolný graf je NP-úplný problém. Jinak řečeno, zatím se nezná polynomiální algoritmus pro nalezení chromatického čísla pro libovolný graf. Chromatické číslo umíme v polynomiálním čase určit jenom u některých typů grafů. Příkladem jsou:

1. bipartitní grafy, jejichž $\chi(G) = 2$,
2. cykly, které mají $\chi(G) = 2$ pro sudé cykly a $\chi(G) = 3$ pro liché cykly,
3. úplné grafy, jejichž $\chi(G) = \mathrm{n}$ a
4. chordální grafy.

Jedné z velkých tříd grafů, kterým umíme v polynomiálním čase zjistit chromatické číslo se říká *perfektní grafy* (perfect graph), pro které platí, že $\forall \mathrm{H} \leq \mathrm{G}$: $\chi(H) = \omega(H)$. Perfektními grafy jsou tedy mimo jiné i bipartitní grafy, chordální grafy, sudé cykly, úplné grafy apod.

Poznámka. Díky chromatickému číslu můžeme zavést alternativní definici pro bipartitní grafy. Graf G je *bipartitní* $\iff \chi(G) = 2$. Lze to dokonce rozšířit i pro k-partitní grafy. Graf G je *k-partitní* $\iff \chi(G) = \mathrm{k}$.

Definice 36. *V částečně obarveném grafu definujeme* nasycenost *(saturation degeree) vrcholu v, značeno jako $\rho(v)$, jako počet rozdílně obarvených sousedů vrcholu v.*

Lemma 10. *Každý souvislý graf, který není úplný, obsahuje dva nesousední vrcholy x a y, které mají společného souseda u.*

Důkaz. Vezmeme dva vrcholy z grafu, které nejsou spojeny hranou a najdeme mezi nimi nejkratší cestu P. Pokud $|\mathrm{P}| \leq 3$, tak krajní vrcholy cesty P jsou hledané vrcholy x, y a vnitřní vrchol cesty je u. Pokud $|\mathrm{P}| > 3$, tak na cestě P existují tři vrcholy x, y a u takové, že x je spojené hranou s u a y je spojené hranou s u, ale x není spojené hranou s y, neboť by se poté nejednalo o nejkratší cestu.

□

Lemma 11. *Nechť G je d-degenerovaný graf, pak $\chi(G) \leq (d + 1)$.*

Důkaz. Důkaz provedeme indukcí podle počtu vrcholů.
Základní případ: pro graf s nejvýše $d + 1$ vrcholy tvrzení triviálně platí.
Indukční krok: předpokládejme, že graf G má více než $d + 1$ vrcholů, pak nechť v je vrchol grafu G stupně nejvýše d. Z indukčního předpokladu víme, že $\mathrm{G} - \mathrm{v}$ lze obarvit $d + 1$ barvami. Sousedi vrcholu v používají nejvýše $deg_G(v) \leq d$ barev. Alespoň jedna barva z $d + 1$ barev je tedy nepoužitá a jednou z těchto barev obarvíme vrchol v. Graf G lze tedy obarvit $d + 1$ barvami.

□

Poznámka. Protože každý graf G je $\Delta(G)$-degenerovaný, tak $\chi(G) \leq \Delta(G) + 1$ pro libovolný graf.

Lemma 12. *Nechť G je souvislý graf obsahující aspoň jeden vrchol stupně menšího než $\Delta(G)$. Potom $\chi(G) \leq \Delta(G)$.*

Důkaz. Stačí dokázat, že takový graf G je $(\Delta(G) - 1)$-degenerovaný.
Nechť H je libovolný neprázdný podgraf grafu G. Tvrdíme, že $\delta(H) < \Delta(G)$. Pokud H obsahuje všechny vrcholy grafu G, tak jeho minimální stupeň je nejvýše $\Delta(G) - 1$, díky existenci vrcholu stupně menšího než $\Delta(G)$ v grafu G. Naopak, pokud H neobsahuje všechny vrcholy grafu G, tak díky souvislosti grafu G existuje hrana $\mathrm{e} = \{\mathrm{x}, \mathrm{y}\}$ taková, že $\mathrm{x} \in \mathrm{V(H)}$ a $\mathrm{y} \notin \mathrm{V(H)}$. Potom tedy $\delta(H) \leq deg_H(x) \leq deg_G(x) - 1 \leq \Delta(G) - 1$.

□

Věta 13 (Brooksova věta (Brooks' theorem))**.** *Nechť G je souvislý graf, který není úplný a není to ani lichá kružnice. Potom $\chi(G) \leq \Delta(G)$.*

Důkaz. [4] Nechť G = (V, E) je souvislý graf, který není úplný a není to ani lichá kružnice. Tvrdíme, že $\chi(G) \leq \Delta(G)$.
Pokud $\Delta(G) \leq 2$, tak G je buď sudá kružnice, nebo cesta, a tedy $\chi(G) \leq \Delta(G)$. Předpokládejme tedy, že $\Delta(G) \geq 3$. Nechť κ_G značí vrcholovou souvislost grafu G.

1. $\kappa_G = 1$.
   Nechť x je vrchol, po jehož odebrání se graf G rozpadne na dvě souvislé komponenty. Nechť G_1 označuje první souvislou komponentu společně s vrcholem x a G_2 označuje druhou souvislou komponentu společně s vrcholem x. Tedy $G_1 \cup G_2 = \mathrm{G}$ a $G_1 \cap G_2 = \{\mathrm{x}\}$. Víme, že $deg_{G_1}(x) < \Delta(G)$ a díky lemmatu 12 tedy $\chi(G_1) \leq \Delta(G)$. Analogicky pro graf G_2. Pokud vrchol x nemá stejnou barvu v obarveních grafů G_1 a G_2, tak stačí přepermutovat barvy v jednom z obarvení. Tedy G_1 a G_2 lze obarvit pomocí $\Delta(G)$ barev, kde vrchol x má v obou obarveních stejnou barvu a tedy sjednocením těchto dvou obarvení vznikne obarvení grafu G pomocí $\Delta(G)$ barev.

2. $\kappa_G = 2$.
   Nechť x a y jsou vrcholy, po jejichž odebrání se graf G rozpadne na dvě souvislé komponenty. Nechť G_1 označuje první souvislou komponentu společně s vrcholy x, y a G_2 označuje druhou souvislou komponentu společně s vrcholy x, y. Tedy $G_1 \cup G_2 = \mathrm{G}$ a $G_1 \cap G_2 = \{\mathrm{x}, \mathrm{y}\}$. Víme, že $deg_{G_1}(x) < \Delta(G)$ a $deg_{G_1}(y) < \Delta(G)$ a díky lemmatu 12 tedy $\chi(G_1) \leq \Delta(G)$. Analogicky pro graf G_2.

   (a) $b_1(x) = b_1(y)$ a $b_2(x) = b_2(y)$.
   Pak přepermutováním barvy v jednom z obarvení lze docílit toho, že vrcholy x a y budou mít stejnou barvu v obou obarveních.

   (b) $b_1(x) \neq b_1(y)$ a $b_2(x) \neq b_2(y)$.
   Pak sjednocení barev vrcholů x a y v obou obarveních lze zase vyřešit přepermutováním barev.

   (c) $b_1(x) = b_1(y)$ a $b_2(x) \neq b_2(y)$.

      i. $deg_{G_1}(\mathrm{x}) \leq \Delta(G) - 2$ nebo $deg_{G_1}(\mathrm{y}) \leq \Delta(G) - 2$.
      Pak přidáním hrany mezi vrcholy x a y v grafu G_1 a přebarvením grafu G_1 získáme obarvení grafu G_1 pomocí Δ(G) barev, kde vrcholy x a y mají různé barvy. Došlo tedy k převedení na předchozí případ. Analogicky pro $deg_{G_2}(\mathrm{x}) \leq \Delta(G) - 2$ nebo $deg_{G_2}(\mathrm{y}) \leq \Delta(G) - 2$.

      ii. $deg_{G_1}(\mathrm{x}) = deg_{G_1}(\mathrm{y}) = \Delta(G) - 1$.
      Pak $deg_{G_1}(\mathrm{x}) = deg_{G_1}(\mathrm{y}) = (\Delta(G) - 1)$ a $deg_{G_2}(\mathrm{x}) = deg_{G_2}(\mathrm{y}) = 1$. Změním obarvení grafu G_2, tak aby vrcholy x a y měly stejnou barvu. Toto lze udělat, neboť předpokládáme graf s $\Delta(\mathrm{G}) \geq 3$ a snažíme se tedy barvit pomocí Δ(G) barev. Tedy vrcholy x a y lze obarvit v grafu G_2 stejnou barvou, kterou nemají jejich dva sousedé. Došlo tedy k převedení na předchozí případ. Analogicky pro $deg_{G_2}(\mathrm{x}) = deg_{G_2}(\mathrm{y}) = \Delta(G) - 1$.

3. $\kappa_G \geq 3$.
   Protože graf G není úplný a je souvislý, tak v grafu existují dva vrcholy x

a y, které mají společného souseda z, díky lemmatu 10. Uspořádejme vrcholy grafu G do posloupnosti v_1 = x, v_2 = y, v_3, ..., v_n = z tak, že ∀i: v_i má alespoň jednoho souseda s větším indexem než i. Taková posloupnost existuje, jednou z možností, jak danou posloupnost vytvořit je například ta, že v_k, ..., v_{n-1} budou sousedi vrcholu z, v_l, ..., v_{k-1} budou sousedi sousedů vrcholů z atd. Následně tuto posloupnost můžeme hladově obarvit, což dokážeme pomocí Δ(G) barev. Vrcholům v_1 a v_2 bude přiřazena stejná barva. Ostatním vrcholům s indexem $i \geq 3$ bude přidělena barva nejvýše Δ(G), neboť daný vrchol má nejvýše $\Delta(G) - 1$ hran do vrcholů s menším indexem, které jsou obarveny nejvýše $\Delta(G) - 1$ barvami. Vrchol z, který má v posloupnosti nalevo všechny ostatní vrcholy grafu G (tedy $deg_G(z) \leq \Delta$(G)), lze také obarvit pomocí Δ(G) barvami, neboť alespoň dva jeho sousedi, konkrétně vrcholy x a y, mají stejnou barvu.

Tedy pro G platí, že $\chi(G) \leq \Delta(G)$.

□

2.1.1 Aplikace barvení grafů

Barvení grafů se využívá pro řešení řady problémů. Jednou z aplikací je plánování procesů, které nemohou nastat současně (rozvrh hodin, plánování zkoušek apod.).

Příklad (Plánování zkoušek). Naším cílem je naplánovat termíny jednotlivých zkoušek na zkouškové období, s co nejmenší dobou trvání tak, aby dvě zkoušky nebyly ve stejném časovém slotu, pokud nějaký student navštěvuje dva předměty s danými dvěma zkouškami.

Tento problém můžeme převést na graf, jehož vrcholy reprezentují jednotlivé zkoušky a dva vrcholy (zkoušky) jsou spojeny hranou, pokud dané zkoušky nemohou probíhat současně. Po obarvení daného grafu, s co nejmenším počtem barev, barvy vrcholů reprezentují časové sloty, kde vrcholy (zkoušky) se stejnou barvou budou probíhat současně.

Například rozvrhovací graf může vypadat jako graf z obr. 2.1. Pak časové sloty jednotlivých zkoušek můžou být například následovné:

- zkoušky obarvené modrou barvou (tj. *1, 5, 6, 8, 10*) budou probíhat v pondělí dopoledne,
- zkoušky obarvené červenou barvou (tj. *3, 7, 9*) budou probíhat v pondělí odpoledne a
- zkoušky obarvené zelenou barvou (tj. *2, 4*) budou probíhat v úterý dopoledne.

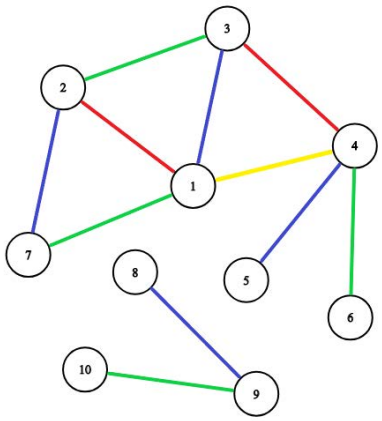
2.2 Hranové barvení grafů

Neformálně, hranové obarvení přiřazuje barvy hranám, a to takovým způsobem, že dvě hrany incidentní se stejným vrcholem, mají různé barvy.

Definice 37. Hranové obarvení *(edge coloring) grafu G = (V, E) je funkce c: E → N, kde pro dvě různé hrany* e_1, $e_2 \in E$ *incidentní se stejným vrcholem platí* $c(e_1) \neq c(e_2)$. *Graf G nazveme* hranově k-obarvitelným *(k-edge-colorable), pokud lze hranově obarvit k barvami.*

Definice 38. *Nechť G je graf. Nejmenší k, pro které existuje hranové k-obarvení grafu G, se nazývá* chromatický index *(chromatic index), který se značí* $\chi'(G)$.

Příklad. Příklad optimálního hranového obarvení grafu z obr. 1.1 je na obr. 2.2.



Obrázek 2.2: Příklad optimálního hranového obarvení grafu s $\chi'(G) = 4$

Poznámka. Díky tomu, že hrany vycházející z jednoho vrcholu musí mít různé barvy, nám dává dolní odhad chromatického indexu $\chi'(G) \geq \Delta(G)$.

Poznámka. Díky předchozí poznámce je vidět, že obarvení grafu na obr. 2.2 je vážně optimální. Neboť $\Delta(G) = \chi'(G)$.

Hranové obarvení lze převést na vrcholové obarvení, a to takovým způsobem, že daný graf G, který chceme hranově obarvit, převedeme na hranový graf L(G), který vrcholově obarvíme. Poté barva hrany v grafu G odpovídá barvě příslušného vrcholu v grafu L(G). Platí tedy $\chi'(G) = \chi(L(G))$.

Věta 14 (Vizingova věta (Vizing's theorem))**.** *Pro každý graf G = (V, E) platí* $\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1$.

Poznámka. Grafům splňující $\Delta(G) = \chi'(G)$ se říká grafy Vizingovy třídy 1 a grafům splňující $\Delta(G) + 1 = \chi'(G)$ se říká grafy Vizingovy třídy 2.

Poznámka. Je NP-úplné rozhodnout, zda daný graf má Vizingovu třídu 1.

3. Algoritmy

Jak již bylo zmíněno v kapitole 2.1, nezná se optimální algoritmus s polynomiální časovou složitostí pro získání optimálního obarvení libovolného grafu. Následující algoritmy tedy nezaručují optimalitu pro všechny grafy, ale zpravidla zaručují optimalitu pro několik specifických tříd grafů.

Poznámka. Terminologie, popisy jednotlivých algoritmů a příslušné SHC a HC grafy pro jednotlivé algoritmy byly získány z publikace *Classical Coloring of Graphs*[2]. Všechny důkazy v této kapitole jsou mým dílem, neboť výše zmíněná publikace neobsahuje žádné důkazy a tyto důkazy se mi nepodařilo najít ani v jiných publikacích.

Poznámka. Nechť G je graf a A je algoritmus. Pak A(G) označuje počet barev použitých při obarvení grafu G algoritmem A.

Definice 39. *Pro algoritmus A definujeme* performance guarantee *A(n) následovně: A(n) = max{A(G)/χ(G): G je graf s n vrcholy}.*

Poznámka. Performance guarantee A(n) udává horní odhad "špatnosti" algoritmu A pro grafy s počtem vrcholů n. Performance guarantee tedy nepopisuje chování algoritmu v průměrném případě.

Definice 40. *Graf G nazveme* slightly hard-to-color *pro algoritmus A, pokud existuje alespoň jedna implementace algoritmu A, která obarví graf G neoptimálně.*

Definice 41. *Pro algoritmus A označujeme* SHC(A) *množinu všech nejmenších (vzhledem k počtu vrcholů a hran) slightly hard-to-color grafů pro algoritmus A.*

Definice 42. *Graf G nazveme* hard-to-color *pro algoritmus A, pokud všechny implementace algoritmu A obarví graf G neoptimálně.*

Definice 43. *Pro algoritmus A označujeme* HC(A) *množinu všech nejmenších (vzhledem k počtu vrcholů a hran) hard-to-color grafů pro algoritmus A.*

Algoritmy se porovnávají podle dvou kritérií. Prvním kritériem je časová složitost algoritmu a druhým kritériem je performance guarantee. Obě kritéria nám říkají, co můžeme nejhoršího očekávat od algoritmu pro počet vrcholů n $\longrightarrow \infty$. Pro grafy s menším počtem vrcholů Hansen a Kuplinsky v 90. letech představili koncept hard-to-color grafů. Díky znalosti hard-to-color grafů lze příslušný algoritmus vylepšit tak, abychom u něho co nejvíce omezili hard-to-color grafy. Hard-to-color grafy jsou tedy další možností, jak porovnávat dva algoritmy.

3.1 Hladové obarvení

Hladové obarvení (greedy coloring) je jednou z poměrně častých metod, jak obarvit vrcholy daného grafu. Nechť G je graf a K = (v_1, ..., v_n) je uspořádaná

posloupnost jeho vrcholů. Hladové obarvení grafu G spočívá v tom, že se prochází postupně vrcholy posloupnosti K a obarvují se co nejmenší možnou barvou.

Algorithm 1: hladové obarvení

```
1 Greedy-Color (G, K)
2 |  foreach v :- v_1 to v_n do
3 |  |  obarvi vrchol v nejmenší možnou barvou;
4 |  end
```

Časová složitost obarvení grafu G = (V, E) pomocí hladového algoritmu je $\mathcal{O}(|V| + |E|)$.

3.2 Optimální obarvení

Definice 44. *Nechť G je graf a $K = (v_1, \ldots, v_n)$ je posloupnost jeho vrcholů. Posloupnost K nazveme* optimální posloupností *(optimal sequence), pokud hladový algoritmus daný graf G obarví pomocí posloupnosti K optimálně.*

Věta 15. *Pro každý graf existuje optimální posloupnost.*

Důkaz. Důkaz provedeme konstruktivně, tj. sestrojíme optimální posloupnost. Nechť G je graf, který je optimálně obarven, a jehož $\chi(G) = d$. Množinu všech vrcholů grafu G obarvených barvou i, kde $1 \leq i \leq d$, označme S_i, kde $|S_i| = k_i$. Pak optimální posloupností grafu G je posloupnost K = $(v_1^1, v_1^2, \ldots, v_1^{k_1}, \ldots, v_i^1, v_i^2, \ldots, v_i^{k_i}, \ldots, v_d^1, v_d^2, \ldots, v_d^{k_d})$.
Ukážeme, že hladový algoritmus obarví graf G pomocí posloupnosti K optimálně.

- Začneme hladově obarvovat všechny vrcholy z množiny S_1. Dané vrcholy budou obarveny nejmenší možnou barvou, tj. barvou *1*.
- Předpokládejme, že jsme obarvili všechny vrcholy z množin $S_1, \ldots, S_{i-1}$ a chceme hladově obarvit vrcholy z množiny S_i. Pro každý vrchol $v \in S_i$ mohou nastat právě dvě situace:
  1. vrchol v bude obarven jednou z barev *1, ...,* $(i-1)$,
  2. pokud vrchol v nemůže být obarven ani jednou z barev *1, ...,* $(i-1)$, pak určitě může být obarven barvou i, protože jinak by v S_i existoval vrchol u, který by byl spojený hranou s vrcholem v, což nastat nemůže, neboť by v původním (optimálním) obarvení nemohly mít stejnou barvu.

V každé množině S_i, kde $1 \leq i \leq d$, existuje alespoň jeden vrchol v, který bude obarven barvou i, jinak by graf bylo možné obarvit méně než d barvami, což by bylo ve sporu s jeho původním (optimálním) obarvením. Na hladové obarvení pomocí posloupnosti K jsme využili právě d barev.

□

Poznámka. Optimální posloupnost pro daný graf nemusí být jednoznačně určena.

3.2.1 Barvení chordálních grafů

Ukážeme, že chordální grafy umíme obarvit optimálně v polynomiálním čase.

Tvrzení 16. *Nechť G je chordální graf, pak jeho perfektní eliminační schéma je optimální posloupností.*

Důkaz. Nechť G je chordální graf a $(v_1, \ldots, v_n)$ je jeho perfektní eliminační schéma. Nechť Predchudci(v) označuje počet sousedů vrcholu v, které se nacházejí v perfektním eliminačním schématu před vrcholem v (tzn. že mají menší index). Nechť v_s je vrchol, pro který platí

$$Predchudci(v_s) = \max_{i = 1,\ldots,n} Predchudci(v_i).$$

Potom $\omega(G) \geq$ Predchudci(v_s) + 1 a tedy $\chi(G) \geq$ Predchudci(v_s) + 1. Z definice hladového obarvení plyne, že $\chi(G) \leq$ Predchudci(v_s) + 1, z čehož nakonec plyne $\chi(G) =$ Predchudci(v_s) + 1.

□

Poznámka. Perfektní eliminační schéma pro libovolný chordální graf jsme schopni nalézt v polynomiálním čase (viz kapitola 6.2).

3.3 Sekvenční algoritmy

Sekvenční algoritmus (sequential algorithm) na obarvení grafu G se skládá z následujících dvou kroků:

1. vytvoření posloupnosti vrcholů K = $(v_1, \ldots, v_n)$,
2. hladového obarvení grafu G pomocí posloupnosti K - *Greedy-Color(G, K).*

Sekvenční algoritmy se tedy liší tím, jakým způsobem vytvářejí danou posloupnost vrcholů K.

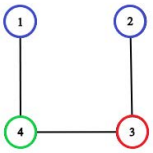
3.3.1 Random sequence metoda

Random sequence metoda (RS metoda), často označovaná jako naivní metoda, vytváří posloupnost K zcela náhodně.

Algorithm 2: random sequence metoda

```
1 RandomSequenceMethod (G)
2 |  K :- náhodná posloupnost vrcholů grafu G;
3 |  Greedy-Color(G, K);
```

Nejmenší SHC graf pro tuto metodu je P_4, kde příslušná posloupnost vrcholů je zobrazena na obr. 3.1. Díky větě 15 neexistuje HC graf pro tento algoritmus.



Obrázek 3.1: Nejmenší slighty hard-to-color graf pro RS metodu

RS metodu lze implementovat s časovou složitostí $\mathcal{O}(|V| + |E|)$.

3.3.2 Largest first metoda

Largest first metoda (LF metoda) je jednou z nejstarších a nejjednodušších sekvenčních algoritmů, který vymyslel Welsh a Powell v roce 1967. Vychází z pozorování, že vrcholy s větším stupněm by se měly obarvit dříve něž vrcholy s menším stupněm.

Algorithm 3: largest first metoda

```
1 LargestFirstMethod (G)
2 |  K :- neklesající posloupnost vrcholů seřazených podle jejich stupňů;
3 |  Greedy-Color(G, K);
```

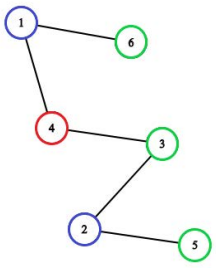
LF metoda je optimální pro následující grafy: liché kružnice, sudá kola a hvězdy.

Tvrzení 17. *LF metoda obarví S_n optimálně.*

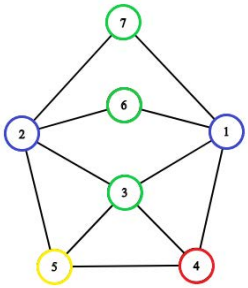
Důkaz. Předpokládejme, že S_n má ≤ 2 vrcholy, pak hladové obarvení pro libovolnou posloupnost K obarví graf G optimálně.
Předpokládejme tedy, že n > 2, pak S_n má $(n - 1)$ vrcholů stupně 1 a jeden vrchol v_1 stupně $(n - 1)$. Do posloupnosti K bude jako první prvek přidán vrchol v_1 a následně v libovolném pořadí zbytek vrcholů. Hladový algoritmus obarví vrchol v_1 barvou *1* a zbytek vrcholů barvou *2*. Jedná se o optimální obarvení, neboť S_n je bipartitní graf a tedy $\chi(S_n) = 2$.

□

Hansen a Kuplinsky v roce 1991 ukázali, že nejmenší SHC graf pro tuto metodu je P_6, zobrazený na obr. 3.2. Nejmenší HC graf pro LF metodu je znám pod označením *obálka* (envelope), který je zobrazen na obr. 3.3.



Obrázek 3.2: Nejmenší slighty hard-to-color graf pro LF metodu



Obrázek 3.3: Nejmenší hard-to-color graf pro LF metodu

LF metodu lze implementovat s časovou složitostí $\mathcal{O}(|V| + |E|)$.

3.3.3 Smallest last metoda

Smallest last metoda (SL metoda), kterou vymyslel Matula a kolektiv v roce 1972, obdobně jako LF metoda, je založena na myšlence, že obarvení vrcholů s menším stupněm by se mělo odkládat co nejdéle.

Algorithm 4: smallest last metoda

```
1 SmallestLastMethod (G)
2 |  K :- ∅
3 |  while V \ K ≠ ∅ do
4 |  |  přidat nakonec posloupnosti K vrchol v s nejmenším stupněm
  |  |    v podgrafu generovaný vrcholy V \ K
5 |  end
6 |  Greedy-Color(G, K̄); // kde K̄ značí inverzní posloupnost k K
```

SL metoda je optimální pro následující grafy: stromy, cykly, kola, úplné bipartitní grafy a Mycielského grafy. Navíc tato metoda na obarvení d-degenerovaného grafu použije nejvýše $(d + 1)$ barev.

Tvrzení 18. *SL metoda obarví P_n optimálně.*

Důkaz. Nechť P_n je graf s vrcholy $v_1, \ldots, v_n$. Pak vrcholy v_1 a v_n mají stupeň *1* a všechny ostatní vrcholy mají stupeň *2*. BÚNO předpokládejme, že vrchol v_1 je vložen do posloupnosti K jako první vrchol. Podgraf $G - v_1$ je P_{n-1} a postupujeme analogicky, tj. do posloupnosti K vložíme vrchol v_2 jako druhý vrchol atd. Tedy $K = (v_1, \ldots, v_n)$ a $\bar{K} = (v_n, \ldots, v_1)$. Hladový algoritmus tedy použije dvě barvy na obarvení P_n pomocí posloupnosti K, což je optimální obarvení.

□

Tvrzení 19. *SL metoda obarví C_n optimálně.*

Důkaz. V C_n mají všechny vrcholy stejný stupeň 2. Nechť v_i je vrchol, který byl jako první vložen do posloupnosti K. Pak $G - v_i$ je P_{n-1} a díky tvrzení 18 víme, že bude obarven SL metodou pomocí 2 barev. V posloupnosti $\bar{K}$ jsou tedy obarveny všechny vrcholy kromě posledního, tj. vrcholu v_i. Pokud C_n je sudá kružnice, tak vrcholy v_{i-1} a v_{i+1} jsou obarveny stejnou barvou a vrchol v_i bude obarven opačnou barvou. Pokud C_n je lichá kružnice, tak vrcholy v_{i-1} a v_{i+1}

jsou obarveny různou barvou a tedy vrchol v_i bude obarven třetí (novou) barvou. Pro sudou kružnici (resp. lichou kružnici) se jedná o optimální obarvení, neboť $\chi(C_n) = 2$ (resp. $\chi(C_n) = 3$).

□

Tvrzení 20. *SL metoda obarví S_n optimálně.*

Důkaz. Graf S_n má $(n - 1)$ vrcholů stupně 1 a jeden vrchol v_1 stupně $(n - 1)$. Vrchol v_1 bude vložen do posloupnosti K jako poslední, nebo předposlední prvek. BÚNO předpokládejme, že vrchol v_1 byl vložen jako poslední. Tedy $\bar{K} = (v_1, \ldots, v_n)$. Hladový algoritmus obarví vrchol v_1 barvou *1* a všechny ostatní vrcholy obarví barvou *2*. Jedná se o optimální obarvení, neboť S_n je bipartitní graf a jeho $\chi(G) = 2$.

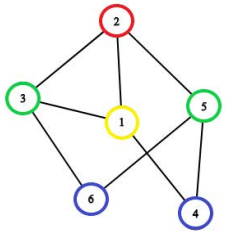
□

Tvrzení 21. *SL metoda obarví W_n optimálně.*

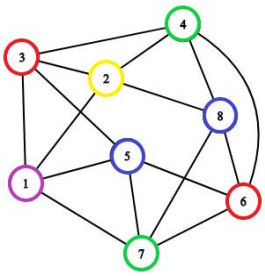
Důkaz. Předpokládejme, že W_n má ≤ 4 vrcholy, pak hladové obarvení pro libovolnou posloupnost K obarví graf optimálně.
Předpokládejme tedy, že n > 4, pak W_n má $(n - 1)$ vrcholů stupně 3 a jeden vrchol v_1 stupně $(n - 1)$. BÚNO předpokládejme, že vrchol v_2 byl jako první vložen do posloupnosti K. Pak v grafu G $- v_2$ se snížily stupně u tří vrcholů, které byly sousedy vrcholu v_2, tj. v_1 a nechť zbylé dva vrcholy označíme v_3 a v_n. BÚNO předpokládejme, že z vrcholů v_3 a v_n je do posloupnosti K vložen v_3 jako druhý vrchol a takto postupně přidáme do posloupnosti K vrcholy v_2, …, v_n. Jako poslední vrchol se přidá do posloupnosti K vrchol v_1 (vrchol v_1 může být vložen do posloupnosti K jako předpředposlední, předposlední, nebo poslední prvek, my předpokládáme, že je přidán jako poslední prvek, což neubírá nic na obecnosti). Tedy K = $(v_2, \ldots, v_n, v_1)$ a $\bar{K} = (v_1, v_n, \ldots, v_2)$. Hladový algoritmus obarví vrchol v_1 barvou *1*. Indukovaný podgraf vrcholů $v_n, \ldots, v_2$ tvoří kružnici C_{n-1} a díky tvrzení 19 víme, že vrcholy budou obarveny 2 barvami (resp. 3 barvami), pokud n je liché (resp. sudé). SL metoda použije na obarvení W_n 3 barvy (resp. 4 barvy), pokud n je liché (resp. sudé), což je optimální obarvení W_n, neboť $\chi(W_n) = 3$ (resp. $\chi(W_n) = 4$) pro liché n (resp. sudé n).

□

Kubale v roce 1997 ukázal, že nejmenší SHC graf pro tuto metodu je *prism* graf zobrazený na obr. 3.4. Nejmenší HC graf pro SL metodu je znám pod označením *prismatoid*, který je zobrazen na obr. 3.5.



Obrázek 3.4: Nejmenší slighty hard-to-color graf pro SL metodu



Obrázek 3.5: Nejmenší hard-to-color graf pro SL metodu

SL metodu lze implementovat s časovou složitostí $\mathcal{O}(|V| + |E|)$.

3.3.4 Connected sequential metoda

Connected sequential metoda (CS metoda) je založena na myšlence, že v každé iteraci algoritmu se za kandidáty na obarvení berou ty vrcholy, které mají už nějakého obarveného souseda.

Algorithm 5: connected sequential metoda

```
1 ConnectedSequentialMethod (G)
2 |  K :- posloupnost vrcholů grafu G taková, že každý vrchol (kromě
  |     prvního) má alespoň jednoho souseda s menším indexem v
  |     posloupnosti.
3 |  Greedy-Color(G, K);
```

Poznámka. Posloupnost K lze vytvořit například průchodem do šířky (BFS) nebo průchodem do hloubky (DFS).

CS metoda je optimální pro následující grafy: liché kružnice a bipartitní grafy.

Tvrzení 22. *CS metoda obarví lichou kružnici C_n optimálně.*

Důkaz. Nechť C_n je lichá kružnice. BÚNO předpokládejme, že vrchol v_1 byl jako první vložen do posloupnosti K. Nechť v_2 a v_n jsou sousedi vrcholu v_1. BÚNO předpokládejme, že z vrcholů v_2 a v_n je do posloupnosti K vložen v_2 jako druhý vrchol a takto postupně přidáváme do posloupnosti K vrcholy $v_1, \ldots, v_n$. Tedy K $= (v_1, \ldots, v_n)$. Hladový algoritmus bude vrcholům $v_1, \ldots, v_{n-1}$ přiřazovat barvy *1* a *2* na přeskáčku. Při obarvování vrcholu v_n, má vrchol obarvený sousedy dvěma barvami, a to barvou *1* a *2*, tedy vrcholu v_n bude přiřazena barva *3*, což je optimální obarvení.

□

Tvrzení 23. *CS metoda obarví bipartitní graf optimálně.*

Důkaz. Nechť G je bipartitní graf. Z lemmatu 2 víme, že graf G neobsahuje kružnici liché délky. Všimněme si, že při vytváření posloupnosti K $= (v_1, \ldots, v_n)$ vlastně vytváříme kostru grafu (= strom s kořenem v_1). Algoritmus tedy vrcholy v sudé vrstvě stromu obarví barvou *1* a vrcholy v liché vrstvě stromu obarví barvou *2*, takže každá hrana ve stromu bude incidentní s jedním vrcholem barvy *1* a s jedním vrcholem barvy *2*. Stačí jenom ukázat, že neexistují dva vrcholy ve

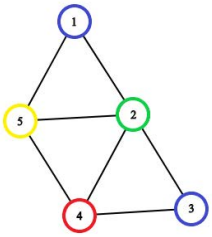
stromě, které jsou oba v sudé (resp. liché) vrstvě a jsou spojeny hranou v grafu G. Důkaz provedeme sporem, nechť tedy vrcholy v_i a v_j (i $\neq$ j) jsou v sudé vrstvě a jsou spojeny hranou.

- Pokud vrcholy v_i a v_j leží na stejné cestě z kořene v daném stromě, pak díky tomu, že oba vrcholy leží v sudé vrstvě, tak délka cesty z v_i do v_j je sudá. Když navíc vezmeme v potaz hranu v grafu G mezi vrcholy v_i a v_j, tak vznikne lichá kružnice, což je spor s tím, že graf je bipartitní.
- Pokud vrcholy v_i a v_j neleží na stejné cestě z kořene v daném stromě, tak nechť v_k je nejmenší (v rámci vrstvy) společný předchůdce. BÚNO předpokládejme, že vrchol v_k leží také v sudé vrstvě. Pak délky cest ve stromě mezi vrcholy v_i a v_k a vrcholy v_j a v_k jsou sudé. Když navíc vezmeme v potaz hranu v grafu G mezi vrcholy v_i a v_j, tak vznikne lichá kružnice, což je ve sporu s bipartitností grafu.

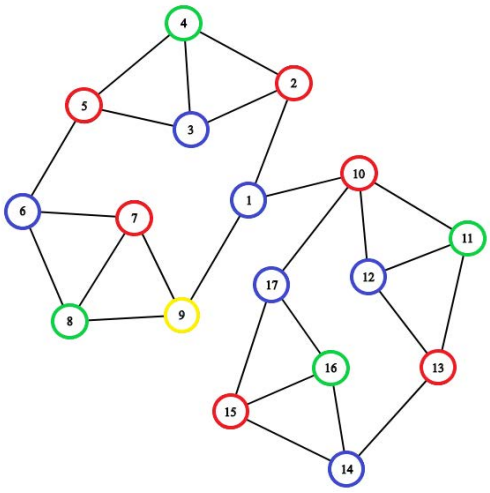
Analogicky pro vrcholy v_i a v_j, které leží v liché vrstvě.

□

Nejmenší SHC graf pro tuto metodu je F_5 zobrazený na obr. 3.6. Nejmenší HC graf pro CS metodu je znám pod označením *twin dragon*, který je zobrazen na obr. 3.7.



Obrázek 3.6: Nejmenší slighty hard-to-color graf pro CS metodu



Obrázek 3.7: Nejmenší hard-to-color graf pro CS metodu

CS metodu lze implementovat s časovou složitostí $\mathcal{O}(|V| + |E|)$.

3.3.5 Interchange metoda

Interchange je jednoduchá metoda pro rozšíření libovolného sekvenčního algoritmu, jehož cílem je omezit vytváření nových barev. Pokud je potřeba přiřadit vrcholu v novou barvu, tak je snaha prohodit barvy v bichromartickém podgrafu tak, aby se u vrcholu v uvolnila jedna barva.

Algorithm 6: interchange

```
1 Interchange (G, K)
2   foreach v :- v_1 to v_n do
3     if v potřebuje novou barvu then
4       snaha o prohození barev v bichromatickém podgrafu G tak, aby
          se uvolnila jedna barva pro vrchol v;
5     obarvit vrchol v nejmenší možnou barvou;
6   end
```

Jak ale zjistit, zda takový bichromatický podgraf existuje a jak ho najít? Nechť G je částečně obarvený graf pomocí barev $c_1, \ldots, c_k$ a nechť při pokusu o obarvení vrcholu v došlo k zavolání metody interchange, tzn. že sousedi vrcholu v používají všechny barvy $c_1, \ldots, c_k$. Pro dvě barvy c_i *a* c_j (i < j) nageneruje me bichromatický podgraf následovně. Nejdříve do bichromatického podgrafu přidáme všechny sousedy vrcholu v s barvou c_i. Poté pro každý vrchol s barvou c_i (resp. c_j) v bichromatickém podgrafu přidáme všechny jeho sousedy s barvou c_j (resp. c_i) a tento postup opakujeme, dokud už nelze žádný nový vrchol přidat. Nagenerovali jsme tedy bichromatický podgraf, který obsahuje vrcholy s barvou c_i a c_j.

- Pokud v bichromatickém podgrafu se nevyskytuje vrchol s barvou c_j, který je zároveň sousedem vrcholu v, pak všem vrcholům v bichromatickém podgrafu můžeme prohodit barvy z c_i na c_j, resp. z c_j na c_i, díky čemuž se vrcholu v uvolní barva c_i, pomocí které může být obarven.

- Pokud naopak v bichromatickém podgrafu se vyskytuje vrchol s barvou c_j, který je zároveň sousedem vrcholu v, pak nelze prohodit barvy v bichromatickém podgrafu tak, aby se uvolnila nějaká barva pro vrchol v.

Takto procházíme všechny bichromatické podgrafy pro každé dvě barvy, dokud nenajdeme nějaký bichromatický podgraf, ve kterém lze prohodit barvy. Pokud žádný takový bichromatický podgraf nenajdeme, tak vrcholu v přiřadíme novou barvu c_{k+1}.

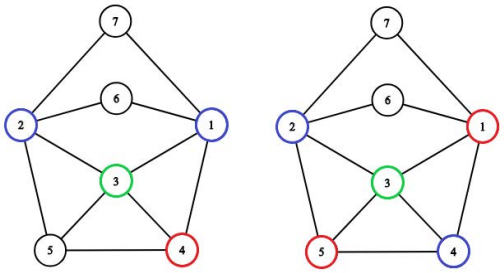
Příklad. Příklad metody interchange je ukázán na obr. 3.8, kde je částečně obarvený graf (levý graf) a cílem je obarvit vrchol *5*. Protože na obarvení vrcholu *5* je potřeba použít novou barvu, tak se pokusíme najít nějaký bichromatický podgraf, ve kterém lze prohodit barvy tak, aby se uvolnila nějaká barva pro vrchol *5*.

- Pro *zelenou* a *modrou* barvu bichromatický podgraf obsahuje vrcholy {*1, 2, 3*}. Protože vrchol *2* s modrou barvou je sousedem vrcholu *5*, tak v tomto bichromatickém podgrafu nelze prohodit barvy.

- Pro *zelenou* a *červenou* barvu bichromatický podgraf obsahuje vrcholy {*3, 4*}. Protože vrchol *4* s červenou barvou je sousedem vrcholu *5*, tak ani v tomto bichromatickém podgrafu nelze prohodit barvy.

- Pro *červenou* a *modrou* barvu bichromatický podgraf obsahuje vrcholy {*4, 1*}. Protože neexistuje vrchol v bichromatickém podgrafu s modrou barvou, který je sousedem vrcholu *5*, tak lze v tomto bichromatickém podgrafu prohodit barvy. Vrchol *4* lze přebarvit na *modrou* barvu a vrchol *1* lze přebarvit na *červenou* barvu, čímž se uvolní *červená* barva pro vrchol *5*.

Výsledný přebarvený graf s obarveným vrcholem *5* je znázorněn vpravo.



Obrázek 3.8: Příklad metody interchange při obarvování vrcholu *5*

Protože každá hrana je incidentní s nejvýše dvěma obarvenými vrcholy a každý vrchol se může vyskytovat v nejvýše b bichromatických podgrafech, kde b je počet barev grafu, tak časová složitost jednoho volání interchange je $\mathcal{O}(|V|^2)$, neboť b $\leq$ |V|.

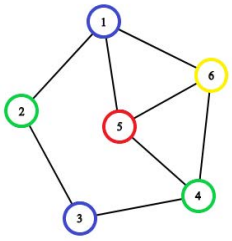
Random sequence interchange metoda

Random sequence interchange metoda (RSI metoda) je RS metoda rozšířená metodou interchange.

Algorithm 7: random sequence interchange metoda

```
1 RandomSequenceInterchangeMethod (G)
2 |  K :- náhodná posloupnost vrcholů grafu G;
3 |  Interchange(G, K);
```

Nejmenší SHC graf pro tuto metodu je zobrazen na obr. 3.9. Díky větě 15 neexistuje HC graf pro tento algoritmus.



Obrázek 3.9: Nejmenší slighty hard-to-color graf pro RSI metodu

RSI metodu lze implementovat s časovou složitostí $\mathcal{O}(|V|^3)$.

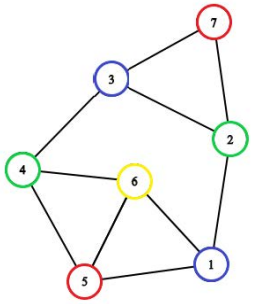
Largest first interchange metoda

Largest first interchange metoda (LFI metoda) je LF metoda rozšířená metodou interchange.

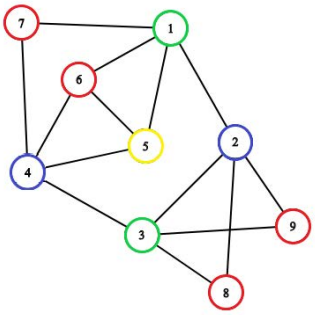
Algorithm 8: largest first interchange metoda

```
1 LargestFirstInterchangeMethod (G)
2 |  K :- neklesající posloupnost vrcholů seřazených podle jejich stupňů;
3 |  Interchange(G, K);
```

Nejmenší SHC graf pro tuto metodu je zobrazen na obr. 3.10. Nejmenší HC graf pro LFI metodu je zobrazen na obr. 3.11.



Obrázek 3.10: Nejmenší slighty hard-to-color graf pro LFI metodu



Obrázek 3.11: Nejmenší hard-to-color graf pro LFI metodu

LFI metodu lze implementovat s časovou složitostí $\mathcal{O}(|V|^3)$.

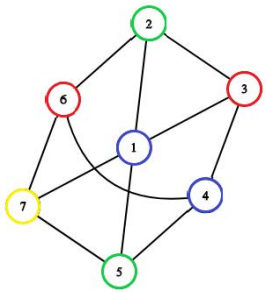
Smallest last interchange metoda

Smallest last interchange metoda (SLI metoda) je SL metoda rozšířená metodou interchange.

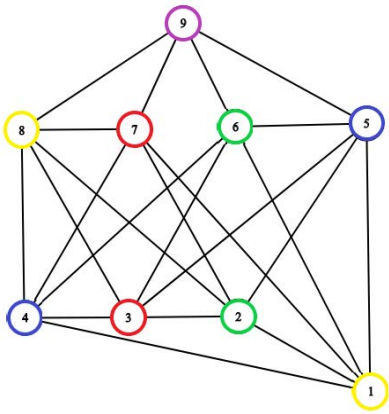
Algorithm 9: smallest last interchange metoda

```
1 SmallestLastInterchangeMethod (G)
2 |  K :- ∅
3 |  while V \ K ≠ ∅ do
4 |  |  přidat nakonec posloupnosti K vrchol v s nejmenším stupněm
  |  |    v podgrafu generovaný vrcholy V \ K
5 |  end
6 |  Interchange(G, K̄); // kde K̄ značí inverzní posloupnost k K
```

Nejmenší SHC graf pro tuto metodu je zobrazen na obr. 3.12. Nejmenší HC graf pro SLI metodu je zobrazen na obr. 3.13.



Obrázek 3.12: Nejmenší slighty hard-to-color graf pro SLI metodu



Obrázek 3.13: Nejmenší hard-to-color graf pro SLI metodu

SLI metodu lze implementovat s časovou složitostí $\mathcal{O}(|V|^3)$.

3.4 Saturation largest first metoda

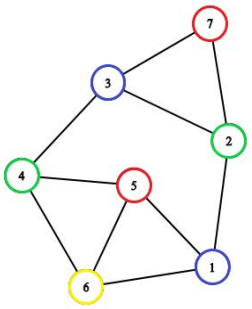
Saturation largest first metoda (SLF / DSATUR metoda), kterou vymyslel Brélaz v roce 1979, je upravená LF metoda, která nejdříve obarvuje vrcholy s větší nasyceností.

Algorithm 10: saturation largest first metoda

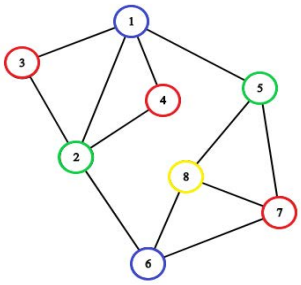
```
1 SaturationLargestFirstMethod (G)
2     while nejsou obarveny všechny vrcholy grafu G do
3         v :- vrchol s největší nasyceností ρ(v) v grafu G;
4         obarvi vrchol v nejmenší možnou barvou;
5     end
```

SLF metoda je optimální pro následující grafy: bipartitní grafy a kaktusy.

Nejmenší SHC graf pro tuto metodu je zobrazen na obr. 3.14. Nejmenší HC graf pro SLF metodu je zobrazen na obr. 3.15.



Obrázek 3.14: Nejmenší slighty hard-to-color graf pro SLF metodu



Obrázek 3.15: Nejmenší hard-to-color graf pro SLF metodu

SLF metodu lze implementovat s časovou složitostí $\mathcal{O}((|V| + |E|) \times log(|V|))$.

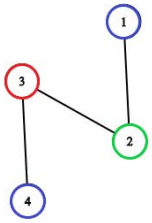
3.5 Greedy independent sets metoda

Greedy independent sets metoda (GIS metoda), kterou vymyslel Johnson v roce 1974, je založena na myšlence, že v *i-té* iteraci algoritmu se obarví co největší počet vrcholů barvou *i*.

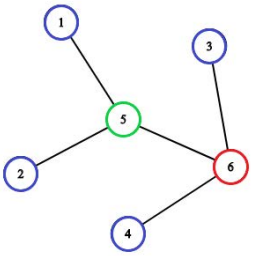
Algorithm 11: greedy independent sets metoda

```
1  GreedyIndependentSetsMethod (G)
2      c :- 1;
3      while nejsou obarveny všechny vrcholy grafu G do
4          available :- množina neobarvených vrcholů grafu G;
5          while available ≠ ∅ do
6              v :- vrchol s nejmenším stupněm v podgrafu generovaný vrcholy
                z množiny available;
7              obarvi vrchol v barvou c;
8              odstraň vrchol v a jeho sousedy z množiny available;
9          end
10         c := c + 1;
11     end
```

Nejmenší SHC graf pro tuto metodu je P_4 zobrazený na obr. 3.16. Nejmenší HC graf pro GIS metodu je znám pod označením *double star*, který je zobrazen na obr. 3.17.



Obrázek 3.16: Nejmenší slighty hard-to-color graf pro GIS metodu



Obrázek 3.17: Nejmenší hard-to-color graf pro GIS metodu

GIS metodu lze implementovat s časovou složitostí $\mathcal{O}(|V| \times |E|)$.

3.6 Kombinační metoda

Kombinační metoda (combination method) kombinuje LF a SLF metodu, kde v liché iteraci jsou k obarvení vybírány vrcholy LF metodou a v sudé iteraci jsou k obarvení vybírány vrcholy SLF metodou.

Algorithm 12: kombinační metoda

```
1  CombinationMethod (G)
2      c :- 1;
3      while nejsou obarveny všechny vrcholy grafu G do
4          if c je liché then
5              v :- neobarvený vrchol s největším stupněm v grafu G;
6          else
7              v :- vrchol s největší nasyceností ρ(v) v grafu G;
8          end
9          obarvi vrchol v nejmenší možnou barvou;
10         c :- c + 1;
11     end
```

Tuto metodu lze implementovat s časovou složitostí $\mathcal{O}((|V| + |E|) \times log(|V|))$.

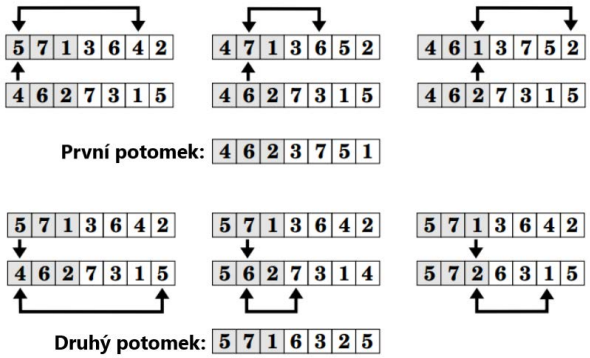
3.7 Genetický algoritmus

Genetický algoritmus (genetic algorithm) je technika založená na přírodní evoluci. V přírodě, schopnější jedinci mají větší šanci na přežití, a proto další generace jedinců by měla být ještě schopnější, protože byli zplozeni ze schopnějších rodičů. Obdobné myšlenky se využívá i v genetickém algoritmu.

Algoritmus začíná s náhodnou množinou populací velikosti p, kde jedincem populace se rozumí posloupnost vrcholů grafu. Následně jsou vybrány dva rodiče k vytvoření dvou nových potomků. Výběr rodičů je proveden na základě fitovací funkce, která ohodnocuje "kvalitu" jedince, v tomto případě je jediným kritériem počet barev použitých u hladového obarvení dané posloupnosti vrcholů. Poté dojde ještě ke genetické mutaci nově vzniklého potomka. Takto vytvoříme novou populaci velikosti p. Toto provádíme iterativně.

Ke *křížení jedinců* (crossover) lze například využít metodu *partially matched crossover* (PMX). Nejdříve se náhodně vygeneruje index v posloupnosti, tzv. *cut point*. První potomek se vytvoří tak, že se z druhého rodiče zkopírují všechny prvky do cut pointu a z prvního rodiče se zkopírují všechny prvky od cut pointu s tím, že se musí ještě provádět příslušné prohazování prvků, aby nedocházelo k vytváření nevalidních jedinců. Druhý potomek se vytvoří obdobně s tím, že se nejdříve kopíruje z prvního rodiče a následně z druhého rodiče.

Příklad. Příklad křížení dvou rodičů je znázorněn na obr. 3.18.



Obrázek 3.18: Příklad křížení dvou jedinců, kde jedinec je reprezentován posloupností vrcholů

Mutace (mutation) jsou operace, které nějakým způsobem mění daného jedince, v tomto případě tedy mění posloupnost vrcholů. Používáme 3 typy mutací:

1. *náhodné prohození* (random swap) vybere náhodně dva indexy v posloupnosti a jejich prvky zamění,

2. *prohození sousedů* (adjacent swap) vybere náhodně jeden index v posloupnosti a zamění daný prvek s jeho sousedem,

3. *invertování podposloupnosti* (inverted exchange) vybere náhodně dva indexy v posloupnosti a podposloupnost ohraničenou těmito dvěma indexy zinvertuje.

Algorithm 13: genetický algoritmus

```
1  GeneticAlgorithm (G)
2      P :- populace, tj. p náhodných permutací vrcholů grafu G;
3      k :- počet iterací algoritmu;
4      for i ← 0 to k do
5          P_new := ∅;
6          for j ← 0 to |P|/2 do
7              p_1, p_2 :- vybrat dva jedince z populace P na základě fitovací
                funkce;
8              o_1, o_2 :- Crossover(p_1, p_2);
9              AdjacentSwap(o_1);
10             AdjacentSwap(o_2);
11             RandomSwap(o_1);
12             RandomSwap(o_2);
13             InvertedExchange(o_1);
14             InvertedExchange(o_2);
15             P_new := P_new + o_1;
16             P_new := P_new + o_2;
17         end
18         P :- P_new;
19         K :- doposud nejlepší jedinec;
20     end
21     Greedy-Color(G, K);
```

Poznámka. Genetický algoritmus s velikostí populace *10* a počtem iterací n^1 (resp. n^2), označovaný jako *genetický algoritmus s exponentem 1* (resp. *genetický algoritmus s exponentem 2)*, budeme označovat *Gen1* (resp. *Gen2).*

Časová složitost obarvení grafu G = (V, E) pomocí genetického algoritmu je $\mathcal{O}(k \times p \times (p \times |V| + |E|))$, kde p je velikost populace a k je počet iterací algoritmu.

3.8 Connected largest first metoda

Connected largest first metoda (CLF metoda), kterou vymyslel Illner v roce 2019, je upravená LF metoda zkombinovaná s CS metodou.

Algorithm 14: connected largest first metoda

```
1  ConnectedLargestFirstMethod (G)
2      v :- vrchol s maximálním stupněm, pokud vrcholů s maximálním
         stupněm je více, tak se z nich vybere libovolný vrchol s největším
         počtem sousedů sousedů;
3      vlož vrchol v (s počtem jeho sousedů) do prioritní fronty
         priorityQueue;
4      while nejsou obarveny všechny vrcholy grafu G do
5          v :- prvek z priorityQueue s největším počtem sousedů;
6          vlož do priorityQueue všechny sousedy (s počtem jejich sousedů)
             vrcholu v, pokud nejsou již obarveny a nenacházejí se
             v priorityQueue;
7          if na obarvení vrcholu v není potřeba nové barvy then
8              obarvi vrchol v barvou, která je nejpoužívanější mezi jeho
                 sousedy sousedů;
9          else
10             obarvi vrchol v novou barvou;
11         end
12     end
```

CLF metoda je optimální pro následující grafy: liché kružnice, sudá kola a bipartitní grafy.

Tvrzení 24. *CLF metoda obarví lichou kružnici C_n optimálně.*

Důkaz. Nechť C_n je lichá kružnice. Protože všechny vrcholy mají stejný stupeň a stejný počet sousedů sousedů, tak metoda může vybrat libovolný první vrchol k obarvení. BÚNO předpokládejme, že metoda vybrala vrchol v_1. Vrcholu v_1 bude přiřazena barva *1* a do prioritní fronty budou přidáni sousedi vrcholu v_1, kde nechť v_2 a v_n jsou sousedi vrcholu v_1. BÚNO předpokládejme, že v druhé iteraci algoritmu bude vybrán vrchol v_2 k obarvení. Vrcholu v_2 bude přiřazena barva *2*. Analogicky pokračujeme pro vrcholy $v_3, \ldots, v_{n-1}$. Při obarvení vrcholu v_n, jeho sousedi mají všechny doposud použité barvy, tj. barvu *1* a *2*, a proto vrcholu v_n bude přiřazena nová barva *3*, což je optimální obarvení.

□

Tvrzení 25. *CLF metoda obarví bipartitní graf optimálně.*

Důkaz. Obdobný důkaz jako u tvrzení 23.

□

Obarvení grafu G = (V, E) pomocí CLF metody lze implementovat s časovou složitostí $\mathcal{O}(|V|^2)$.

3.9 Rozšíření metody interchange

Interchange extended, kterou vymyslel Illner v roce 2019, je rozšíření standardní metody interchange. Cílem je zeslabit poměrně silnou podmínku pro prohození barev v bichromatickém podgrafu.

Algorithm 15: Rozšíření metody interchange

```
1  InterchangeExtended (G, v)
2      foreach pro každou dvojici barev c_i a c_j (i < j) do
3          vytvoř bichromatický podgraf s barvami c_i a c_j;
4          if v bichromatickém podgrafu lze prohodit barvy then
5              prohoď barvy v bichromatickém podgrafu;
6              obarvi vrchol v barvou c_i;
7              return;
8          else
9              if neexistují sousedi vrcholu v s barvou c_i a c_j, které jsou v
                bichromatickém podgrafu spojeny hranou then
10                 foreach artikulaci v_a v bichromatickém podgrafu takovou, že
                    všichni sousedé vrcholu v s barvou c_i jsou v jedné
                    komponentě souvislosti a všichni sousedé vrcholu v s barvou
                    c_j, které jsou v bichromatickém podgrafu, jsou v jiných
                    komponerntách souvislosti do
11                     if artikulaci v_a lze přebarvit na jinou barvu než c_i a c_j
                        then
12                         přebarvi artikulaci v_a na jinou barvu než c_i a c_j;
13                         přehoď barvy v komponentě bichromatického
                            podgrafu obsahující sousedy vrcholu v s barvou c_i.
14                         obarvi vrchol v barvou c_i;
15                         return;
16                 end
17         end
18     end
19     obarvi vrchol v novou barvou;
```

Nechť G je částečně obarvený graf pomocí barev $c_1, \ldots, c_k$ a nechť při pokusu o obarvení vrcholu v došlo k zavolání metody interchangeExtended, tzn. že sousedi vrcholu v používají všechny barvy $c_1, \ldots, c_k$. Pro dvě barvy c_i a c_j (i < j) nagenerujeme bichromatický podgraf. Pokud v bichromatickém podgrafu lze přehodit barvy, tak je prohodíme a vrchol v obarvíme barvou c_i. Pokud barvy přehodit nejdou, tak víme, že existuje alespoň jeden soused vrcholu v s barvou c_j, který je v bichromatickém podgrafu. Naším cílem je v bichromatickém podgrafu najít takovou artikulaci v_a, po jehož odstranění z bichromatického podgrafu, by se bichromatický podgraf rozpadl na několik komponent souvislosti takových, že v jedné z nich (označme k) by byli všichni sousedé vrcholu v s barvou c_i a žádný soused vrcholu v s barvou c_j, který je v bichromatickém podgrafu. Pokud by takový vrchol v_a šlo přebarvit na jinou barvou než c_i a c_j, pak by se po prohození barev všech vrcholů v komponentě k uvolnila barva c_i pro vrchol v. Všimněme si, že nemá cenu hledat artikulace v bichromatickém podgrafu, pokud nějaký soused vrcholu v s barvou c_i je spojen hranou s některým ze sousedů vrcholu v s barvou c_j, neboť nějaké artikulace by mohly existovat, ale určitě neexistuje artikulace, která by v případných komponentách souvislosti odfiltrovala sousedy vrcholu v s barvou c_i od sousedů vrcholu v s barvou c_j, které jsou v bichromatickém podgrafu. Takto procházíme všechny bichromatické podgrafy pro každé dvě barvy, dokud nenajdeme buď nějaký bichromatický podgraf, ve kterém lze přeho-

dit barvy, nebo nenalezneme artikulaci, kterou lze přebarvit a po jejíž odstranění se bichromatický graf "hezky" rozpadne.

Nyní se zaměříme na přebarvení případné artikulace v_a v nějakém bichromatickém podgrafu. Naivním přístupem může být rozdíl dvou množin, kde první množina obsahuje všechny použité barvy v grafu a v druhé množině jsou barvy sousedů vrcholu v_a obohaceny barvou vrcholu v_a. Pokud výsledná množina je neprázdná, tak lze vrchol v_a přebarvit, v opačném případě nikoliv. Jinak řečeno, zaměříme se pouze na daný vrchol v_a .Na problém přebarvení artikulace se může pohlížet i z větší perspektivy, kdy je snahou nejdříve přebarvit artikulaci a pokud se toto nedaří, tak je snaha přebarvit všechny jeho sousedy, které mají stejnou barvu, díky čemuž se uvolní barva pro artikulaci v_a. U případného přebarvení sousedů na barvy obsažené v bichromatickém grafu se musí dávat zřetel na to, zda se po přebarvení sousedů a případném prohození barev v dané komponentě souvislosti bude jednat o validní obarvení.

Algorithm 16: pokus o přebarvení artikulace

```
1  TryRecolorCutVertex (G, v_a, recolor)
2      if TryRecolorVertex(G, v_a, recolor) then
3          return true;
4      else
5          C_n :- množina barev sousedů vrcholu v_a;
6          foreach barva c_i z C_n do
7              canBeRecolored :- true;
8              foreach soused n_i vrcholu v_a s barvou c_i do
9                  if !TryRecolorVertex(G, n_i, false) then
10                     canBeRecolored :- false;
11                     break;
12             end
13             if canBeRecolored then
14                 if recolor then
15                     foreach soused n_i vrcholu v_a s barvou c_i do
16                         TryRecolorVertex(G, n_i, true)
17                     end
18                     obarvi vrchol v_a barvou c_i;
19                 return true;
20         end
21     end
22     return false;
```

Algorithm 17: pokus o přebarvení vrcholu

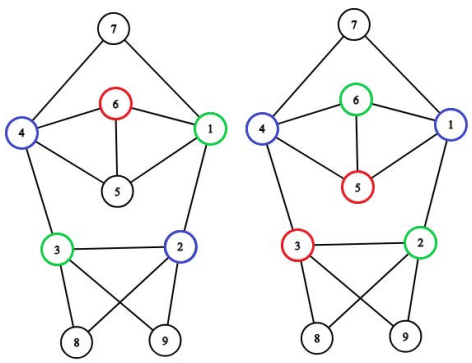
```
1  TryRecolorVertex (G, v, recolor)
2      C_g :- množina všech použitých barev v částečně obarveném grafu G;
3      C_n :- množina barev sousedů vrcholu v, včetně barvy vrcholu v;
4      C :- C_g ∩ C_n;
5      if C = ∅ then
6          return false;
7      else
8          if recolor then
9              obarvi vrchol v barvou z C;
10         return true;
11     end
```

Příklad. Příklad rozšířené metody interchange je ukázán na obr. 3.19, kde je částečně obarvený graf (levý graf) a cílem je obarvit vrchol *5* na jehož obarvení je potřeba nové barvy.

1. Pro *modrou* a *červenou* barvu bichromatický podgraf obsahuje vrcholy {*4, 6*}. Protože vrchol *6* s *červenou* barvou je sousedem vrcholu *5*, tak v tomto bichromatickém podgrafu nelze prohodit barvy, tedy standardní interchange selhal. Protože existuje soused vrcholu *5* s *modrou* barvou (*4*), který je spojen hranou se sousedem vrcholu *5* s *červenou* barvou (*6*), tak nemá význam hledat artikulace, tedy i rozšířený interchange v tomto případě selhal.

2. Pro *modrou* a *zelenou* barvu bichromatický podgraf obsahuje vrcholy {*4, 3, 2, 1*}. Protože vrchol *1* se *zelenou* barvou je sousedem vrcholu *5*, tak nelze použít standardní interchange ani v tomto bichromatickém podgrafu. Zkusíme tedy problém vyřešit pomocí rozšířeného interchange. Poněvadž neexistuje soused vrcholu *5* s *modrou* barvou, který je spojen hranou s nějakým sousedem vrcholu *5* se *zelenou* barvou, tak má smysl hledat potencionální artikulace bichromatického podgrafu, které jsou {*3, 2*}. Artikulaci *3* lze přebarvit na *červenou* barvu. Po přehození barev v první komponentě souvislosti, tj. v komponentě s vrcholy {*1, 2*}, se uvolní *červená* barva pro vrchol *5*.

Výsledný přebarvený graf s obarveným vrcholem *5* je znázorněn vpravo.



Obrázek 3.19: Příklad rozšířené metody interchange při obarvování vrcholu *5*

Bohužel i přes toto rozšíření v některých případech selže i rozšířený interchange, tzn. že danému vrcholu přiřadí novou barvu, i když to není potřeba. Naštěstí některé z případů lze vyřešit pomocí následujícího rozšíření. Cílem je najít speciální indukovaný podgraf K_3, ve kterém by se po přebarvení jeho vrcholů uvolnila barva pro daný vrchol.

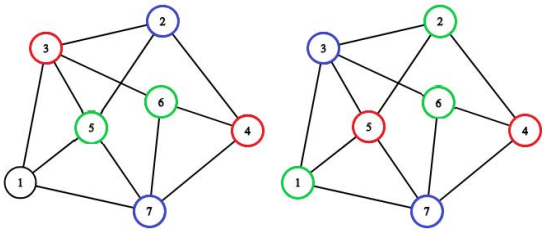
Nechť G je částečně obarvený graf pomocí barev $c_1, \ldots, c_k$ a nechť při pokusu o obarvení vrcholu v došlo k zavolání metody interchangeExtended, tzn. že sousedi vrcholu v používají všechny barvy $c_1, \ldots, c_k$. Nechť máme nagenerovaný bichromatický podgraf pro barvy c_i a c_j (i < j) a nechť vrchol v má právě jednoho souseda s barvou c_i (označme v_1) a nechť existuje nějaký vrchol, který je sousedem vrcholu v s barvou c_j, který je spojem hranou s vrcholem v_1, množinu všech těchto vrcholů si označíme N_2. Tedy v této situaci se nemůže uplatnit standardní ani rozšířená metoda interchange. Cílem je najít dva sousedy vrcholu v_1, které jsou spojeny navzájem hranou, ale nejsou spojeny hranou s libovolným vrcholem z množiny N_2 a po jejichž přebarvení by se uvolnila barva pro vrchol v. Analogicky pokud vrchol v má právě jednoho souseda s barvou c_j.

Příklad. Příklad rozšířené metody interchange s přebarvením K3 je ukázán na obr. 3.20, kde je částečně obarvený graf (levý graf) a cílem je obarvit vrchol *1*, na jehož obarvení je potřeba nové barvy.

1. Pro *zelenou* a *červenou* barvu bichromatický podgraf obsahuje vrcholy {*3, 4, 5, 6*}. Protože vrchol *3* s červenou barvou je sousedem vrcholu *1*, tak v tomto bichromatickém podgrafu nelze použít standardní metodu interchange. Protože vrchol *5* se *zelenou* barvou a vrchol *3* s *červenou* barvou jsou spojeny hranou a jedná se o sousedy vrcholu *1*, tak nelze použít ani rozšířenou metodu interchange. Přestože počet sousedů vrcholu *1* se *zelenou* (příp. *červenou*) barvou je jedna, tak nemůžeme použít rozšíření pomocí přebarvení K3, protože neexistuje žádný takový indukovaný podgraf K3, neboť vrchol *3* má 4 sousedy, z nichž vrcholy *1* a *5* nelze použít a zbylé dva sousedi nejsou spojeni hranou. Pro vrchol *5* obdobně.

2. Pro *modrou* a *zelenou* barvu bichromatický podgraf obsahuje vrcholy {*2, 5, 6, 7*}. Protože vrchol *5* se *zelenou* barvou je sousedem vrcholu *1*, tak ani v tomto bichromatickém podgrafu nelze použít standardní metodu interchange. Protože vrchol *7* s *modrou* barvou a vrchol *5* se *zelenou* barvou jsou spojeny hranou a jedná se o sousedy vrcholu *1*, tak ani v tomto bichromatickém podgrafu nelze použít rozšířenou metodu interchange. Neboť počet sousedů vrcholu *1* s *modrou* (příp. *zelenou*) barvou je jedna, tak má smysl hledat indukované podgrafy K3. Pro vrchol *5* připadají v potaz pouze dva jeho sousedi, a to vrcholy *2* a *3*, neboť vrcholy *1* a *7* nejsou povoleny. V tomto indukovaném podgrafu K3 lze přebarvit vrcholy tak, aby se uvolnila nějaká barva pro vrchol *1*.

Výsledný přebarvený graf s obarveným vrcholem *1* je znázorněn vpravo. Nechváme na rozmyšlení čtenáři, že pro všechny bichoromatické podgrafy by standardní a rozšířená metoda interchange selhala.



Obrázek 3.20: Příklad rozšířené metody interchange s rozšířením K3 při obarvování vrcholu *1*

Poznámka. Rozšířenou metodou interchange (interchangeExtended) budeme ve zbytku textu rozumět metodu, která pro každý bichromatický podgraf upřednostňuje standardní interchange před rozšířeným interchangem, který pro přebarvení artikulace využívá i přebarvení jeho sousedů, a nebere v potaz rozšíření s přebarvením K3. Navíc, pokud artikulace nebo jeho soused může být přebarven pomocí více barev, tak se vybírá ta barva, která je nejpoužívanější mezi jeho sousedy sousedů.

Rozšířenou metodou interchange s přebarvením K3 (interchangeExtendedK3) budeme ve zbytku textu rozumět rozšířenou metodu interchage s přebarvením indukovaného podgrafu K3, kde k případnému přebarvení indukovaného podgrafu K3 nastane pouze, pokud pro každý bichromatický podgraf standardní a rozšířená metoda interchange selže. Důvod malé priority přebarvení indukovaného podgrafu K3 vychází z experimentů, z kterých vyplynulo, že je lepší upřednostňovat standardní a rozšířený interchange před přebarvením K3.

Poznámka. RS metodu používající rozšířenou metodu interchange (příp. rozšířený interchange s přebarvením K3) budeme označovat *RSIE* (příp. *RSIEK3*).

LF metodu používající rozšířenou metodu interchange (příp. rozšířený interchange s přebarvením K3) budeme označovat *LFIE* (příp. *LFIEK3*).

SL metodu používající rozšířenou metodu interchange (příp. rozšířený interchange s přebarvením K3) budeme označovat *SLIE* (příp. *SLIEK3*).

CLF metodu používající rozšířenou metodu interchange (příp. rozšířený interchange s přebarvením K3) budeme označovat *CLFIE* (příp. *CLFIEK3*).

Nyní spočítáme časovou složitost pro jedno zavolání rozšířené metody interchange z algoritmu 15. Řádky *2* až *7* jsou stejné jako u standardního interchange a tedy mají stejnou časovou složitost $\mathcal{O}(|V|^2)$. Vyhodnotit podmínku z řádku *9* zvládneme pro všechny bichromatické podgrafy v $\mathcal{O}(\Delta^2(G))$. Časová složitost pro nalezení všech artikulací, kterých může být až $b \times |V|$, kde b je počet barev grafu, ve všech bichromatických podgrafech je stejná jako nalezení samotných bichromatických podgrafů, tedy $\mathcal{O}(|V|^2)$. Nyní se zaměříme na to, s jakou časovou složitostí jsme schopni zjistit, zda případná artikulace "hezky rozbije" bichromatický podgraf. Nejdříve musíme z bichromatického podgrafu odebrat danou artikulaci a její incidentní hrany, což zvládneme v čase $\mathcal{O}(\Delta(G))$. Získání komponent souvislosti v bichromatickém podgrafu zvládneme v čase $\mathcal{O}(V + E)$. Zjištění, zda dané obarvené vrcholy spadají do správných komponent zvládneme v konstantním čase a případné přidání artikulace zpět do bichromatického podgrafu spolu s jejími incidentními hranami zvládneme zase v čase $\mathcal{O}(\Delta(G))$. Neboť máme nejvýše $b * |V|$ potencionálních artikulací, tak zjištění, zda daná artikulace "hezky

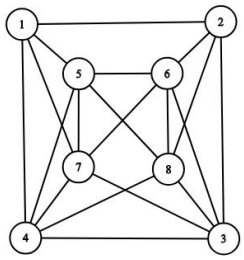
rozbije" graf, zvládneme pro všechny artikulace ve všech bichromatických podgrafech v $\mathcal{O}(b \times |V| \times (\Delta(G) + |V| + |E| + \Delta(G))$. Poslední, co nám zbývá odhadnout, je časová složitost na zjištění, zda jdou dané artikulace, příp. jejich sousedi přebarvit. Pro každý vrchol si můžeme vytvořit seznam možných barev pro přebarvení s časovou složitostí $\mathcal{O}(|V| \times \Delta(G) + |V| \times (\Delta(G) + b))$. Následné zjištění, zda danou artikulaci dokážeme přebarvit, umíme v konstantním čase a případné zjištění, zda sousedy artikulace umíme přebarvit, dokážeme v čase $\mathcal{O}(\Delta(G))$. Celková časová složitost jednoho zavolání rozšířené metody interchange je tedy $\mathcal{O}(|V|^3)$.

Časová složitost rozšířeného interchange s přebarvením K3 má stejnou časovou složitost jako rozšířený interchange s tím, že se musí započítat hledání případných K3 podgrafů včetně ověření, zda jdou přebarvit. Získání všech vhodných K3 podgrafů zvládneme v čase $\mathcal{O}(\Delta^2(G))$. Zjištění, zda jeden K3 podgraf dokážeme přebarvit, zvládneme v čase $\mathcal{O}(b^2)$. Celková časová složitost jednoho zavolání rozšířené metody interchange s přebarvením K3 je tedy $\mathcal{O}(|V|^4)$.

3.10 Benchmark a weak benchmark

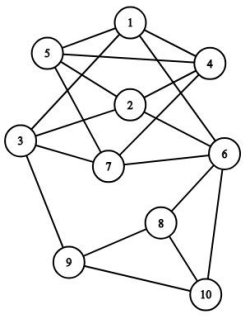
Definice 45. *Nechť $A = (A_1, \ldots, A_k)$ je množina algoritmů. Řekněme, že graf G je* benchmark *pro algoritmy z množiny A, pokud G je HC pro všechny algoritmy A_i, $i = 1, \ldots, k$. Řekněme, že graf G je* weak benchmark *pro algoritmy z množiny A, pokud G je SHC pro všechny algoritmy A_i, $i = 1, \ldots, k$.*

Benchmark pro algoritmy *LF* a *SL* je zobrazen na obr. 3.21.



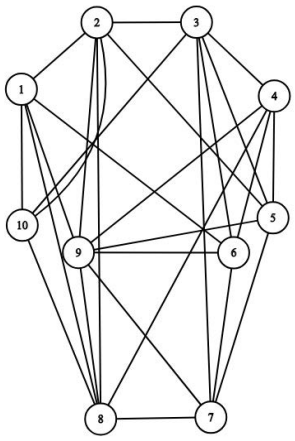
Obrázek 3.21: Benchmark pro algoritmy *LF* a *SL*

Benchmark pro algoritmy *LF*, *SL* a *SLF* je zobrazen na obr. 3.22.



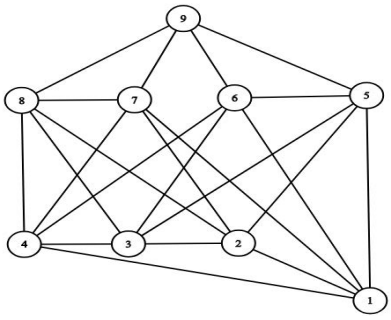
Obrázek 3.22: Benchmark pro algoritmy *LF*, *SL* a *SLF*

Benchmark pro algoritmy *LF*, *SL*, *SLF* a *GIS* je zobrazen na obr. 3.23.



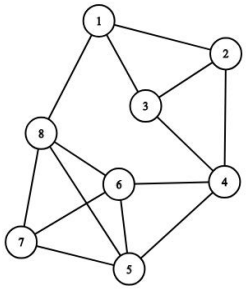
Obrázek 3.23: Benchmark pro algoritmy *LF*, *SL*, *SLF* a *GIS*

Benchmark pro algoritmy *LF*, *SL*, *SLF*, *LFI* a *SLI* je zobrazen na obr. 3.24.



Obrázek 3.24: Benchmark pro algoritmy *LF*, *SL*, *SLF*, *LFI* a *SLI*

Weak benchmark pro algoritmy *LF*, *SL*, *SLF*, *LFI*, *SLI* a *GIS* je zobrazen na obr. 3.25.



Obrázek 3.25: Weak benchmark pro algoritmy *LF*, *SL*, *SLF*, *LFI*, *SLI* a *GIS*

Poznámka. Žádný benchmark a hard-to-color graf z této kapitoly 3 není hard-to-color graf pro algoritmy využívající rozšířenou metodu interchange (příp. rozšířenou metodu interchange s přebarvením K3), tj. *LFIE*, *LFIEK3*, *SLIE*, *SLIEK3*, *CLFIE* a *CLFIEK3*.

4. Meta-algoritmus

Meta-algoritmus má jediný úkol, na základě vlastností daného grafu vybrat potencionálně nejvhodnější algoritmus z kapitoly 3.

Meta-algoritmus je v aplikaci *GraphColoring* označen jako "AI".

Meta-algoritmus bude obsahovat pro každý algoritmus jeden model, který bude vytvořen pomocí ML, jehož vstupními parametry budou vlastnosti daného grafu a výstup bude informace, zda je daný algoritmus na daný graf vhodný (true / false).

V kapitole 3 jsme se dozvěděli, že některé algoritmy zaručují optimalitu pro určité třídy grafů. Meta-algoritmus této znalosti využije a nejprve pro daný graf zjistí, zda nespadá do třídy grafů, kterou dokážeme optimálně obarvit.

Třídy grafů, které dokážeme optimálně obarvit v polynomiálním čase, jsou následující:

- chordální graf - obarví se hladově podle perfektního eliminačního schématu,
- bipartitní graf - první partita grafu se obarví jednou barvou a druhá partita grafu se obarví druhou barvou,
- úplný bipartitní graf - obarví se pomocí SL algoritmu,
- úplný graf - obarví se hladově podle libovolné posloupnosti vrcholů,
- kružnice a stromy - obarví se pomocí CS algoritmu.

Poznámka. Pokud danou třídu grafů umí optimálně obarvit více algoritmů, tak je vybrán algoritmus s nejmenší časovou složitostí.

Pokud meta-algoritmus nemá k dispozici žádné modely, tak nejvhodnější algoritmus vybere pouze na základě počtu vrcholů.

Pokud meta-algoritmus má pouze jeden model, pak všechny grafy obarvuje pouze jedním algoritmem, kterému patří daný model.

V případě, že meta-algoritmus obsahuje více než jeden model, tak na základě vlastností grafu a jednotlivých modelů zjistí, které algoritmy jsou vhodné pro daný graf. Těchto algoritmů může být samozřejmě více, a proto jako druhé kritérium se bude brát kvalita jednotlivých odhadů a ohodnocení daných modelů (jako např. přesnost modelu apod.), pokud ani toto nestačí k jednoznačnému výběru algoritmu, tak se vybírá algoritmus s nejmenší časovou složitostí.

Poznámka. Důvody, proč jsem se vydal cestou, kdy každý algoritmus má svůj model, místo toho, aby existoval pouze jeden model, jehož vstupními parametry budou vlastnosti grafu a výstupem bude přímo nejvhodnější algoritmus, jsou dva. První zásadnější důvod je ten, že s větším počtem možných výstupů se zvyšuje komplexnost celého modelu, díky čemuž bych musel nagenerovat velké množství dat, což by bylo časově neúnosné. Druhým důvodem je rozšiřitelnost, kdy se nemusí pro nový algoritmus vytvářet zcela nový model pro všechny algoritmy, ale vytvoří se pouze jeden model pro nově přidaný algoritmus.

Poznámka. Způsob, jakým se dané modely vytvářejí a ohodnocují je uveden v kapitole 6.4.2. Implementační detaily meta-algoritmu jsou uvedeny v kapitole 6.2.2.

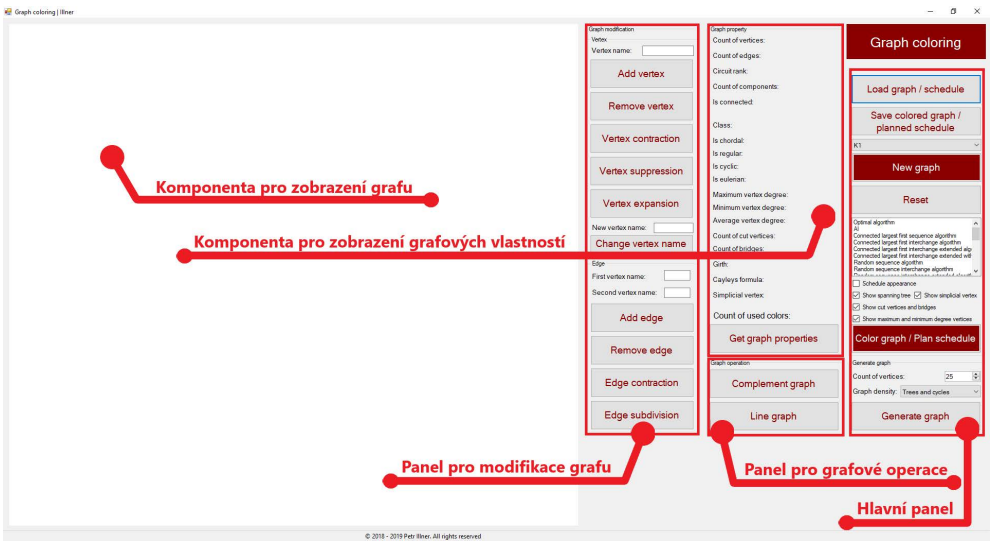
5. Uživatelská dokumentace

5.1 Program - GraphColoring

GraphColoring je aplikace s uživatelským rozhraním, která umožňuje následující:

1. načíst graf ze souboru,
2. vygenerovat náhodný graf s daným počtem vrcholů,
3. načíst graf z knihovny, která obsahuje přes 70 známých grafů,
4. obarvit graf pomocí více než 20 implementovaných algoritmů, z nichž jeden algoritmus využívá umělou inteligenci (označovaný jako AI), který pro daný graf na základě jeho vlastností vybere nejvhodnější algoritmus,
5. uložit (obarvený) graf do souboru,
6. vykreslit (obarvený) graf s několika jeho vlastnostmi (artikulace, mosty atd.),
7. uložit (obarvený) graf ve formátu *.jpg*,
8. pro daný graf spočítat více než 15 vlastností,
9. modifikovat grafy pomocí 9 modifikací a
10. provádět operace s grafy.

Při startu aplikace se nejdříve provedou automatické testy, které zkontrolují, zda je aplikace ve funkčním stavu. Chyba v některém z testů nutně neznamená, že se aplikace nespustí, pouze dá uživateli najevo, že mohou v dané oblasti, ve které test selhal, nastávat nějaké komplikace. Po provedení testů se spustí hlavní aplikace, která je zobrazena na obr. 5.1.



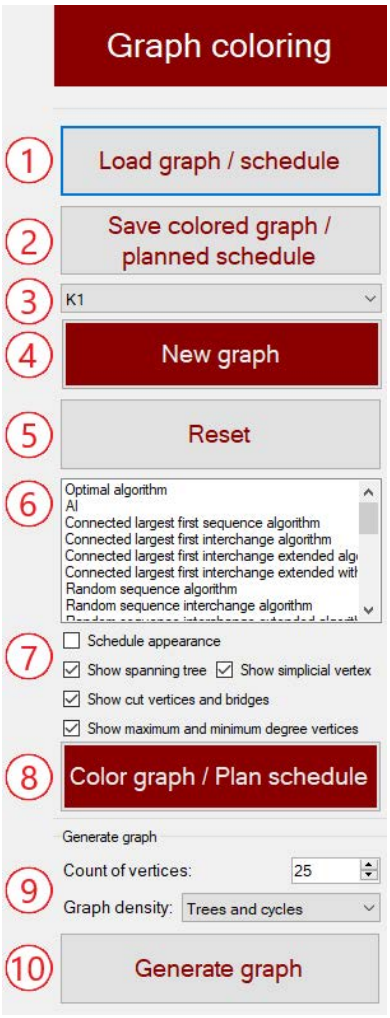
Obrázek 5.1: Aplikace *GraphColoring*

Uživatelské rozhraní aplikace se sestává ze 4 panelů, komponenty pro zobrazení (obarvených) grafů a status boxu, který se nachází v patičce aplikace.

Aplikace navíc vše vypisuje na konzoli, kam se mimo jiné i detailněji vypisují případné neočekávané chyby.

5.1.1 Hlavní panel

Hlavní panel, který je zobrazen na obr. 5.2, je nejpravější panel aplikace, který slouží pro nahrávání, ukládání, generování a obarvení grafů.



Obrázek 5.2: Hlavní panel

1. Tlačítko, které slouží k nahrání grafu z textového souboru s příponou *.graph*. Nahrání grafu lze také realizovat pomocí funkcionality *drag and drop*, kdy daný soubor s grafem stačí přetáhnout do aplikace. Pokud daný soubor bude jakýmkoliv způsobem nevalidní, tak se zobrazí chybová hláška.

2. Tlačítko, které slouží k uložení grafu. Pokud daný graf byl vytvořen jiným způsobem než načtením ze souboru, tak uživatel zadá cestu a jméno souboru, do kterého se má graf uložit. Pokud daný graf je obarvený, tak se do souboru uloží i dané obarvení včetně algoritmu, kterým byl daný graf obarven.

3. Kombinované pole (combo box), které obsahuje známé grafy.

4. Slouží k vytvoření grafu, který je vybrán v poli výše.

5. Po kliknutí na tlačítko *Reset*, dojde k resetu aplikace. Toto lze například využít v případě, kdy výpočty trvají delší dobu.

6. Výběrový seznam (list box), který obsahuje všechny barvící algoritmy.

7. Zaškrtávací pole (check box), které určují, jaké vlastnosti grafů se mají vykreslovat.

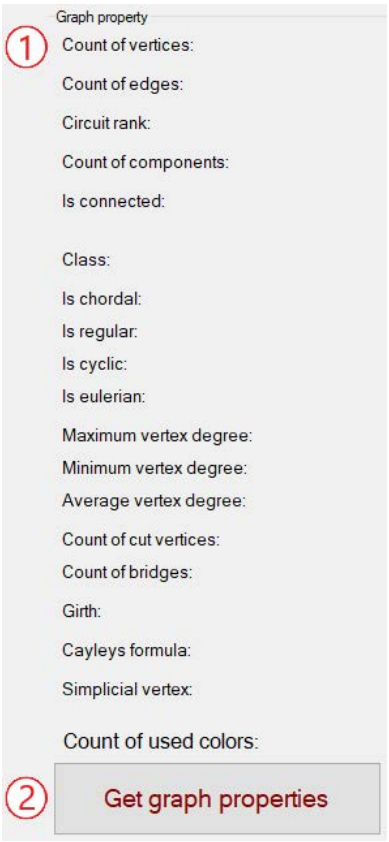
8. Obarví daný graf pomocí algoritmu vybraného ze seznamu z bodu *6*.

9. Parametry potřebné pro vygenerování grafu. Prvním parametrem je počet vrcholů a druhým parametrem je hustota grafu.

10. Vygeneruje graf podle parametrů nastavených v bodě *9*.

5.1.2 Panel pro zobrazení grafových vlastností

Jak již název napovídá, jedná se o panel, který uživateli zobrazuje vlastnosti grafu (počet vrcholů, počet hran, počet komponent atd.). *Panel pro zobrazení grafových vlastností* je zobrazen na obr. 5.3.



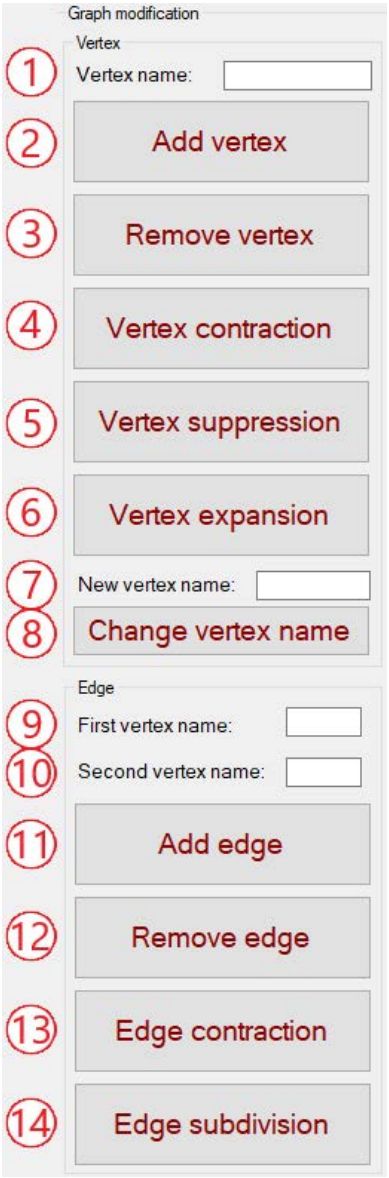
Obrázek 5.3: Panel pro zobrazení grafových vlastností

1. O grafu se zjišťují následující vlastnosti:
   - **počet vrcholů**, **hran** a **komponent**
   - **circuit rank** - lineární kombinace počtu hran, vrcholů a komponent, kde $r = |E| - |V| + |C|$,
   - **souvislost grafu**,
   - **třída grafu** (úplný graf (K_n), strom (T_n), kružnice (C_n), bipartitní graf, úplný bipartitní graf, nebo žádný),
   - **chordálnost** a **regulárnost** grafu,
   - zda se jedná o **eulerovský graf**,
   - **maximální**, **minimální** a **průměrný stupeň** grafu,
   - **počet artikulací** a **mostů**,
   - **girth**, tj. délka nejmenšího cyklu v grafu,
   - **Cayleyho formule**,
   - jeden libovolný **simpliciální vrchol**, pokud je graf chordální.
2. Tlačítko, které slouží k vypočítání všech vlastností grafu.

Poznámka. Pokud uživatel nechce z nějakého důvodu počítat všechny vlastnosti grafu, ale jenom některé z nich, tak stačí kliknout levým tlačítkem na příslušné popisky vlastností.

5.1.3 Panel pro modifikace grafu

Další panel, zobrazený na obr. 5.4, slouží pro grafové modifikace, které jsou uvedeny v kapitole 1.1.5.



Obrázek 5.4: Panel pro modifikace grafu

1. Název (identifikátor) vrcholu, na který chceme aplikovat nějakou vrcholovou modifikaci.
2. Přidá nový vrchol s názvem uvedeným v bodě *1*. Pokud daný vrchol existuje, tak se zobrazí chybová hláška.
3. Odstraní vrchol s názvem uvedeným v bodě *1*. Pokud daný vrchol neexistuje, tak se zobrazí chybová hláška.
4. Provede kontrakci vrcholu s názvem uvedeným v bodě *1*. Pokud daný vrchol neexistuje, tak se zobrazí chybová hláška.
5. Potlačí vrchol s názvem uvedeným v bodě *1*. Pokud stupeň daného vrcholu je jiný než dva, nebo daný vrchol neexistuje, tak se zobrazí chybová hláška.
6. Provede expanzi vrcholu s názvem uvedeným v bodě *1*. Pokud daný vrchol neexistuje, tak se zobrazí chybová hláška.
7. Pole pro vložení nového názvu (identifikátoru) vrcholu.
8. Přejmenuje vrchol s názvem uvedeným v bodě *1* na název uvedený v bodě *7*. Pokud daný vrchol neexistuje, nebo název je již použit u jiného vrcholu, tak se zobrazí chybová hláška.
9. Název (identifikátor) prvního vrcholu, na který chceme aplikovat nějakou hranovou modifikaci.
10. Název (identifikátor) druhého vrcholu, na který chceme aplikovat nějakou hranovou modifikaci.
11. Přidá hranu mezi vrcholy s názvy uvedenými v bodech *9* a *10*. Pokud jeden z vrcholů neexistuje, nebo hrana mezi vrcholy již existuje, tak se zobrazí chybová hláška.
12. Odstraní hranu incidentní s vrcholy, jejichž názvy jsou uvedeny v bodech *9* a *10*. Pokud jeden z vrcholů neexistuje, nebo hrana mezi vrcholy neexistuje, tak se zobrazí chybová hláška.
13. Provede kontrakci hrany incidentní s vrcholy, jejichž názvy jsou uvedeny v bodech *9* a *10*. Pokud jeden z vrcholů neexistuje, nebo neexistuje hrana mezi vrcholy, tak se zobrazí chybová hláška.

14. Provede dělení hrany incidentní s vrcholy, jejichž názvy jsou uvedeny v bodech *9* a *10*. Pokud jeden z vrcholů neexistuje, nebo hrana mezi vrcholy neexistuje, tak se zobrazí chybová hláška.

5.1.4 Panel pro grafové operace

Aplikace nabízí pouze dvě grafové operace, a to vytvoření komplementárního a hranového grafu. Panel pro grafové operace je zobrazen na obr. 5.5.

1. Tlačítko pro vytvoření komplementárního grafu.
2. Tlačítko pro vytvoření hranového grafu.

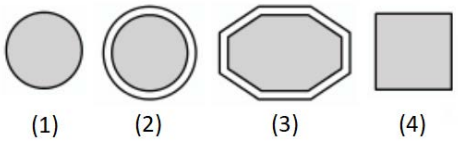
Obrázek 5.5: Panel pro grafové operace

5.1.5 Komponenta pro zobrazení grafu

Komponenta pro zobrazení grafu mimo samotného zobrazení (obarveného) grafu, zobrazuje i následující vlastnosti (pokud jsou zaškrtlé příslušné zaškrtávací pole v hlavním panelu v bodě *7*):

- všechny **vrcholy s nejmenším** a **největším stupněm**,
- jeden libovolný **simpliciální vrchol**,
- **kostru grafu**, která vznikla prohledáváním do šířky (BFS),
- **artikulace** a **mosty** grafu.

Simpliciální vrcholy a vrcholy s nejmenším, resp. největším stupněm se odlišují pomocí různých tvarů vrcholů, které jsou zobrazeny na obr. 5.6.



Obrázek 5.6: Tvary vrcholů

1. Obyčejný vrchol.
2. Vrchol s nejmenším stupněm v grafu.
3. Vrchol s největším stupněm v grafu.
4. Simpliciální vrchol.

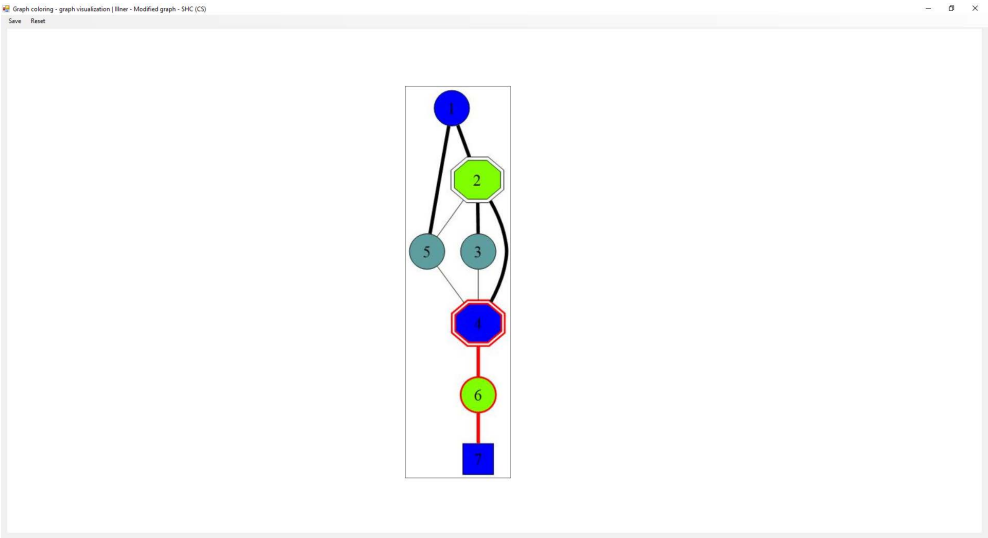
Poznámka. Pokud nějaký vrchol je simpliciálním vrcholem a zároveň má nejmenší (resp. největší) stupeň v grafu, tak se preferuje simpliciálnost, tzn. že se vykreslí jako simpliciální vrchol.

Artikulace a mosty se neodlišují tvarem vrcholu, resp. hrany, ale barvou, která je červená.

Hrany, které patří do kostry jsou vykresleny **tučně**.

Vykreslují se grafy, jejichž počet vrcholů je nejvýše 50. Toto omezení je ze dvou důvodů. Prvním je, že větší grafy i s menší hustotou se stávají nepřehlednými a druhým důvodem je větší časová složitost na vykreslení větších grafů. Další podmínkou pro vykreslení obarveného grafu je použití maximálně 14 barev.

Po kliknutí na komponentu pro zobrazování grafů se otevře nové okno (*GraphVisualization*) s vykresleným grafem. V tomto okně lze zoomovat a posouvat se libovolně v grafu. Toto okno také podporuje uložení (obarveného) grafu ve formátu *.jpg*. Okno je zobrazeno na obr. 5.7.



Obrázek 5.7: Vizualizace grafu

5.1.6 Soubory s příponou *.graph*

Aplikace *GraphColoring* načítá grafy (příp. ukládá grafy) pouze ze souborů s příponou *.graph* se strukturou popsanou níže. Jiné formáty nejsou povoleny. Případný převod ze standardního formátu *.col* bude uveden v kapitole 5.2.4.

Struktura souboru *.graph*:

```
1  Graph coloring. Graph representation: reprezentace grafu
2
3  Graph name: název grafu
4  Number of vertices: počet vrcholů
5
6  GRAPH
7  popis grafu
8
9  COLORED GRAPH
10 popis obarveného grafu
```

Tučné texty a prázdné řádky jsou pro všechny grafy stejné, *texty kurzívou* se pro různé grafy mohou lišit.

Aplikace podporuje pouze 3 reprezentace grafů. Seznam hran (**edgeList**), matici sousednosti (**adjacencyMatrix**) a seznam sousedů (**adjacencyList**).

Název grafu může být libovolný řetězec.

Popis grafu se pro každou reprezentaci liší.

Pro reprezentaci grafu pomocí seznamu hran je na každém řádku popsaná jedna hrana pomocí dvou názvů vrcholů, které jsou uzávorkovány.

Příklad popisu grafu pomocí *seznamu hran*:

```
1 (název vrcholu 1) (název vrcholu 2)
2 (název vrcholu 3) (název vrcholu 4)
3 ...
```

Popis grafu pomocí matice sousednosti obsahuje matici $|V| \times |V|$, jejíž každý prvek obsahuje buď jedničku, nebo nulu.

Příklad popisu grafu pomocí *matice sousednosti*:

```
1 0  ...  1
2    ⋱
3 1  ...  0
```

Poslední reprezentací grafu je seznam sousedů, která se používá na ukládání vygenerovaných grafů v aplikaci.

Příklad popisu grafu pomocí *seznamu sousedů*:

```
1 název vrcholu 1
2 (název prvního souseda vrcholu 1)
3 (název druhého souseda vrcholu 1)
4 ...
5 (název posledního souseda vrcholu 1)
6 název vrcholu 2
7 (název prvního souseda vrcholu 2)
8 ...
```

Popis obarveného grafu je stejný pro všechny grafové reprezentace.

Popis obarveného grafu:

```
1 Number of colors: počet barev
2 Used algorithm: název algoritmu
3 − 1
4 −− název prvního vrcholu, který je obarven barvou 1
5 −− název druhého vrcholu, který je obarven barvou 1
6 ...
7 −− název posledního vrcholu, který je obarven barvou 1
8 − 2
9 −− název prvního vrcholu, který je obarven barvou 2
10 ...
```

Poznámka. Každý algoritmus může mít uložené v souboru svoje obarvení. Aplikace *GraphColoring* ale neukládá obarvení grafu, pokud v souboru již existuje obarvení pomocí samého **nepravděpodobnostního** algoritmu. V opačném případě, pokud se jedná o pravděpodobnostní algoritmus, tak se do souboru ukládá každé obarvení, i přestože v souboru se již nachází obarvení pomocí daného algoritmu. Důvod je zřejmý, nepravděpodobnostní algoritmus obarvuje stejný graf stále stejně, zatím co u pravděpodobnostního algoritmu se každé obarvení může lišit.

Příklad. Soubor s grafem na obr. 5.7 popsaný pomocí matice incidence a obarvený

dvěma algoritmy by vypadal následovně:

PrikladSouboru.graph:

```
1  Graph coloring. Graph representation: adjacencyMatrix
2
3  Graph name: Příklad grafu
4  Number of vertices: 7
5
6  GRAPH
7  0 1 0 0 1 0 0
8  1 0 1 1 1 0 0
9  0 1 0 1 0 0 0
10 0 1 1 0 1 1 0
11 1 1 0 1 0 0 0
12 0 0 0 1 0 0 1
13 0 0 0 0 0 1 0
14
15 COLORED GRAPH
16 Number of colors: 3
17 Used algorithm: connectedLargestFirst
18 - 1
19 -- 2
20 - 2
21 -- 4
22 -- 1
23 -- 7
24 - 3
25 -- 5
26 -- 3
27 -- 6
28
29 Number of colors: 4
30 Used algorithm: randomSequence
31 - 1
32 -- 1
33 -- 3
34 -- 6
35 - 2
36 -- 5
37 -- 7
38 - 3
39 -- 4
40 - 4
41 -- 2
```

5.2 Program - GraphColoringConsole

GraphColoringConsole je konzolová aplikace, která je určena pro práci s velkým počtem grafů a k experimentování. Aplikace umožňuje následující:

1. generování náhodných grafů do souboru,
2. generování náhodných grafů do databáze,
3. vložení grafů ze souborů do databáze,
4. vytvoření modelů pro AI,
5. převod grafů ze standardního formátu *.col* na formát *.graph*,
6. měření časových složitostí jednotlivých algoritmů a
7. obarvení grafů.

Po spuštění aplikace se zobrazí menu, které je zobrazeno na obr. 5.8.

Obrázek 5.8: Menu aplikace *GraphColoringConsole*

5.2.1 Generování náhodných grafů

Aplikace umožňuje generovat a obarvit (pomocí všech algoritmů) náhodné grafy, které následně uloží do souboru nebo do databáze.

Soubory s příponou *.graphDB*

Soubory s příponou *.graphDB* slouží k ukládání grafů a jejich obarvení. Grafy se ukládají v redukovaném formátu, kdy se neukládá celý graf, nýbrž pouze vlastnosti daného grafu.

Struktura souboru *.graphDB*:

1 **Graph:** *počet-vrcholů počet-hran třída-grafu je-chordální je-regulární je-cyklický je-eulerovský maximální-stupeň-grafu minimální-stupeň-grafu průměrný-stupeň-grafu medián-stupně-grafu počet-artikulací počet-mostů girth neklesající-skóre-grafu*
2 *název-nepravděpodobnostního-algoritmu: počet-použitých-barev*
3 *název-pravděpodobnostního-algoritmu: počet-iterací-algoritmu nejmenší-počet-použitých-barev největší-počet-použitých-barev*
4 ...

Hodnoty pro položku *třída grafu* jsou: *none, completeGraph, treeGraph, cycleGraph, bipartiteGraph a completeBipartiteGraph.*

Hodnoty pro položku *eulerovský graf* jsou: *eulerian, semiEulerian a notEulerian.*

V souboru se také mohou nacházet řádky s komentáři. Tyto řádky začínají slovem **Comment:** .

Příklad. Soubor s dvěma grafy a s dvěma obarveními (jedno obarvení pomocí nepravděpodobnostního algoritmu a druhé pomocí pravděpodobnostního algoritmu) by vypadal následovně:

PrikladSouboru.graphDB:

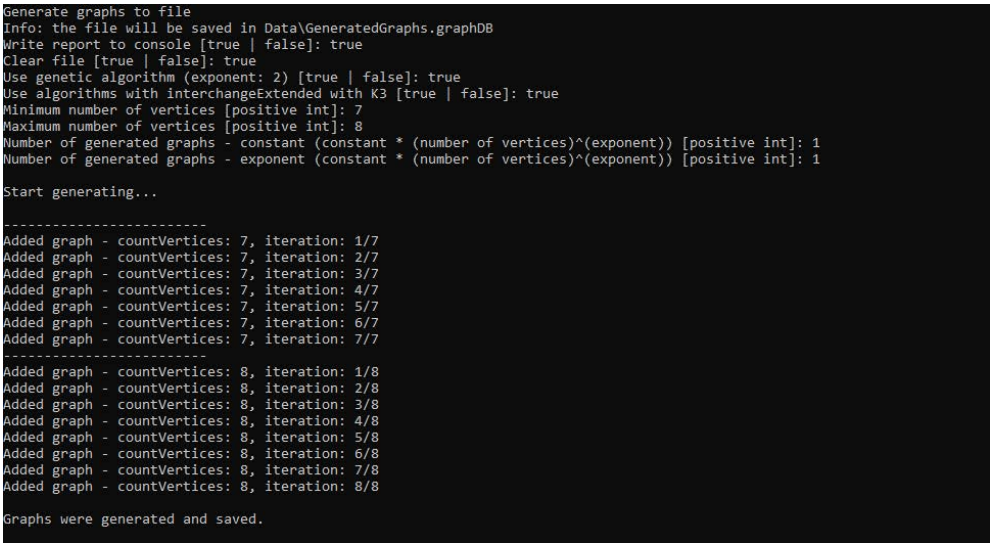
```
1 Comment: HC (CS)
2 Graph: 18 27 none False True True notEulerian 3 3 3 3 2 1 3 3 3 3 3 3 3 3
    3 3 3 3 3 3 3 3 3 3 3
3 randomSequence 10 3 4
4 connectedSequential 4
5 Comment: HC (GIS)
6 Graph: 6 5 treeGraph True False False notEulerian 3 1 1.66666666666667 1
    2 5 0 1 1 1 1 3 3
7 greedyIndependentSet 3
8 geneticAlgorithm2 10 2 2
```

Generování náhodných grafů do souboru

Vygenerované a obarvené grafy se budou ukládat do souboru s umístěním *Data\GeneratedGraphs.graphDB*. Před samotným generováním a obarvováním grafů bude po uživateli požadováno zadání několika dodatečných informací:

1. má se vypisovat průběh generování a obarvování grafů na konzoli,
2. pokud existuje neprázdný soubor *GeneratedGraphs.graphDB*, má se smazat jeho obsah,
3. může se použít na obarvování grafů genetický algoritmus s exponentem 2,
4. můžou se použít na obarvování grafů algoritmy, které využívají metodu *interchangeExtendedK3*,
5. od jakého počtu vrcholů se mají generovat grafy,
6. do jakého počtu vrcholů se mají generovat grafy a
7. kolik se má vygenerovat grafů, kde počet vygenerovaných grafů s počtem vrcholů n se odvíjí od vzorečku $c \times n^e$, kde c je konstanta a e je exponent.

Příklad. Příklad generování grafů do souboru je na obr. 5.9.



Obrázek 5.9: Generování grafů do souboru - GraphColoringConsole

Poznámka. Pro pravděpodobnostní algoritmy se grafy obarvují 10x, kde se bere nejmenší a největší počet použitých barev na obarvení daného grafu.

Poznámka. Důvod, proč si uživatel může vybrat, zda chce grafy obarvovat genetickým algoritmem s exponentem 2 nebo algoritmy využívající metodu *interchangeExtendedK3*, je kvůli jejich větší časové složitosti.

Poznámka. Počet vygenerovaných grafů s počtem vrcholů menší než 7 je pevný a není ovlivněn vzorečkem $c \times n^e$. Důvod je ten, že počet neisomorfních grafů s počtem vrcholů $<$ 7 je malý a snahou je neplýtvat časem na generování a obarvování zbytečně mnoha (stejných) grafů. Počet vygenerovaných grafů je následující:

- pro n = 1 se vygeneruje 1 graf,
- pro n = 2 se vygeneruje 1 graf,
- pro n = 3 se vygenerují 2 grafy,
- pro n = 4 se vygenerují 4 grafy,
- pro n = 5 se vygeneruje 11 grafů,
- pro n = 6 se vygeneruje 34 grafů,

což odpovídá přesně počtu neisomorfních grafů.

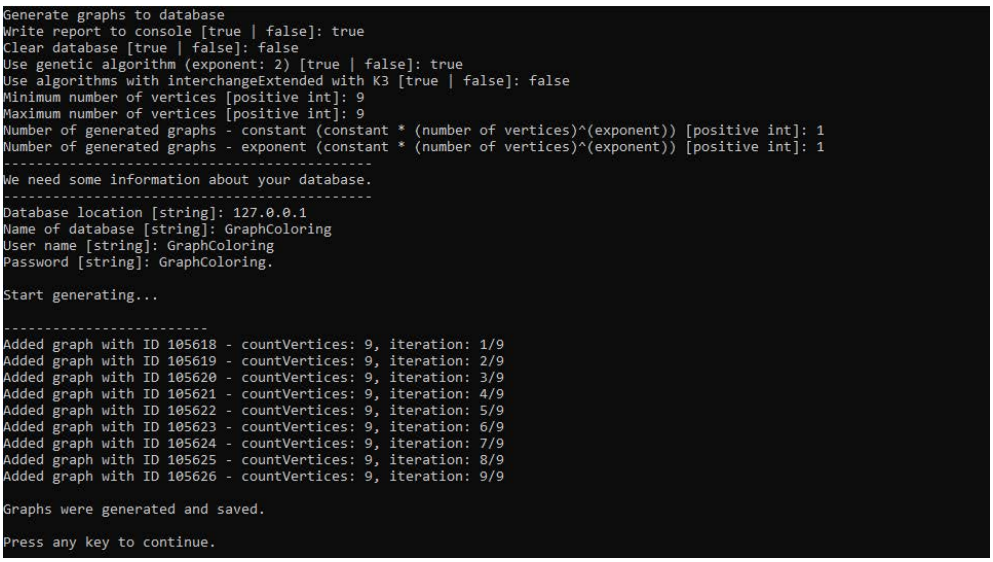
Generování náhodných grafů do databáze

Vygenerované a obarvené grafy se budou ukládat do databáze, jejíž schéma je uvedeno v kapitole 6.1. Před samotným generováním a obarvováním grafů bude po uživateli požadováno několik dodatečných informací, které jsou stejné jako při generování náhodných grafů do souboru obohacené o informace týkající se databáze:

8. umístění databáze,

9. jméno databáze,
10. přihlašovací uživatelské jméno a
11. přihlašovací heslo.

Příklad. Příklad generování grafů do databáze je na obr. 5.10



Obrázek 5.10: Generování grafů do databáze - GraphColoringConsole

Poznámka. Pokud přihlášení do databáze proběhne v pořádku, tak aplikace si tyto údaje uloží a uživatel již v budoucnosti nebude muset vyplňovat tyto informace znova.

5.2.2 Vytvoření modelů pro meta-algoritmus

Pomocí aplikace lze vytvořit i modely pro meta-algoritmus, které využívá aplikace *GraphColoring* pro algoritmus *AI*. Pro každý algoritmus se vytvoří jeden model. Modely se ukládají do složky *Data* a mají příponu *.zip*. Data určené na trénování a testování se berou z databáze. Při vytvoření modelu se v konzoli vypisuje i ohodnocení daného modelu, díky čemuž se uživatel může rozhodnout, zda nevymění stávající model daného algoritmu v aplikaci *GraphColoring*, který se nachází ve složce *AIModels*. Pokud pro daný algoritmus nelze z důvodu malého množství dat vytvořit model, tak se zobrazí chybová hláška.

Poznámka. Aktuální modely a jejich ohodnocení, které používá aplikace *GraphColoring*, jsou uvedeny v příloze A.

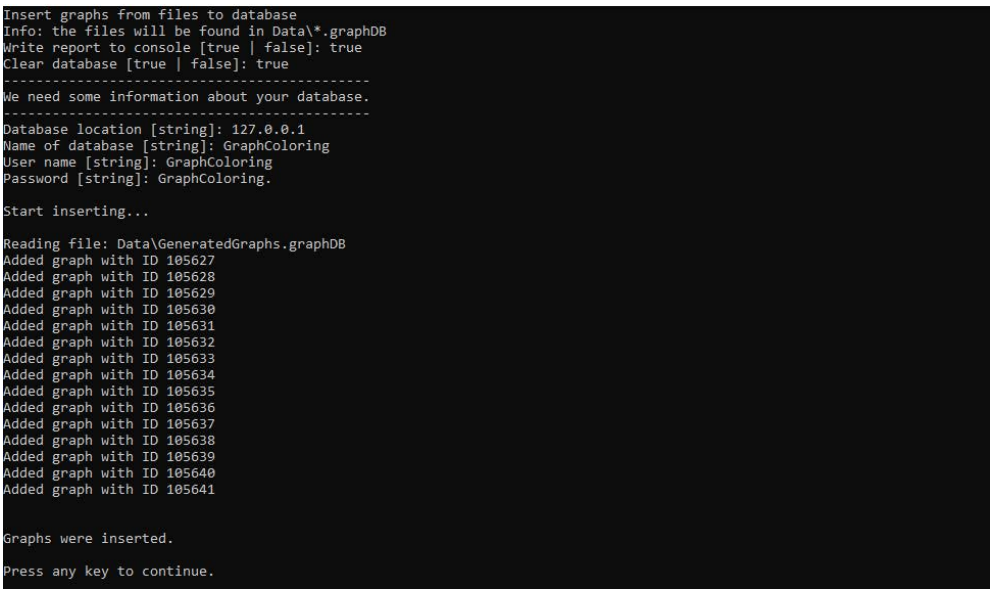
5.2.3 Vložení grafů ze souborů do databáze

Aplikace umožňuje vložení grafů ze souborů *.graphDB* do databáze. Berou se všechny soubory s umístěním *Data*.graphDB*. Před nahráním grafů do databáze se požadují následující informace:

1. má se vypisovat průběh nahrávání grafů do databáze a

2. mají se před vložením odstranit všechny grafy z databáze.

Příklad. Příklad vkládání grafů ze souborů do databáze je na obr. 5.11.



Obrázek 5.11: Vkládání grafů ze souborů do databáze - GraphColoringConsole

5.2.4 Převod z *.col* na *.graph*

Existuje běžně používaná kolekce grafů *DIMACS*, jejíž grafy jsou ve formátu *.col*, a proto aplikace umí převádět grafy z formátu *.col* na formát *.graph*. Potencionální soubory k převodu se hledají ve složce *Data*.

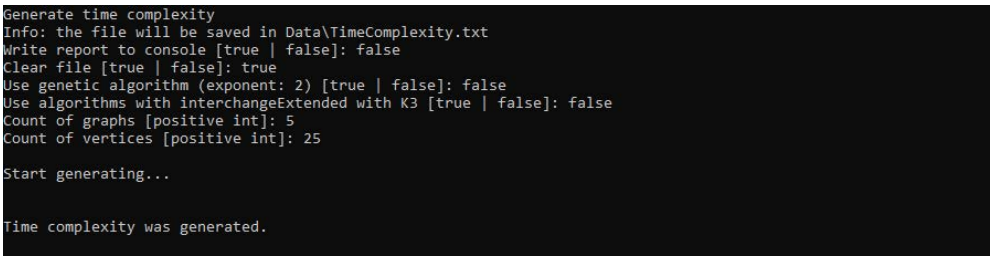
Příklad. Příklad převodu grafů z *.col* na *.graph* je na obr. 5.12.

Obrázek 5.12: Převod grafů z *.col* na *.graph* - GraphColoringConsole

5.2.5 Měření časových složitostí algoritmů

Aplikace umí pro každý algoritmus změřit čas potřebný na obarvení náhodného grafu s daným počtem vrcholů. Výsledek z měření se uloží do souboru s umístěním *Data\TimeComplexity.txt*. Kvůli přesnosti měření se daný graf obarvuje jedním algoritmem 100x, a tedy obarvování větších grafů může trvat i několik hodin až dní! Uživatel zadá počet vrcholů, na základě čehož aplikace vygeneruje dva grafy, z nichž jeden je řidší a druhý je hustější, které postupně obarví pomocí všech algoritmů. Toto lze iterovat pro několik grafů, jehož počet zadal uživatel před spuštěním. U algoritmů používající libovolnou variantu metody interchange je uveden i počet jeho volání.

Příklad. Příklad měření časových složitostí je na obr. 5.13.

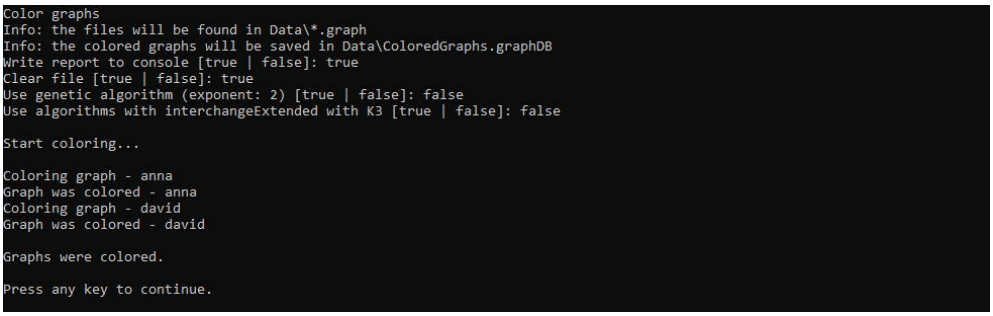


Obrázek 5.13: Měření časových složitostí - GraphColoringConsole

5.2.6 Obarvování grafů

Další funkcionalitou aplikace je obarvování grafů. Grafy k obarvení se hledají ve složce *Data*. Výsledek z obarvování se uloží do souboru s umístěním *Data\ColoredGraphs.graphDB*, který je formátu *.graphDB* a obsahuje vždy komentář s názvem grafu, následovaný popisem grafu pomocí jeho vlastností a případných obarvení pomocí jednotlivých algoritmů.

Příklad. Příklad obarvování grafů je na obr. 5.14.



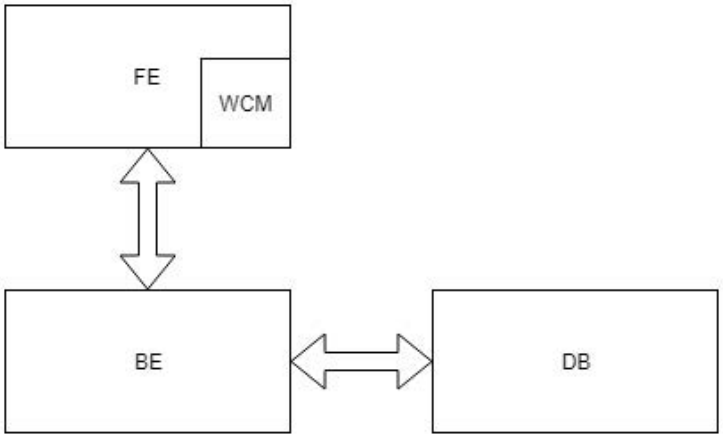
Obrázek 5.14: Obarvování grafů - GraphColoringConsole

6. Vývojová dokumentace

Aplikace je tvořena třemi vrstvami, které jsou od sebe navzájem odděleny. Jedná se o následující vrstvy:

- databáze (*DB*),
- backend (*BE*) a
- frontend (*FE*) s *WCM*.

Komunikace mezi vrstvami je zobrazena na obr. 6.1.



Obrázek 6.1: Komunikace mezi vrstvami

Databáze má dvě využití. První využití je pro uživatele, který pomocí připravených pohledů a metod může získávat data pro experimenty. Databázi ale využívá i *BE*, který využívá její data jako trénovací a testovací data pro ML.

BE je netriviální částí aplikace, který se stará například o

- načítání (příp. ukládání) grafů ze souborů,
- reprezentaci grafů,
- výpočty vlastností grafů,
- generování grafů,
- barvení grafů atd.

FE zprostředkovává vazbu mezi uživatelem a *BE*. *WCM* obsahuje všechny textace.

Poznámka. Kvůli rozsáhlosti projektu zde budou popsány jen ty nejdůležitější věci.

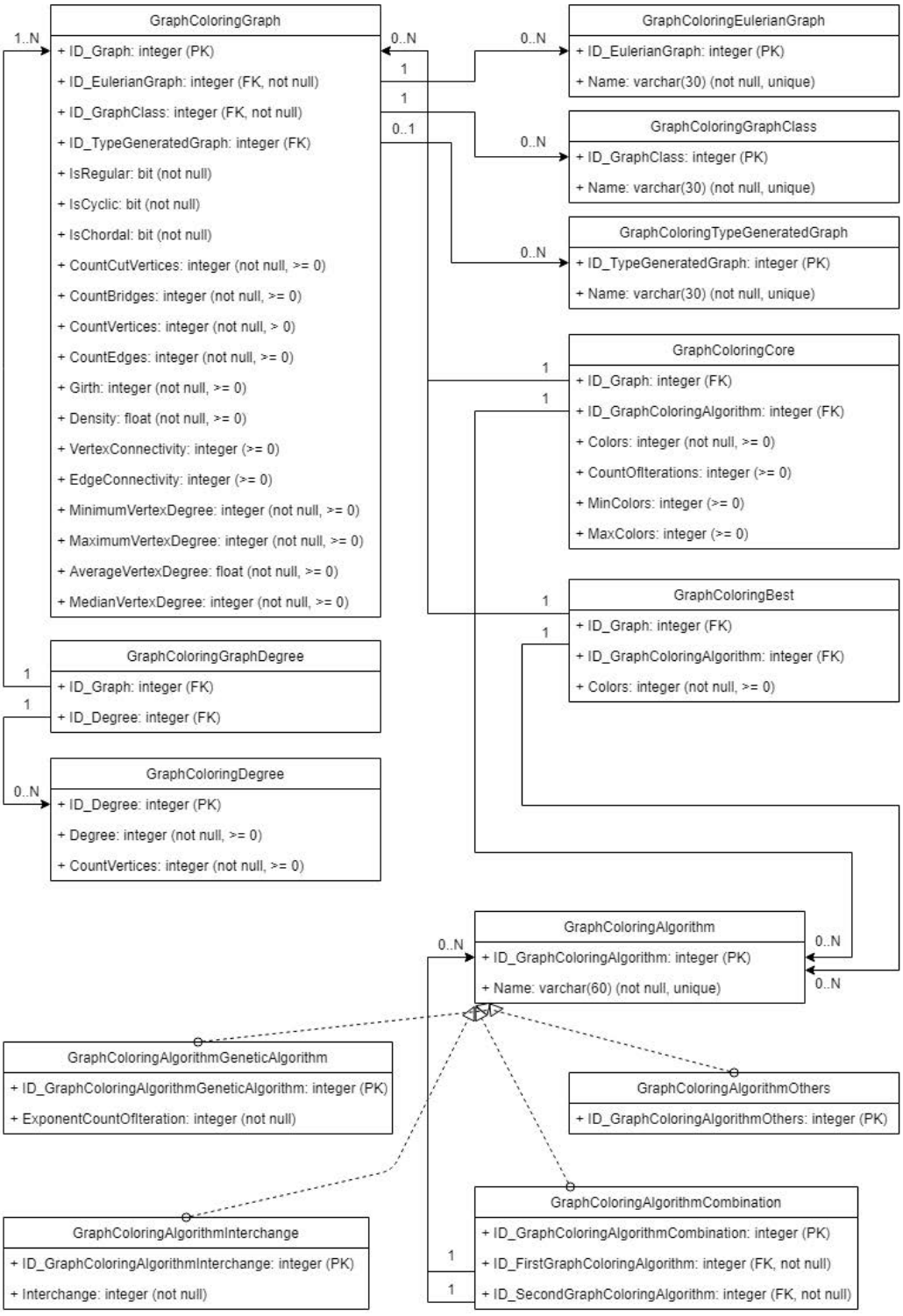
Tuto dokumentaci doplňují zdrojové kódy a komentáře, které tyto zdrojové kódy obsahují. Podrobnou strukturu funkcí a procedur lze nejlépe pochopit právě v těchto zdrojových kódech.

6.1 Databáze

Databáze se sestává ze 13 tabulek, z nichž nejdůležitější jsou *GraphColoringGraph*, *GraphColoringCore* a *GraphColoringBest*.

Poznámka. Skripty pro vytvoření schématu databáze jsou určeny pro MS SQL.

Schéma databáze je zobrazené na obr. 6.2.



Obrázek 6.2: Schéma databáze

6.1.1 Tabulka GraphColoringGraph

V této tabulce jsou uloženy všechny grafy, které jsou reprezentovány pouze jeho vlastnostmi. V tabulce se nevyskytují dva stejné grafy, tj. grafy se stejnými vlastnostmi, za což ručí procedury a funkce, které modifikují danou tabulku.

Při vkládání nového grafu se kontroluje, zda sedí počet vrcholů se skórem grafu a zda platí princip sudosti.

Poznámka. V tabulce jsou vlastnosti *vertexConnectivity* (vrcholová souvislost) a *edgeConnectivity* (hranová souvislost), které nejsou z časové vytíženosti na BE implementovány, a proto budou mít vždy hodnotu *NULL*.

6.1.2 Tabulka GraphColoringCore

Tato tabulka obsahuje obarvení grafů pomocí algoritmů. Každý záznam obsahuje identifikátor grafu, algoritmu a počet použitých barev, pokud se jedná o pravděpodobnostní algoritmus, tak obsahuje navíc počet iterací (kolikrát byl graf pomocí daného algoritmu obarven) včetně nejmenšího a největšího počtu použitých barev. U pravděpodobnostního algoritmu se počet použitých barev spočítá jako aritmetický průměr z nejmenšího a největšího počtu použitých barev se zaokrouhlením nahoru.

Záznam s jedním grafem a jedním algoritmem se může v tabulce vyskytovat nejvýše jednou. Pokud se zavolá přidání obarvení a obarvení již existuje, tak dojde k modifikaci uloženého obarvení, kdy se aktualizovaný počet použitých barev spočítá jako aritmetický průměr ze staré a nové hodnoty se zaokrouhlením nahoru.

Poznámka. Metody pro přidání obarvení jsou rozděleny pro nepravděpodobnostní a pravděpodobností algoritmy a za jejich správné volání ručí ten, kdo je volá. Jinak řečeno, databáze neuchovává informaci o tom, zda algoritmus je pravděpodobnostního charakteru a tudíž nemá možnost ověřit, zda u volané metody nechybí vyplněné parametry, které jsou povinné pro pravděpodobnostní algoritmy.

6.1.3 Tabulka GraphColoringBest

Tabulka obsahující nejlepší obarvení pro grafy. Tuto tabulku modifikují pouze triggery, tzn. tabulku nejde modifikovat přímo pomocí metod! Při každém vložení / modifikaci záznamu v tabulce *GraphColoringCore* dojde k zavolání příslušného triggeru, který zaručí validní data v tabulce *GraphColoringBest*. Každý záznam obsahuje identifikátor grafu, algoritmu a počet použitých barev.

Poznámka. Z této tabulky se vytvářejí data pro BE, které jsou použity k učení ML.

6.2 Backend - GraphColoring

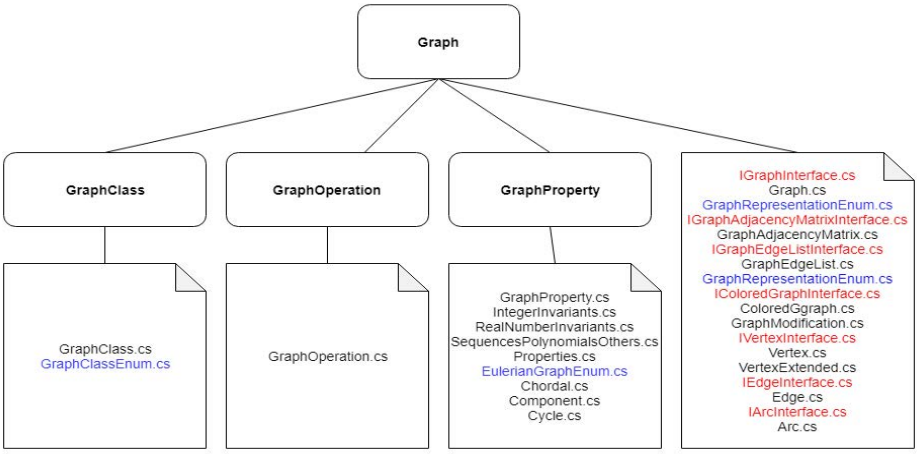
BE se stará o následující věci:

- vytvoření grafu (složka *Graph*),
- reprezentaci grafu (složka *Graph*),
- obarvení grafu a všechny věci s tím spojené (složka *Graph\ColoredGraph* a *GraphColoringAlgorithm*),
- zjištění třídy grafu (složka *Graph\GraphClass*),
- grafové modifikace (složka *Graph\GraphModification*),
- grafové operace (složka *Graph\GraphOperation*),
- výpočty vlastností grafu (složka *Graph\GraphProperty*),
- generování náhodného grafu (složka *GenerateGraph*),
- vizualizaci grafu (složka *GraphVisualization*),
- práci se soubory (složka *ReaderWriter*) a
- testování jednotlivých funkcionalit (složka *Tests*).

6.2.1 Graph

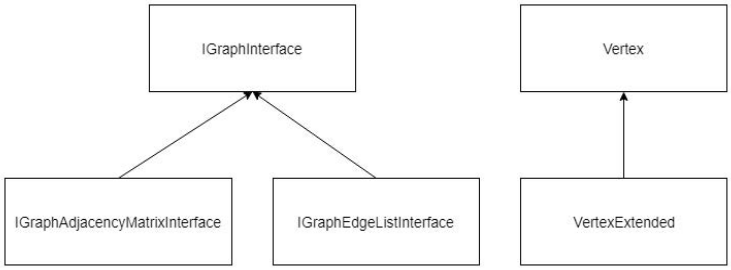
Namespace *Graph* obsahuje třídy pro reprezentaci vrcholu, reprezentaci hrany, reprezentaci grafu, modifikace grafu, operace grafu, vlastnosti grafu a třídy grafu.

Hierarchie namespacu *Graph* je zobrazena na obr. 6.3.



Obrázek 6.3: Hierarchie namespacu *Graph*

Hierarchie dědění v namespacu *Graph* je zobrazena na obr. 6.4.



Obrázek 6.4: Hierarchie dědění v namespacu *Graph*

Graf je reprezentovaný seznamem sousedů, kde seznam vrcholů je reprezentovaný *Dictionary* a seznam sousedů je reprezentovaný *HashSet*em. Důvod výběru této reprezentace je snaha o minimalizaci časových složitostí pro operace nad grafy i za cenu větší paměťové složitosti (způsobené hashovací tabulkou). Zjištění, zda dva vrcholy jsou spojeny hranou a přidání, resp. odebrání hrany v grafu dokážeme s amortizovanou časovou složitostí v konstantním čase a nalezení všech sousedů k nějakému vrcholu v má časovou složitost $\mathcal{O}(N_G(v))$, což jsou nejmenší možné časové složitosti pro dané operace.

Každý vrchol obsahuje dva identifikátory:

- identifikátor (*identifier*), což je vygenerované číslo programem, s kterým pracuje pouze program,
- uživatelské jméno vrcholu (*userName*), což je řetězec definovaný uživatelem.

Důvod dvou identifikátorů je ten, že program pracuje s číslem, což je rychlejší a uživatel pracuje s řetězcem, což je pro uživatele příjemnější. Navíc si dané vrcholy může označovat "libovolně".

Graf tedy kromě seznamu sousedů obsahuje ještě dvě *Dictionary*, jednu pro převod z identifikátoru na uživatelské jméno (*mapping*) a druhou pro převod z uživatelského jména na identifikátor (*mappingUserName*).

Vrchol navíc obsahuje atribut barvu (*color*), což je nezáporné číslo.

Rozdíl mezi třídou *Vertex* a *VertexExtended* je ten, že *VertexExtended* má *settery* na identifikátor, uživatelské jméno a barvu, zatím což *Vertex* obsahuje pouze *gettery* a žádné veřejné *settery*. Cílem je, aby třídu *VertexExtended* používala pouze třída *Graph* a všichni ostatní používali třídu *Vertex*, neboť nechceme, aby někdo jiný než *Graph* měnil hodnoty vrcholů, díky čemuž by mohla vzniknout nekonzistence dat.

Na vytváření grafu se využívají dvě zděděné třídy od *Graph*, a to *GraphAdjacencyMatrix* (pro matice sousedů) nebo *GraphEdgeList* (pro seznamy hran a seznamy sousedů). Třída *GraphAdjacencyMatrix* obsahuje metodu *SetOfNeighborsOfVertex (List<bool>)*, která načítá jeden řádek matice sousednosti. Třída *GraphEdgeList* má metodu *AddEdge (userName1, userName2)*, pomocí které si přidávají hrany do grafu. Protože graf může mít izolované vrcholy, které nejsou spojeny žádnou hranou, tak třída obsahuje ještě jednu metodu *AddVertex (userName)*, která slouží pouze pro přidání nového vrcholu.

Poté, co je graf vytvořený, tak se musí inicializovat voláním metody *InitializeGraph()*, jinak s grafem nelze standardně pracovat (obarvovat, modifikovat apod.)!

Další třídou je *ColoredGraph*, což je vnitřní privátní třída třídy *Graph*. Tato třída obsahuje vše nezbytné pro obarvení grafu, jako je obarvení vrcholu libovolnou barvou, hladové obarvení grafu pomocí posloupnosti vrcholů apod. Třída *ColoredGraph* implementuje i saturation, tzn. že si drží strukturu o nasycenosti jednotlivých vrcholů. Tato funkcionalita musí být před použitím zapnuta pomocí metody *SetSaturation*, neboť je standardně vypnuta! Po obarvení grafu musí být obarvený graf inicializován pomocí metody *InitializeColoredGraph()*, díky čemuž se zkontroluje, zda dané obarvení je validní. Kdokoliv mimo třídu *Graph* může získat referenci na danou instanci *ColoredGraph* pomocí metody *GetColoredGraph()*.

GraphModification.cs slouží pro grafové modifikace, které jsou implementovány intuitivně a není potřeba je popisovat. Nutno jenom podotknout, že po vykonání modifikací, je potřeba buď aktualizovat vlastnosti grafu pomocí příslušné metody, nebo resetovat všechny jeho vlastnosti a deinicializovat případný obarvený graf, aby nedocházelo k nekonzistencím dat.

GraphProperty

Vlastnosti grafu se dělí na:

- vlastnosti s booleovskými hodnotami (*Properties.cs*),
- vlastnosti s číselnými hodnotami (*IntegerInvariants.cs*),
- vlastnosti s reálnými hodnotami (*RealNumberInvariants.cs*),
- posloupnosti a jiné (*SequencesPolynomialsOthers.cs*).

Každá vlastnost grafu se počítá zvlášť. Pokud při výpočtu nějaké vlastnosti lze získat i jinou vlastnost, tak se toho využije.

Kdokoliv mimo třídu *Graph* může získat referenci na danou instanci *GraphProperty* pomocí metody *GetGraphProperty()*.

GraphProperty.cs obsahuje metodu pro reset vlastností a metody, které se volají při jednotlivých modifikacích grafu a jejichž cílem je zachovat (resp. poupravit) co nejvíce vlastností grafu, aby se nemusely zbytečně počítat znova.

Properties.cs zjišťuje o grafu následující:

- zda je graf souvislý (se získáním komponent souvislosti a počtu komponent) (*Component.cs*),
- zda je graf regulární (se získáním skóre grafu),
- zda je graf eulerovský (se získáním skóre grafu a zjištěním, zda je graf souvislý),
- zda graf obsahuje cyklus a
- jestli se jedná o chordální graf (s vytvořením perfektního eliminační schématu) (*Chordal.cs*).

Ke zjištění nesouvislosti grafu se nejdříve zkontroluje, zda $|E| < |V| - 1$, a to díky větě 3. V opačném případě se používá standardní algoritmus, kdy se vytvoří struktura obsahující vrcholy s číslem komponenty, které jsou ze začátku vynulované. Pro každý vrchol s číslem komponenty 0 se provede BFS, kdy se vrcholům, které byly nalezeny, přiřadí číslo, které je o jedna větší než dosavadní počet nalezených komponent. Časová složitost je $\mathcal{O}(|V| + |E|)$, neboť se projde celý graf jednou.
Pokud graf je souvislý, tak se do seznamu souvislých komponent dá ukazatel na daný graf, pokud počet komponent je větší než 1, tak se pro každou souvislou komponentu **vytvoří nový graf** a odkaz na daný graf se přidá do seznamu komponent.

Regulárnost grafu se zjistí jednoduše, a to tak, že se porovná maximální a minimální stupeň grafu. Časová složitost je $\mathcal{O}(|V| + |E|)$, neboť se zároveň vytváří i skóre grafu.

Pokud všechny stupně grafu jsou sudé, tak se jedná o eulerovský graf. Pokud počet vrcholů s lichým stupněm je nejvýše dva, tak se jedná o semi-eulerovský graf. Graf s větším počtem lichých stupňů než dva není eulerovským grafem. Navíc graf je eulerovský, resp. semi-eulerovský pokud je souvislý. Časová složitost je $\mathcal{O}(|V| + |E|)$, neboť se zároveň vytváří skóre grafu a zjišťuje se, zda je graf souvislý.

Ke zjištění cyklu jsou naimplementovány dvě metody. První využívá standardní BFS algoritmus s časovou složitostí $\mathcal{O}(|V| + |E|)$. Druhá varianta používá *Union-Find* s časovou složitostí operace Find $\mathcal{O}(1)$ a Union $\mathcal{O}(|V|)$. Tedy časová složitost na zjištění cyklu pomocí metody *Union-Find* je $\mathcal{O}(|V| \times |E|)$. Na základě časové složitosti se používá pouze první varianta.

Chordálnost grafu se ověřuje tak, že se pomocí algoritmu *MCS*[5] vygeneruje posloupnost vrcholů, která je pro chordální grafy perfektním eliminačním schématem. Tedy po získání posloupnosti od algoritmu *MCS* stačí ověřit, zda se jedná o perfektní eliminační schéma.

Algorithm 26: MCS

```
1  MCS (G)
2      foreach vrchol v grafu G do
3          w(v) = 0;
4      end
5      for i :- n to 1 do
6          v :- neoznačený vrchol s největší váhou w(v);
7          foreach neoznačený soused v_i vrcholu v do
8              w(v_i) = w(v_i) + 1;
9          end
10         sequence[i] = v;
11         označit vrchol v;
12     end
13     return sequence;
```

MCS lze implementovat s časovou složitostí $\mathcal{O}(|V| \times log(|V|) + |E|)$.

Tvrzení 26. *Pokud graf G je chordální, pak MCS algoritmus vrátí perfektní eliminační schéma.*

Důkaz. Důkaz provedeme sporem. Nechť tedy G je chordální graf a MCS algoritmus vrátil posloupnost vrcholů $(v_1, v_2, \ldots, v_n)$, která není perfektním eliminačním schématem grafu G. Pak se tedy v posloupnosti nachází vrchol v_i jehož sousedi vpravo v posloupnosti (tj. vrcholy s větším indexem) netvoří kliku. Dva pravé sousedy vrcholu v_i, které nejsou spojeny hranou, si označme v_j a v_k. Všechny vrcholy v_{n-1}, ..., v_i byly do posloupnosti přidány díky tomu, že jeden z jejich sousedů (tzv. *předchůdce*) se již v posloupnosti nacházel. Nechť v_l je nejbližší společný předchůdce pro vrcholy v_j a v_k. Pak jsme ale nalezli indukovaný cyklus $(v_i, v_j, \ldots, v_l, \ldots, v_k, v_i)$, který má délku ≥ 4, což je spor s tím, že graf G je chordální.

□

MCS algoritmus je implementován pomocí prioritní fronty, a to pomocí Fibonacciho haldy.

IntegerInvariants.cs zjišťuje o grafu následující vlastnosti:

- počet vrcholů,
- počet hran,
- počet komponent (se získáním komponent souvislosti a informace, zda je graf souvislý)
- circuit rank (se získáním počtu komponent),
- girth (se získáním informace, zda graf obsahuje cyklus),
- Cayleyho formuli,
- maximální, minimální a průměrný stupeň vrcholu (se získáním skóre grafu).

Počet vrcholů a počet hran není potřeba vypočítávat, při vytvoření grafu jsou tyto hodnoty známy a při modifikacích se adekvátně aktualizují.

Girth se počítá tak, že se na každém vrcholu provede BFS, díky čemuž se nalezne nejmenší cyklus, který vede přes daný vrchol. Protože BFS pro jednotlivé vrcholy jsou nezávislé, tak se využívá paralelního výpočtu. Časová složitost je $\mathcal{O}(|V| \times (|V| + |E|))$.

SequencesPolynomialsOthers.cs zjišťuje o grafu následující:

- skóre grafu,
- libovolnou kostru grafu,
- perfektní eliminační schéma (se získáním informace, zda je graf chordální),
- artikulace a mosty[6].

Skóre grafu se spočítá projitím všech vrcholů a spočítáním jejich sousedů. Volající si může vyžádat seznam stupňů bez vrcholů, seznam vrcholů bez stupňů (využívá metoda LF), nebo obojího, jejichž případné setřídění podle skóre ovlivňuje volající. Cílem je neplýtvat čas na setřídění skóre, pokud to není potřeba.

Kostra grafu se získá průchodem do šířky (BFS).

Skóre grafu i kostra grafu se dá zjistit s časovou složitostí $\mathcal{O}(|V| + |E|)$, neboť se projde celý graf jednou.

GraphClass

GraphClass je statická třída, která pro daný graf zjistí jeho třídu. Testují se následující třídy:

- úplný graf (*completeGraph*),
- strom (*treeGraph*),
- kružnice (*cycleGraph*),
- bipartitní graf (*bipartiteGraph*) a
- úplný bipartitní graf (*completeBipartiteGraph*).

Volající má možnost otestovat všechny třídy po jedné nebo využít metodu *GetGraphClass(IGraphInterface)*, která vrátí rovnou danou třídu. Pokud graf neodpovídá žádné třídě, tak funkce vrátí *None*. Na druhou stranu, graf může odpovídat více třídám (např. stromu a bipartitnímu grafu), pak se upřednostňuje nejspecifičtější třída, tj. třída, která se vyskytuje v seznamu nejvýše (viz seznam výše).

Zjištění, zda graf je úplný, je triviální, stačí ověřit zda $|\mathrm{E}| = \binom{|V|}{2}$, což se zvládne s konstantní časovou složitostí.

Pro strom stačí díky větě 4 ověřit, zda je graf souvislý a $|\mathrm{V}| = |\mathrm{E}| + 1$, což se zvládne s časovou složitostí $\mathcal{O}(|V| + |E|)$.

Pro kružnici se zjišťuje, zda je graf souvislý, regulární a maximální (= minimální) stupeň grafu je 2, což se zvládne také s časovou složitostí $\mathcal{O}(|V| + |E|)$.

Bipartitní graf se testuje pomocí BFS, který případně rozdělí vrcholy do dvou partit. Pro úplný bipartitní graf se navíc zjišťuje, zda vrcholy v jednotlivých partitách mají správné stupně. Případné partity se ukládají do proměnných *firstPartite* a *secondPartite* v *GraphProperty*. Obojí má časovou složitost $\mathcal{O}(|V| + |E|)$.

GraphOperation

Jedná se o statickou třídu, která umí provádět následující operace s grafy:

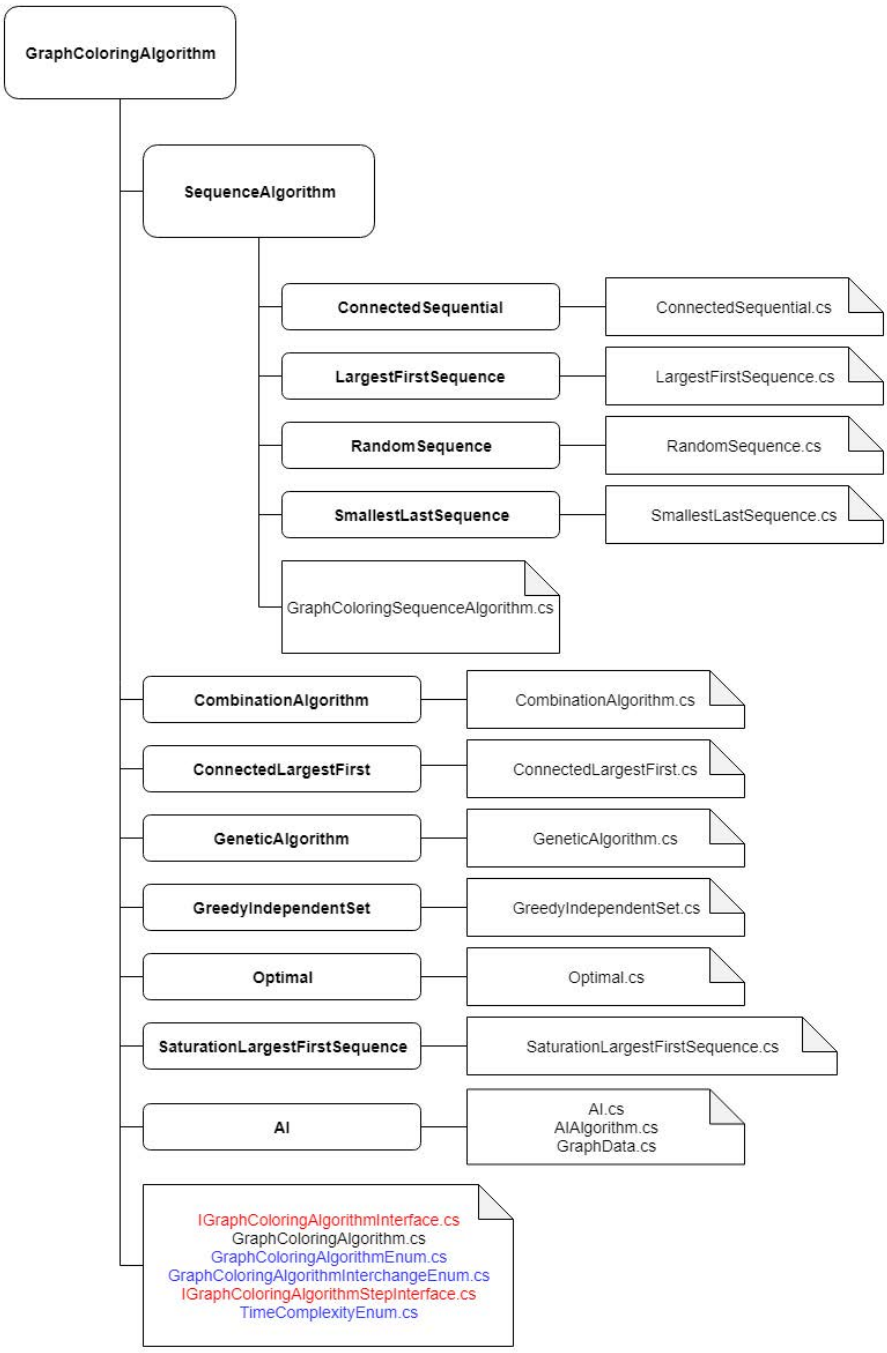
- vytvořit komplementární graf,
- vytvořit hranový graf,
- vytvořit podgraf pro danou množinu vrcholů a
- zkopírovat graf.

Všechny operace uvedené výše vytvářejí nový graf, který není nijak svázán s grafem, ze kterého byl vytvářen. Operace jsou implementovány intuitivně a není potřeba je popisovat. Všechny operace mají časovou složitost $\mathcal{O}(|V| + |E|)$, neboť projdou graf jenom jednou.

6.2.2 GraphColoringAlgorithm

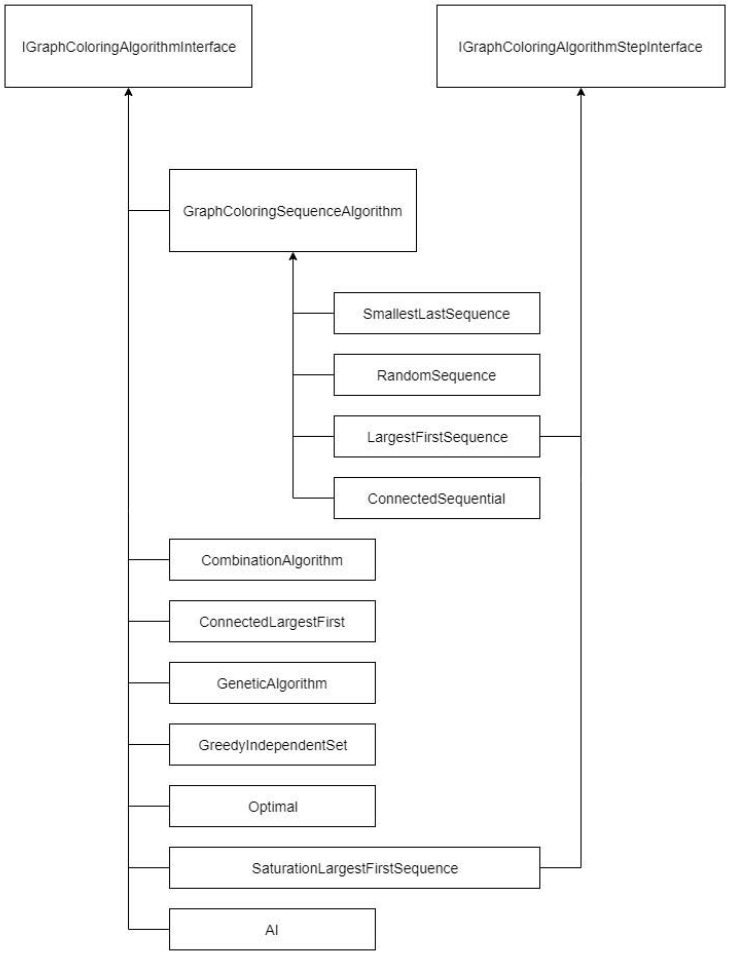
Namespace *GraphColoringAlgorithm* obsahuje třídy s algoritmy pro obarvování.

Hierarchie namespacu *GraphColoringAlgorithm* je zobrazena na obr. 6.5.



Obrázek 6.5: Hierarchie namespacu *GraphColoringAlgorithm*

Hierarchie dědění v namespacu *GraphColoringAlgorithm* je zobrazena na obr. 6.6.



Obrázek 6.6: Hierarchie dědění v namespacu *GraphColoringAlgorithm*

Každý algoritmus musí přímo, nebo nepřímo implementovat interface *IGraphColoringAlgorithmInterface*, jehož jednou veřejnou metodou je *Color()*, která obarví daný graf. Algoritmy, které mohou být použity pro kombinační algoritmus, musí navíc implementovat i interface *IGraphColoringAlgorithmStepInterface*, který obsahuje jednu veřejnou metodu, a to metodu *Step()*, která obarví pouze jeden vrchol v částečně obarveném grafu.

Sekvenční algoritmy dědí ze třídy *GraphColoringSequenceAlgorithm*, která implementuje metodu *Color()* tak, že daný graf obarví hladově podle posloupnosti vrcholů, kterou vytvoří daná zděděná třída (pomocí metody *CreateVertexSequence()*).

Každý algoritmus obsahuje atributy *name* a *timeComplexity*. Atribut *name* využívá hlavně *FE* a atribut *timeComplexity* používá metoda *AI*, a to v případě, když pro nějaký graf bylo shledáno několik algoritmů "stejně" dobrých, tak druhým kritériem pro výběr nejlepšího algoritmu pro daný graf je právě časová složitost algoritmu.

Konstruktor genetického algoritmu bere navíc dva atributy, které se týkají velikosti populace a počtu iterací algoritmu. Parametr *populationSize* určuje velikost

populace genetického algoritmu, defaultně je nastaveno na konstantu *10*. Druhý parametr *exponentCountOfIteration* = e určuje počet iterací, a to předpisem $|V|^e$.

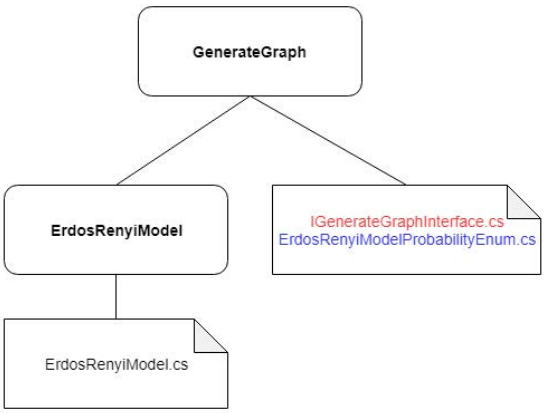
AI algoritmus si při první inicializaci načte všechny modely pro jednotlivé algoritmy. Tyto modely načítá ze složky *GraphColoringAlgorithm\AIModels*, kde soubor s modelem musí mít název tvaru *model-název algoritmu podle GraphColoringAlgorithmEnum.zip* (např. *model-smallestLastSequenceInterchange.zip*). Pokud není načten žádný model, tak se pro obarvení grafu používají dva algoritmy, jejichž výběr záleží na počtu vrcholů daného grafu. Pro grafy s počtem vrcholů < 50 se používá genetický algoritmus s exponentem 2 a pro větší grafy se používá SLIE algoritmus. Tyto algoritmy byly vybrány na základě experimentů, ze kterých vyšly jako nejlepší barvící algoritmy pro dané grafy. Pokud byly načteny nějaké modely, tak se na základě grafových vlastností zjistí, zda daný algoritmus má potenciál nejlépe obarvit daný graf. Pokud těchto algoritmů je více, tak se vybírá na základě přesnosti modelu (*negative recall* a *positive precision*) a až následně se případně vybírá podle časových složitostí jednotlivých algoritmů. Pokud graf lze optimálně obarvit pomocí nějakého konkrétního algoritmu, tak se nevyhodnocují dané modely a rovnou se použije daný algoritmus.

Algoritmy jsou implementovány podle kapitoly 3.

6.2.3 GenerateGraph

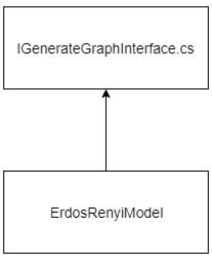
Namespace *GenerateGraph* obsahuje třídy, které slouží ke generování náhodných grafů.

Hierarchie namespacu *GenerateGraph* je zobrazena na obr. 6.7.



Obrázek 6.7: Hierarchie namespacu *GenerateGraph*

Hierarchie dědění v namespacu *GenerateGraph* je zobrazena na obr. 6.8.



Obrázek 6.8: Hierarchie dědění v namespacu *GenerateGraph*

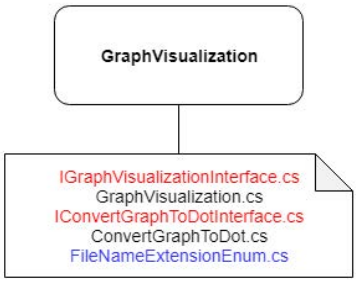
Každá třída pro generování náhodných grafů musí implementovat interface *IGenerateGraphInterface*, jehož jedinou veřejnou metodou je *GenerateGraph()*.

V celém projektu se generují náhodné grafy pomocí *Erdos-Rényi model*u.

6.2.4 GraphVisualization

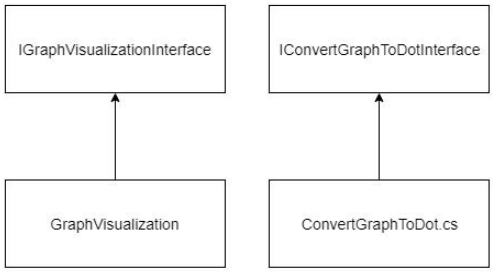
Namespace *GraphVisualization* obsahuje třídy, které slouží k převodu grafů do bitmapy.

Hierarchie namespacu *GraphVisualization* je zobrazena na obr. 6.9.



Obrázek 6.9: Hierarchie namespacu *GraphVisualization*

Hierarchie dědění v namespacu *GraphVisualization* je zobrazena na obr. 6.10.



Obrázek 6.10: Hierarchie dědění v namespacu *GraphVisualization*

Třída *GraphVisualization* nejdříve odfiltruje grafy, které se nemají vykreslovat, tj. grafy s počtem vrcholů > 50 nebo obarvené grafy s počtem barev > 14. Pokud daný graf splňuje podmínky pro vykreslení, tak se zavolá třída *ConvertGraphToDot*, která daný graf převede do obrázku, jehož formát vybírá *GraphVisualization* (zpravidla *.jpg*), pomocí knihovny **DOT**. Obrázek, který vrátí *Con-*

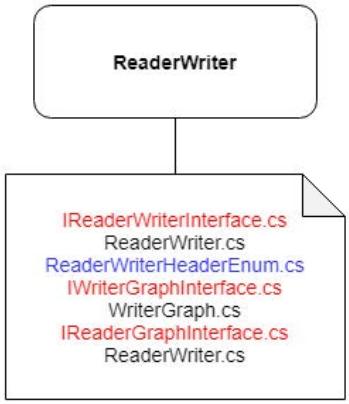
vertGraphToDot, se převede do bitmapy. Příjemce poté po provolání metody *GetImage()* dostane bitmapový obrázek s daným grafem, nebo s infohláškou, proč daný graf nešlo vykreslit.

Knihovna *DOT* se vyskytuje ve složce *GraphVisualization\DOT*.

Poznámka. Knihovnu *DOT* jsem si vybral z několika důvodů. Prvním byla dřívější zkušenost s touto knihovnou a druhým důvodem byly rozmanité možnosti, které knihovna nabízí (vykreslování různých tvarů, barev, šířky okraje apod.), které jsem zvětšiny využil.

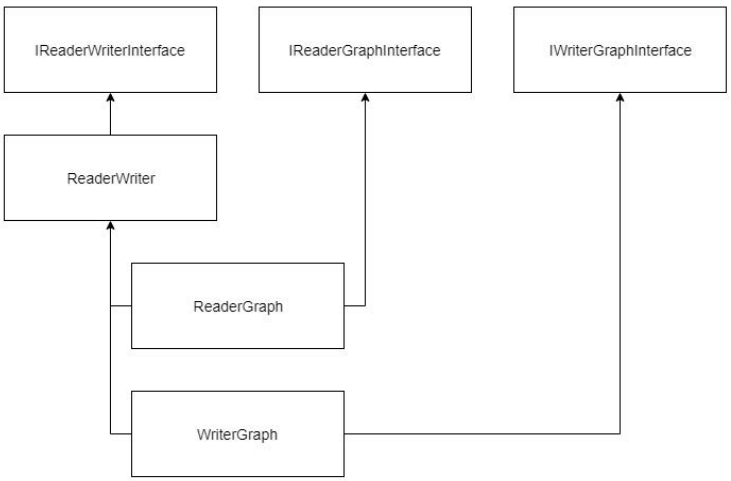
6.2.5 ReaderWriter

Namespace *ReaderWriter* obsahuje třídy, které slouží pro práci se soubory. Hierarchie namespacu *ReaderWriter* je zobrazena na obr. 6.11.



Obrázek 6.11: Hierarchie namespacu *ReaderWriter*

Hierarchie dědění v namespacu *ReaderWriter* je zobrazena na obr. 6.12.



Obrázek 6.12: Hierarchie dědění v namespacu *ReaderWriter*

Třída *ReaderWriter* obsahuje standardní metody pro práci se soubory, jako je vytvoření souboru, smazání souboru, zvalidování absolutní cesty apod.

Třída *ReaderGraph* umí číst soubory s formátem *.graph* a z daného souboru umí případně vytvořit graf.

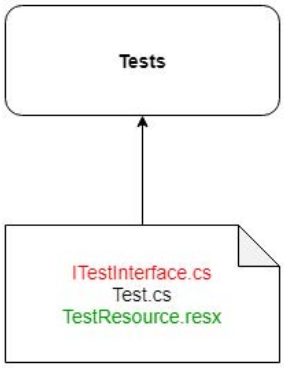
Třída *WriterGraph* umí vytvářet soubory s formátem *.graph*, kam umí uložit graf a obarvení grafu. Před tím než uloží obarvení grafu pomocí nepravděpodobnostního algoritmu do souboru, tak zkontroluje, zda se již v souboru dané obarvení nevyskytuje.

6.2.6 Tests

Každá komponenta aplikace může využívat testování, které se nachází v namespacu *Tests*.

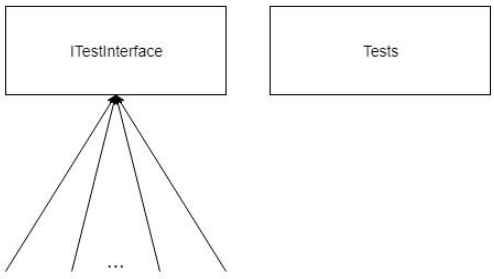
Komponenty, které využívají testování: práce se soubory (ReaderWriter, Reader, Writer), grafy a grafové vlastnosti (Component, DegreeSequenceList, Cycle, SpanningTree, ColoredGraph, BridgesCutVertices, Chordal, Class), grafové modifikace (GraphModification), grafové operace (Complement, Copy, SubGraph, LineGraph), grafová vizualizace (ConvertGraphToDot), generování náhodných grafů pomocí *Erdos-Rényi model*u (ErdosRenyiModel) a několik algoritmů (LF, SL, Optimal, SLF, CS, GIS, Combination, Genetic).

Hierarchie namespacu *Tests* je zobrazena na obr. 6.13.



Obrázek 6.13: Hierarchie namespacu *Tests*

Hierarchie dědění v namespacu *Tests* je zobrazena na obr. 6.14.



Obrázek 6.14: Hierarchie dědění v namespacu *Tests*

Třída implementující interface *ITestInterface* musí implementovat jednu metodu *Test()* a vlastnost *GetPath()*. Metoda *Test()* slouží k otestování dané komponenty a vrací *StringBuilder*, který obsahuje výstup testu, který může být libo-

volný. Vlastnost *GetPath()* vrací umístění souboru, kam se bude případně ukládat report z testu.

Třída *Tests* umožňuje testovat více testů pohromadě. Podmínkou je, aby dané testy měly soubory se správnými výstupy v *TestResource.resx* a navíc, aby dané testy byly uloženy v dictionary *testsDictionary*. Testování má dva možné výstupy. Prvním z nich je výpis na konzoli, kde se vždy vypíše název testu a informace, zda test proběhl úspěšně, tzn. že výstup daného testu je stejný s obsahem daného souboru v *TestResource.resx*. Druhým možným výstupem je vypsání výstupu z testu do souboru, jehož cesta se bere právě z vlastnosti *GetPath()*.

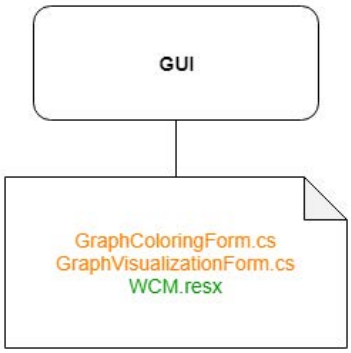
Třída *Tests* se spouští vždy při startu aplikace s výpisem do konzole, aby se zkontrolovalo, zda je aplikace ve funkčním stavu, a proto by jednotlivé testy neměly být časově náročnější.

Testy mimo jiné mohou sloužit i vývojářům, aby předešli případným defektům při vyvíjení nové komponenty nebo opravě stávající komponenty.

6.3 Frontend - GraphColoring

FE se nachází v namespacu *GUI*, který obsahuje formulář *GraphColoring*, *GraphVisualization* a resource *WCM*, který obsahuje veškeré textace.

Hierarchie namespacu *GUI* je zobrazena na obr. 6.15.



Obrázek 6.15: Hierarchie namespacu *GUI*

Formulář *GraphColoring* obsahuje zpravidla metody pro zpracování událostí, které jsou implementovány intuitivně. Nutno jenom podotknout, že *FE* se stará za případné rozdělení nesouvislého grafu do seznamu souvislých komponent. Neboť některé barvící algoritmy nejsou schopny obarvit validně nesouvislý graf! Navíc všechny složité výpočty jako načtení grafu, obarvení grafu, vygenerování nového grafu apod. se provádí v novém vlákně *coreThread*, a to z toho důvodu, aby klient mohl případně použít tlačítko *Reset* a navíc aby při časově složitějších operacích nedocházelo k neodpovídání aplikace.

6.4 Backend - GraphColoringConsole

BE se stará o následující věci:

- generování a obarvování náhodných grafů, které ukládá do souboru nebo přímo do databáze (složka *GenerateGraphs* a *Database*),
- vkládání grafů ze souborů do databáze (složka *GenerateGraphs* a *Database*),
- vytváření modelů pro AI (složka *ML*),
- převádění grafů z formátu *.col* na *.graph* (složka *ConvertGraphs*),
- měření časových složitostí jednotlivých algoritmů (složka *GenerateTimeComplexity*) a
- obarvování grafů ze souborů (složka *ColorGraphs*).

Třídy z namespaců *GenerateGraphs*, *ConvertGraphs*, *GenerateTimeComplexity* a *ColorGraphs* jsou implementovány intuitivně a není potřeba je popisovat.

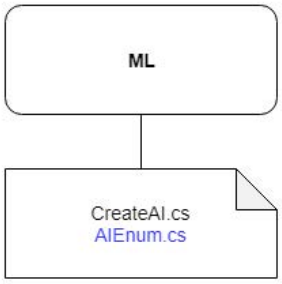
6.4.1 Database

Namespace *Database* obsahuje pouze jednu stejnojmennou třídu, která obsahuje metody pro navázání spojení s databází a metody pro provolání databázových funkcí a procedur.

Jednou z nejdůležitějších metod je *SaveDataFromDatabaseToFile(string, GraphColoringAlgorithmEnum)*, která pro daný algoritmus uloží data určená pro ML do souboru, jehož umístění se předává v prvním parametru metody. Metoda navíc vybírá data tak, aby počet pozitivních a negativních příkladů byl stejný.

6.4.2 ML

Namespace *ML* obsahuje třídu pro vytváření modelů pro AI.
Hierarchie namespacu *ML* je zobrazena na obr. 6.16.



Obrázek 6.16: Hierarchie namespacu *ML*

K vytváření modelů se využívá knihovna *ML.NET*.
Vytvoření modelu se skládá z 5 kroků:

1. vytvoření dat pro ML,

2. transformace dat,
3. vytvoření modelu,
4. ohodnocení vytvořeného modelu a
5. uložení modelu.

Data se získávají z databáze pomocí metody *SaveDataFromDatabaseToFile*, která se následně rozdělí na trénovací a testovací data s určitým poměrem, který je defaultně nastaven na 7:3. K vytvoření modelu je potřeba alespoň 10 000 dat.

U transformace dat se definují featury, do kterých budou data převedena. Featury jsou následující:

- GraphClass,
- EulerianGraph,
- IsRegular,
- IsCyclic,
- IsChordal,
- CountVertices,
- CountEdges,
- CountCutVertices,
- Girth,
- Dense,
- MinimumVertexDegree,
- MaximumVertexDegree a
- AverageVertexDegree.

Tedy featurami jsou téměř všechny vlastnosti grafu, kromě počtu mostů, které jsou korelované s počtem artikulací. Tyto featury byly získány na základě maximalizace *negative recall* a *positive precision*. Výstupem modelu je booleovská hodnota, která říká, zda daný algoritmus je vhodný pro daný graf.

Důvod pro maximalizaci *negative recall* a *positive precision* je ten, že chceme minimalizovat chybu typu I (false positive), tedy případ kdy model predikuje, že algoritmus je vhodný na obarvení daného grafu, což není pravda.

K vytvoření modelu se využívá metoda *fast tree*, která se ukázala jako nejlepší metoda.

Pro vytvořený model se provádí několik ohodnocení:

- přesnost,
- log-loss,

- log-loss reduction,
- F1,
- AUC,
- negative precision,
- **negative recall**,
- **positive precision**,
- positive recall,

což je vše, co nám může nabídnout knihovna *ML.NET*.

Po ohodnocení modelu dojde k uložení modelu do složky *Data*, jehož název má tvar *model-dd-MM-yyyy-HH-mm-ss-název algoritmu podle GraphColoringAlgorithmEnum.zip* (např. *model-04-05-2019-03-17-42-geneticAlgorithm.zip*).

Pro každý algoritmus se vytvoří jeden model.

Poznámka. Knihovnu *ML.NET* jsem si vybral z jediného důvodu, jedná se o poměrně novou knihovnu, s kterou jsem ještě nepracoval a cílem bylo si tuto knihovnu vyzkoušet.

7. Experimenty

Provedeme tři typy experimentů na porovnání:

1. standardní metody interchange a rozšířené metody interchange
2. rozšířené metody interchange a rozšířené metody interchange s přebarvením K3 a
3. CLF algoritmu s ostatními algoritmy.

A jeden experiment na změření efektivity meta-algoritmu.
První experiment bude zaměřený na obarvování grafů z *DIMACS* kolekce grafů, druhý na dobu trvání jednotlivých algoritmů a třetí na obarvování náhodných grafů.

7.1 Obarvování grafů z DIMACS kolekce grafů

Existuje kolekce grafů nazvaná *DIMACS*[7, 8], která se mimo jiné používá i pro ohodnocení barvících algoritmů. Tato kolekce obsahuje přes 80 grafů.

Grafy byly obarveny pravděpodobnostními algoritmy (tj. RS, RSI, RSIE, RSIEK3, Gen1 a Gen2) 10x, kdy výsledný počet barev byl spočítán jako aritmetický průměr z nejmenšího a největšího počtu použitých barev se zaokrouhlením nahoru.
Modře zvýrazněný počet barev u daného algoritmu znamená, že daný algoritmus obarvil graf s nejmenším počtem barev vzhledem k ostatním algoritmům.

Poznámka. Některé grafy nebyly pro větší časovou složitost obarveny genetickým algoritmem s exponentem 2 a algoritmy využívající rozšířený interchange s přebarvením K3.

Poznámka. Tento experiment zabral tři dny výpočetního času na jednom počítači.

Tabulka 7.1 obsahuje grafy z *DIMACS* kolekce grafů, kde je uveden počet vrcholů, počet hran, hustota grafu a chromatické číslo grafu.

Tabulka 7.1: Grafy z *DIMACS* kolekce grafů.

Název grafu G	|V|	|E|	g(G)	$\chi(G)$
myciel3	11	20	0,36	4
myciel4	23	71	0,28	5
queen5_5	25	160	0,53	5
queen6_6	36	290	0,46	7
2-Insertions_3	37	72	0,11	4
myciel5	47	236	0,28	5
queen7_7	49	476	0,40	7
3-Insertions_3	56	110	0,07	4
queen8_8	64	728	0,36	9
jean	80	254	0,08	10
queen9_9	81	1056	0,33	10
david	87	406	0,11	11
mug88_1	88	146	0,04	4
mug88_25	88	146	0,04	4
myciel6	95	755	0,17	7
queen8_12	96	1368	0,30	12
mug100_1	100	166	0,03	4
mug100_25	100	166	0,03	4
queen10_10	100	1470	0,30	11
games120	120	638	0,09	9
queen11_11	121	1980	0,27	11
DSJC125.1	125	736	0,09	5
DSJC125.5	125	3891	0,50	17
DSJC125.9	125	6961	0,90	44
miles500	128	1170	0,14	20
miles750	128	2113	0,26	31
miles1000	128	3216	0,40	42
anna	138	493	0,05	11
queen12_12	144	2596	0,25	12
queen13_13	169	3328	0,23	13
mulsol.i.3	184	3916	0,23	31
mulsol.i.4	185	3946	0,23	31
mulsol.i.5	186	3973	0,23	31
mulsol.i.2	188	3885	0,22	31
myciel7	191	2360	0,13	8
queen14_14	196	4186	0,22	14
mulsol.i.1	197	3925	0,20	49
zeroin.i.3	206	3540	0,17	30
zeroin.i.2	211	3541	0,16	30
zeroin.i.1	211	4100	0,19	49
queen15_15	225	5180	0,21	15
DSJC250.1	250	3218	0,10	≤ 8
DSJC250.5	250	15668	0,50	≤ 28
DSJC250.9	250	27897	0,90	≤ 72
queen16_16	256	6320	0,19	16

Pokračování tabulky 7.1				
Název grafu G	|V|	|E|	g(G)	$\chi(G)$
fpsol2.i.1	496	11654	0,09	65
fpsol2.i.3	425	8688	0,10	30
le450_5a	450	5714	0,06	5
le450_5b	450	5734	0,06	5
le450_15a	450	8168	0,08	15
le450_15b	450	8169	0,08	15
le450_25a	450	8260	0,08	25
le450_25b	450	8263	0,08	25
le450_5d	450	9757	0,10	5
le450_5c	450	9803	0,10	5
le450_15c	450	16680	0,17	15
le450_15d	450	16750	0,17	15
le450_25c	450	17343	0,17	25
le450_25d	450	17425	0,17	25
fpsol2.i.2	451	8691	0,09	30
DSJR500.1	500	3555	0,03	12
DSJR500.5	500	58862	0,47	122
DSJC500.5	500	62624	0,50	≤ 47
DSJC500.1	500	12458	0,10	≤ 12
DSJC500.9	500	112437	0,90	≤ 126
DSJR500.1c	500	121275	0,97	84
inithx.i.3	621	13969	0,07	31
inithx.i.2	645	13979	0,07	31
inithx.i.1	864	18707	0,05	54
qg.order30	900	26100	0,06	30
latin_square_10	900	307350	0,76	≤ 97
wap05	905	43081	0,11	50
wap06	947	43571	0,10	40
DSJC1000.1	1000	49629	0,10	≤ 20
DSJC1000.5	1000	249826	0,50	≤ 82
DSJC1000.9	1000	449449	0,90	≤ 222
qg.order40	1600	62400	0,05	40
wap07	1809	103368	0,06	≤ 41
wap08	1870	104176	0,06	≤ 42
qg.order60	3600	212400	0,03	60

7.1.1 Tabulky s obarveními

Tabulka 7.2: Obarvení grafů z *DIMACS* kolekce - 1. část

Název grafu G	RS	RSI	RSIE	RSIEK3	CS	SLF
myciel3	4	4	4	4	4	4
myciel4	6	5	5	5	5	5
queen5_5	7	6	6	6	8	5
queen6_6	10	9	8	8	10	9
2-Insertions_3	4	4	4	4	4	4
myciel5	6	6	6	6	6	6
queen7_7	11	10	9	9	11	10
3-Insertions_3	4	4	4	4	4	4
queen8_8	13	11	11	11	14	12
jean	11	10	10	10	10	10
queen9_9	13	13	12	12	15	13
david	12	11	11	11	11	11
mug88_1	5	4	4	4	4	4
mug88_25	5	4	4	4	4	4
myciel6	8	7	7	7	7	7
queen8_12	16	14	13	13	15	14
mug100_1	5	4	4	4	4	4
mug100_25	4	4	4	4	4	4
queen10_10	16	14	13	13	17	13
games120	9	9	9	9	9	9
queen11_11	17	15	15	14	18	16
DSJC125.1	8	7	7	7	7	7
DSJC125.5	26	22	21	21	24	22
DSJC125.9	57	51	49	47	55	53
miles500	23	21	21	20	21	20
miles750	34	32	32	32	34	31
miles1000	44	43	43	43	46	42
anna	12	11	11	11	12	11
queen12_12	18	16	16	16	21	16
queen13_13	19	17	17	16	21	17
mulsol.i.3	32	31	31	31	31	31
mulsol.i.4	32	31	31	31	31	31
mulsol.i.5	31	31	31	31	31	31
mulsol.i.2	32	31	31	31	31	31
myciel7	9	8	8	8	8	8
queen14_14	21	19	18	18	23	19
mulsol.i.1	49	49	49	49	49	49
zeroin.i.3	31	30	30	-	30	30
zeroin.i.2	31	30	30	-	30	30
zeroin.i.1	50	49	49	-	49	49
queen15_15	22	20	19	19	22	20
DSJC250.1	13	12	10	10	13	10
DSJC250.5	42	38	36	35	42	39

Pokračování tabulky 7.2						
Název grafu G	RS	RSI	RSIE	RSIEK3	CS	SLF
DSJC250.9	98	88	84	-	96	96
queen16_16	23	21	20	20	26	21
fpsol2.i.1	65	65	65	-	65	65
fpsol2.i.3	31	31	30	-	32	30
le450_5a	14	12	10	-	12	10
le450_5b	14	12	10	-	14	9
le450_15a	21	19	18	-	22	17
le450_15b	22	19	18	-	21	17
le450_25a	29	26	26	-	27	25
le450_25b	28	26	26	-	27	25
le450_5d	16	12	9	-	15	13
le450_5c	17	13	9	-	16	10
le450_15c	31	28	26	-	30	24
le450_15d	31	28	26	-	30	24
le450_25c	36	33	31	-	36	29
le450_25d	36	32	31	-	36	28
fpsol2.i.2	31	30	31	-	32	30
DSJR500.1	15	13	13	-	14	13
DSJR500.5	145	132	133	-	137	127
DSJC500.5	72	66	62	-	73	67
DSJC500.1	20	18	16	-	19	16
DSJC500.9	177	158	151	-	175	165
DSJR500.1c	111	96	94	-	107	92
inithx.i.3	31	31	31	-	32	31
inithx.i.2	32	31	31	-	32	31
inithx.i.1	54	54	54	-	55	54
qg.order30	34	30	30	-	32	32
latin_square_10	149	132	123	-	204	130
wap05	55	51	51	-	57	51
wap06	55	48	48	-	56	51
DSJC1000.1	32	30	26	-	33	27
DSJC1000.5	128	119	109	-	127	117
DSJC1000.9	321	290	277	-	324	303
qg.order40	45	40	40	-	64	43
wap07	60	54	51	-	61	48
wap08	59	54	52	-	68	47
qg.order60	66	60	60	-	64	63

Tabulka 7.3: Obarvení grafů z *DIMACS* kolekce - 2. část

Název grafu G	LF	LFI	LFIE	LFIEK3	GIS
myciel3	4	4	4	4	4
myciel4	5	5	5	5	5
queen5_5	7	7	5	7	5
queen6_6	9	8	8	8	9
2-Insertions_3	5	4	4	4	4
myciel5	6	6	6	6	6
queen7_7	12	10	9	9	10
3-Insertions_3	5	4	4	4	4
queen8_8	13	11	10	11	12
jean	10	10	10	10	11
queen9_9	16	12	12	12	13
david	11	11	11	11	14
mug88_1	4	4	4	4	4
mug88_25	4	4	4	4	4
myciel6	7	7	7	7	7
queen8_12	15	14	12	13	16
mug100_1	4	4	4	4	4
mug100_25	4	4	4	4	4
queen10_10	17	15	13	13	15
games120	9	9	9	9	9
queen11_11	18	15	14	15	16
DSJC125.1	7	7	6	7	8
DSJC125.5	24	23	20	21	24
DSJC125.9	53	51	48	49	54
miles500	20	22	20	20	22
miles750	32	33	31	32	33
miles1000	43	46	42	43	46
anna	11	11	11	11	12
queen12_12	19	16	15	16	17
queen13_13	22	18	15	16	19
mulsol.i.3	31	31	31	31	31
mulsol.i.4	31	31	31	31	31
mulsol.i.5	31	31	31	31	32
mulsol.i.2	31	31	31	31	31
myciel7	8	8	8	8	8
queen14_14	23	18	17	18	20
mulsol.i.1	49	49	49	49	49
zeroin.i.3	30	30	30	-	30
zeroin.i.2	30	30	30	-	30
zeroin.i.1	49	50	49	-	51
queen15_15	25	20	19	19	21
DSJC250.1	11	11	10	10	13
DSJC250.5	41	40	34	36	36
DSJC250.9	91	88	82	-	90
queen16_16	26	22	20	20	21

Pokračování tabulky 7.3					
Název grafu G	LF	LFI	LFIE	LFIEK3	GIS
fpsol2.i.1	65	66	65	-	65
fpsol2.i.3	30	31	30	-	34
le450_5a	11	12	9	-	11
le450_5b	12	11	9	-	10
le450_15a	19	20	16	-	22
le450_15b	18	21	16	-	22
le450_25a	26	27	25	-	34
le450_25b	26	28	25	-	30
le450_5d	13	14	8	-	7
le450_5c	12	13	7	-	8
le450_15c	26	30	23	-	29
le450_15d	26	30	23	-	30
le450_25c	30	36	28	-	38
le450_25d	30	35	27	-	37
fpsol2.i.2	30	31	30	-	34
DSJR500.1	14	14	12	-	15
DSJR500.5	134	152	125	-	165
DSJC500.5	72	69	60	-	65
DSJC500.1	18	18	15	-	17
DSJC500.9	171	158	152	-	158
DSJR500.1c	99	100	92	-	104
inithx.i.3	31	31	31	-	36
inithx.i.2	31	31	31	-	36
inithx.i.1	54	55	54	-	54
qg.order30	32	30	30	-	30
latin_square_10	191	132	128	-	128
wap05	50	54	50	-	57
wap06	47	52	42	-	56
DSJC1000.1	29	30	25	-	27
DSJC1000.5	122	120	107	-	108
DSJC1000.9	314	294	273	-	278
qg.order40	64	40	40	-	40
wap07	51	60	45	-	63
wap08	49	59	44	-	62
qg.order60	64	60	60	-	61

Tabulka 7.4: Obarvení grafů z *DIMACS* kolekce - 3. část

Název grafu G	SL	SLI	SLIE	SLIEK3	combination
myciel3	4	4	4	4	4
myciel4	5	5	5	5	5
queen5_5	7	5	5	5	7
queen6_6	10	9	8	8	9
2-Insertions_3	4	4	4	4	4
myciel5	6	6	6	6	6
queen7_7	12	9	9	9	10
3-Insertions_3	4	4	4	4	4
queen8_8	13	10	10	10	13
jean	10	10	10	10	10
queen9_9	14	12	11	11	15
david	11	11	11	11	11
mug88_1	4	4	4	4	4
mug88_25	4	4	4	4	4
myciel6	7	7	7	7	7
queen8_12	17	13	12	12	16
mug100_1	4	4	4	4	4
mug100_25	4	4	4	4	4
queen10_10	17	12	13	13	15
games120	9	9	9	9	9
queen11_11	17	14	14	14	17
DSJC125.1	8	6	6	6	6
DSJC125.5	25	21	20	20	23
DSJC125.9	53	48	48	46	56
miles500	20	20	20	20	20
miles750	31	31	31	31	32
miles1000	42	42	42	42	43
anna	11	11	11	11	11
queen12_12	20	16	15	15	18
queen13_13	22	17	16	16	19
mulsol.i.3	31	31	31	31	31
mulsol.i.4	31	31	31	31	31
mulsol.i.5	31	31	31	31	31
mulsol.i.2	31	31	31	31	31
myciel7	8	8	8	8	8
queen14_14	24	19	17	17	22
mulsol.i.1	49	49	49	49	49
zeroin.i.3	30	30	30	-	30
zeroin.i.2	30	30	30	-	30
zeroin.i.1	49	49	49	-	50
queen15_15	24	19	18	18	23
DSJC250.1	13	10	9	9	10
DSJC250.5	41	38	34	34	39
DSJC250.9	95	85	84	-	97
queen16_16	25	20	19	19	24

Pokračování tabulky 7.4					
Název grafu G	SL	SLI	SLIE	SLIEK3	combination
fpsol2.i.1	65	65	65	-	65
fpsol2.i.3	30	30	30	-	30
le450_5a	11	10	9	-	12
le450_5b	11	10	9	-	13
le450_15a	18	16	16	-	21
le450_15b	17	16	16	-	20
le450_25a	25	25	25	-	26
le450_25b	25	25	25	-	26
le450_5d	12	9	7	-	14
le450_5c	14	6	8	-	15
le450_15c	26	24	23	-	30
le450_15d	27	24	24	-	31
le450_25c	32	29	28	-	35
le450_25d	30	28	28	-	37
fpsol2.i.2	30	30	30	-	30
DSJR500.1	12	12	12	-	15
DSJR500.5	127	126	124	-	179
DSJC500.5	71	64	60	-	73
DSJC500.1	19	16	15	-	19
DSJC500.9	174	156	152	-	183
DSJR500.1c	104	93	92	-	103
inithx.i.3	31	31	31	-	31
inithx.i.2	31	31	31	-	31
inithx.i.1	54	54	54	-	55
qg.order30	32	30	30	-	41
latin_square_10	204	130	121	-	161
wap05	51	50	50	-	56
wap06	44	43	41	-	56
DSJC1000.1	30	27	25	-	31
DSJC1000.5	123	115	107	-	127
DSJC1000.9	315	283	272	-	320
qg.order40	64	40	40	-	53
wap07	46	44	43	-	56
wap08	45	44	43	-	56
qg.order60	64	60	60	-	80

Tabulka 7.5: Obarvení grafů z *DIMACS* kolekce - 4. část

Název grafu G	CLF	CLFI	CLFIE	CLFIEK3	Gen1	Gen2
myciel3	4	4	4	4	4	4
myciel4	5	5	5	5	5	5
queen5_5	5	5	5	5	5	5
queen6_6	9	8	8	8	8	8
2-Insertions_3	4	4	4	4	4	4
myciel5	6	6	6	6	6	6
queen7_7	13	10	9	9	9	8
3-Insertions_3	4	4	4	4	4	4
queen8_8	14	10	10	10	11	10
jean	10	10	10	10	10	10
queen9_9	15	12	11	11	12	12
david	11	11	11	11	11	11
mug88_1	4	4	4	4	4	4
mug88_25	4	4	4	4	4	4
myciel6	7	7	7	7	7	7
queen8_12	17	13	13	13	13	13
mug100_1	4	4	4	4	4	4
mug100_25	4	4	4	4	4	4
queen10_10	16	13	12	12	13	13
games120	9	9	9	9	9	9
queen11_11	18	14	14	14	15	14
DSJC125.1	7	6	6	6	7	6
DSJC125.5	24	21	20	20	22	22
DSJC125.9	54	50	46	47	51	50
miles500	20	20	20	20	20	20
miles750	31	31	31	31	31	31
miles1000	42	42	42	42	42	42
anna	11	11	11	11	11	11
queen12_12	20	16	15	15	16	16
queen13_13	21	17	16	16	17	17
mulsol.i.3	31	31	31	31	31	31
mulsol.i.4	31	31	31	31	31	31
mulsol.i.5	31	31	31	31	31	31
mulsol.i.2	31	31	31	31	31	31
myciel7	8	8	8	8	8	8
queen14_14	22	18	18	17	19	18
mulsol.i.1	49	49	49	49	49	49
zeroin.i.3	30	30	30	-	30	-
zeroin.i.2	30	30	30	-	30	-
zeroin.i.1	49	49	49	-	49	-
queen15_15	23	19	18	18	20	-
DSJC250.1	12	10	10	10	11	-
DSJC250.5	40	36	35	34	39	-
DSJC250.9	89	84	82	-	90	-
queen16_16	26	21	19	19	21	-

Pokračování tabulky 7.5						
Název grafu G	CLF	CLFI	CLFIE	CLFIEK3	Gen1	Gen2
fpsol2.i.1	65	65	65	-	65	-
fpsol2.i.3	30	30	30	-	30	-
le450_5a	12	10	9	-	12	-
le450_5b	12	10	9	-	12	-
le450_15a	18	17	16	-	20	-
le450_15b	18	16	16	-	19	-
le450_25a	26	25	25	-	26	-
le450_25b	26	25	25	-	25	-
le450_5d	13	9	7	-	13	-
le450_5c	10	8	7	-	14	-
le450_15c	27	24	23	-	28	-
le450_15d	26	24	23	-	28	-
le450_25c	30	29	28	-	33	-
le450_25d	31	29	28	-	34	-
fpsol2.i.2	30	30	30	-	30	-
DSJR500.1	13	12	12	-	13	-
DSJR500.5	133	128	127	-	140	-
DSJC500.5	67	65	61	-	68	-
DSJC500.1	17	16	15	-	18	-
DSJC500.9	170	155	150	-	168	-
DSJR500.1c	100	90	90	-	99	-
inithx.i.3	31	31	31	-	31	-
inithx.i.2	31	31	31	-	31	-
inithx.i.1	54	54	54	-	54	-
qg.order30	41	30	30	-	32	-
latin_square_10	150	131	122	-	142	-
wap05	50	50	50	-	51	-
wap06	46	43	42	-	51	-
DSJC1000.1	30	27	25	-	29	-
DSJC1000.5	123	115	108	-	121	-
DSJC1000.9	309	285	273	-	309	-
qg.order40	53	40	40	-	43	-
wap07	49	47	45	-	55	-
wap08	49	45	44	-	55	-
qg.order60	83	60	60	-	63	-

7.1.2 Grafy

Poznámka. Do grafů, které se zaměřují na porovnání všech algoritmů, nejsou zahrnuty algoritmy RSIEK3, LFIEK3, SLIEK3, CLFIEK3 a Gen2 z důvodu neúplnosti dat.

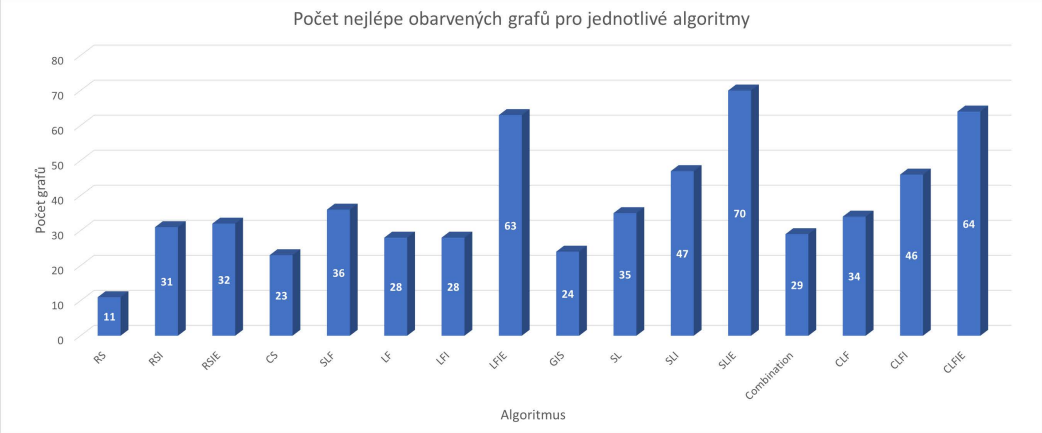
Graf 7.1 zobrazuje pro každý algoritmus počet grafů, které nejlépe obarvil ze všech ostatních algoritmů (včetně algoritmů, které se nenacházejí v grafu, a to RSIEK3, LFIEK3, SLIEK3, CLFIEK3 a Gen2). Z grafu je viditelná významná abnormalita u algoritmů využívající rozšířenou metodu interchange. Dále si všimněme, že hodnoty u LFIE, SLIE a CLFIE jsou podobné. Možné jsou dvě vysvětlení:

- "omezení" způsobuje kombinace heuristiky s preferovaným obarvením vrcholů většího stupně s rozšířenou metodou interchange, nebo
- "omezení" je způsobené pouze rozšířenou metodou interchange (s libovolnou jednoduchou heuristikou pro vybírání pořadí vrcholů pro obarvení).

Díky hodnotám u LF, LFI a LFIE se zdá být pravděpodobnější druhá varianta. Toto je určeno pro další bádání, kdy by se rozšířená metoda interchange použila i u jiných typ heuristik.

Navíc si všimněme, že algoritmům bez heuristiky, tj. algoritmus RSIE, rozšířená metoda interchange moc nepomohla, takže zřejmě lze usoudit, že rozšířená metoda interchange je bez dalších heuristik pro vybírání pořadí vrcholů pro obarvení slabá. I toto je určeno pro další bádání.

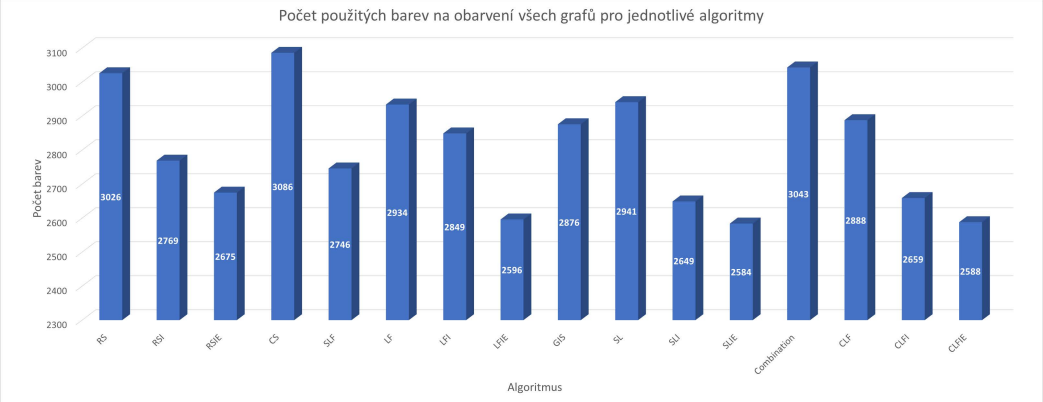
Na základě grafu ještě poznamenejme, že CLF algoritmus patří mezi nejlepší algoritmy, které nepoužívají žádnou variantu metody interchange. A navíc CLFIE algoritmus se dle grafu řadí na druhé místo po SLIE algoritmu.



Graf 7.1: Počet nejlépe obarvených grafů pro jednotlivé algoritmy

Graf 7.2 zobrazuje pro jednotlivé algoritmy, kolik použily celkem barev na obarvení všech grafů. Díky nevyváženosti rozložení hustoty chromatických číslech grafů, není tento graf zase tak vypovídající. Přesto si můžeme zase všimnout abnormalit u algoritmů využívající rozšířenou metodu interchage, které použily přibližně stejný počet barev.

Pro úplnost uveďme, že minimální počet barev na obarvení všech grafů je nejvýše 2333.



Graf 7.2: Počet použitých barev na obarvení všech grafů pro jednotlivé algoritmy

Nyní se zaměříme na porovnání jednotlivých variant interchangů.

Poznámka. Následující data vycházejí pouze z těch grafů, které byly obarveny oběma algoritmy.

Standardní interchange vs rozšířená metoda interchange

Algoritmus LFIE použil o 253 barev **méně** na obarvení všech grafů než algoritmus LFI. Dále algoritmus LFIE nejlépe obarvil o 35 grafů **více** než algoritmus LFI.

Algoritmus SLIE použil o 65 barev **méně** na obarvení všech grafů než algoritmus SLI. Dále algoritmus SLIE nejlépe obarvil o 23 grafů **více** než algoritmus SLI.

Algoritmus CLFIE použil o 71 barev **méně** na obarvení všech grafů než algoritmus CLFI. Dále algoritmus CLFIE nejlépe obarvil o 18 grafů **více** než algoritmus CLFI.

Je očividné, že rozšířená metoda interchange je lepší než standardní metoda interchange.

Rozšířená metoda interchange vs rozšířená metoda interchange s přebarvením K3

Algoritmus LFIE použil o 15 barev **méně** na obarvení všech grafů než algoritmus LFIEK3. Dále algoritmus LFIE nejlépe obarvil o 12 grafů **více** než algoritmus LFIEK3.

Algoritmus SLIE použil o 2 barvy **více** na obarvení všech grafů než algoritmus SLIEK3. Dále algoritmus SLIE nejlépe obarvil o 1 graf **méně** než algoritmus SLIEK3.

Algoritmus CLFIE použil o 1 barvu **více** na obarvení všech grafů než algoritmus CLFIEK3. Dále algoritmus CLFIE nejlépe obarvil o 1 graf **méně** než algoritmus CLFIEK3.

Je vidět, že metody jsou obdobně dobré.

7.2 Měření doby trvání algoritmů

Časové složitosti algoritmů implementovaných na *BE* se vyskytují v tabulce 7.6. Nutno podotknout, že se jedná o **časové složitosti v nejhorších případech** a amortizované časové složitosti se mohou a zpravidla se i liší.

Tabulka 7.6: Časové složitosti jednotlivých algoritmů

Název algoritmu	Časová složitost
RS	$\mathcal{O}(\|V\| + \|E\|)$
RSI	$\mathcal{O}(\|V\|^3)$
RSIE	$\mathcal{O}(\|V\|^4)$
RSIEK3	$\mathcal{O}(\|V\|^5)$
CS	$\mathcal{O}(\|V\| + \|E\|)$
SLF	$\mathcal{O}(\|V\|^3)$
LF	$\mathcal{O}(\|V\|^2)$
LFI	$\mathcal{O}(\|V\|^3)$
LFIE	$\mathcal{O}(\|V\|^4)$
LFIEK3	$\mathcal{O}(\|V\|^5)$
GIS	$\mathcal{O}(\|V\|^3)$
SL	$\mathcal{O}(\|V\|^2 + \|V\| \times \|E\|)$
SLI	$\mathcal{O}(\|V\|^3)$
SLIE	$\mathcal{O}(\|V\|^4)$
SLIEK3	$\mathcal{O}(\|V\|^5)$
Combination	$\mathcal{O}(\|V\|^3)$
CLF	$\mathcal{O}(\|V\|^2)$
CLFI	$\mathcal{O}(\|V\|^3)$
CLFIE	$\mathcal{O}(\|V\|^4)$
CLFEK3	$\mathcal{O}(\|V\|^5)$
Gen1	$\mathcal{O}(\|V\|^2 + \|V\| \times \|E\|)$
Gen2	$\mathcal{O}(\|V\|^3 + \|V\|^2 \times \|E\|)$

Experimenty byly provedeny na grafech s počtem vrcholů *50*, *100*, *150*, *200*, *250* a *300*, kde se pro každý počet vrcholů vygenerovalo celkem 30 grafů po dvou grafech, jeden s menší hustotou (3% - 10%) a druhý s větší hustotou (35% - 40%). Grafy byly generovány náhodně. Pro lepší přesnost naměřených výsledků byly jednotlivé grafy obarveny jedním algoritmem 100x.

Některé grafy nebyly pro větší časovou složitost obarveny některými algoritmy.

Měření probíhalo na počítači: Intel® Core™ i5-8250U CPU @ 1.60GHz, RAM: 8GB, L1: 256KB, L2: 1MB, L3: 6MB, operační systém: Windows 10.

Poznámka. Tento experiment zabral tři dny výpočetního času na jednom počítači.

7.2.1 Tabulky

Tabulka 7.7 zobrazuje průměrné časy na obarvení řídkých a hustých grafů s daným počtem vrcholů. Z tabulky si můžeme všimnout, že pro husté grafy doba trvání algoritmů se standardní metodou interchange a rozšířenou metodou interchange jsou podobné. Dále je vidět, že algoritmy používající rozšířenou metodu interchange s přebarvením K3 mají obrovské průměrné časy.

Tabulka 7.7: Průměrné časy pro jednotlivé algoritmy

|V|	Algoritmus	Průměrný čas v ms (řídké grafy \ husté grafy)	Průměrný počet volání metody interchange (řídké grafy \ husté grafy)
50	CLF	0 \ 1	-
	CLFI	1 \ 4	9 \ 16
	CLFIE	4 \ 10	9 \ 17
	CLFIEK3	6 \ 87	9 \ 17
	RS	0 \ 0	-
	RSI	1 \ 3	10 \ 16
	RSIE	5 \ 9	11 \ 18
	RSIEK3	7 \ 60	12 \ 19
	LF	0 \ 0	-
	LFI	0 \ 3	8 \ 14
	LFIE	4 \ 8	9 \ 15
	LFIEK3	6 \ 74	9 \ 16
	SL	2 \ 3	-
	SLI	2 \ 6	7 \ 13
	SLIE	6 \ 12	7 \ 14
	SLIK3	8 \ 106	7 \ 15
	CS	0 \ 0	-
	SLF	0 \ 1	-
	GIS	2 \ 4	-
	Combination	0 \ 1	-
	Gen1	66 \ 100	-
100	CLF	1 \ 8	-
	CLFI	2 \ 29	13 \ 31
	CLFIE	16 \ 57	15 \ 37
	CLFIEK3	18 \ 757	15 \ 38
	RS	0 \ 0	-
	RSI	2 \ 29	18 \ 35
	RSIE	28 \ 54	19 \ 41
	RSIEK3	29 \ 495	19 \ 42
	LF	0 \ 1	-
	LFI	1 \ 23	12 \ 30
	LFIE	19 \ 54	15 \ 37
	LFIEK3	20 \ 693	15 \ 37
	SL	8 \ 20	-
	SLI	9 \ 42	10 \ 31
	SLIE	23 \ 71	12 \ 36
	SLIK3	24 \ 899	12 \ 38
	CS	0 \ 1	-
	SLF	1 \ 8	-
	GIS	6 \ 27	-
	Combination	1 \ 8	-
	Gen1	270 \ 589	-

Pokračování tabulky 7.6			
|V|	Algoritmus	Průměrný čas v ms (řídké grafy \ husté grafy)	Průměrný počet volání metody interchange (řídké grafy \ husté grafy)
150	CLF	2 \ 23	-
	CLFI	5 \ 97	20 \ 48
	CLFIE	40 \ 168	21 \ 60
	CLFIEK3	41 \ 2727	21 \ 63
	RS	0 \ 1	-
	RSI	4 \ 100	24 \ 53
	RSIE	80 \ 159	30 \ 65
	RSIEK3	81 \ 1852	31 \ 64
	LF	0 \ 1	-
	LFI	3 \ 81	18 \ 48
	LFIE	48 \ 154	22 \ 58
	LFIEK3	48 \ 2603	22 \ 59
	SL	18 \ 57	-
	SLI	20 \ 139	14 \ 49
	SLIE	59 \ 216	17 \ 59
	SLIK3	59 \ 3258	18 \ 60
	CS	1 \ 2	-
	SLF	2 \ 22	-
	GIS	13 \ 87	-
	Combination	2 \ 22	-
	Gen1	627 \ 1686	-
200	CLF	3 \ 50	-
	CLFI	7 \ 254	21 \ 66
	CLFIE	76 \ 383	27 \ 84
	CLFIEK3	78 \ 8349	27 \ 84
	RS	0 \ 2	-
	RSI	7 \ 266	29 \ 75
	RSIE	165 \ 370	38 \ 92
	RSIEK3	165 \ 5693	35 \ 89
	LF	0 \ 2	-
	LFI	4 \ 227	20 \ 66
	LFIE	75 \ 358	26 \ /82
	LFIEK3	76 \ 8808	26 \ 83
	SL	32 \ 125	-
	SLI	35 \ 352	16 \ 71
	SLIE	104 \ 504	22 \ 89
	SLIK3	105 \ 10192	22 \ 91
	CS	1 \ 3	-
	SLF	3 \ 47	-
	GIS	21 \ 215	-
	Combination	3 \ 48	-
	Gen1	1100 \ 3708	-

Pokračování tabulky 7.6			
|V|	Algoritmus	Průměrný čas v ms (řídké grafy \ husté grafy)	Průměrný počet volání metody interchange (řídké grafy \ husté grafy)
250	CLF	4 \ 90	-
	CLFI	9 \ 529	23 \ 85
	CLFIE	127 \ 747	30 \ 112
	RS	0 \ 2	-
	RSI	9 \ 555	34 \ 93
	RSIE	304 \ 709	45 \ 119
	LF	1 \ 3	-
	LFI	6 \ 471	25 \ 86
	LFIE	125 \ 683	31 \ 108
	SL	50 \ 231	-
	SLI	54 \ 719	19 \ 92
	SLIE	155 \ 956	26 \ 116
	CS	1 \ 4	-
	SLF	3 \ 85	-
	GIS	33 \ 437	-
	Combination	3 \ 85	-
	Gen1	1733 \ 6799	-
300	CLF	5 \ 149	-
	CLFI	10 \ 992	27 \ 106
	CLFIE	226 \ 1292	38 \ 136
	RS	0 \ 3	-
	RSI	11 \ 1041	41 \ 114
	RSIE	477 \ 1224	50 \ 144
	LF	1 \ 4	-
	LFI	7 \ 878	27 \ 103
	LFIE	207 \ 1149	35 \ 134
	SL	67 \ 379	-
	SLI	73 \ 1310	21 \ 113
	SLIE	221 \ 1593	28 \ 142
	CS	2 \ 6	-
	SLF	4 \ 138	-
	GIS	44 \ 806	-
	Combination	4 \ 138	-

Poznámka. Průměrná doba trvání $t = 0$ se musí brát s nadhledem. Tyto hodnoty se pohybují mezi 0 ms $< t <$ 1 ms.

Následující tabulka 7.8 zobrazuje průměrnou dobu trvání jednoho zavolání metody interchange pro jednotlivé algoritmy (*RS*, *LF*, *SL* a *CLF*). Hodnoty vznikly jako podíl průměrného času na obarvení grafů a průměrným počtem volání metody interchange. Nejedná se tedy o přesný průměrný čas strávený zavoláním metody interchange. Ale protože u daných algoritmů má metoda interchange zpravidla největší režii, tak nejsou tyto hodnoty bezvýznamné. I z této tabulky si můžeme všimnout, že průměrný čas jednoho zavolání standardní a rozšířené metody interchange jsou podobné pro husté grafy. Dále je vidět, že průměrné časy

na zavolání jedné rozšířené metody interchange s přebarvením K3 jsou obrovské.

Tabulka 7.8: Průměrné časy jednoho zavolání metody interchange

|V|	Varianta interchange	Algoritmus	Průměrný čas v ms jednoho zavolání metody interchange (řídké grafy \ husté grafy)
50	interchange	CLFI	0.111 \ 0.25
	interchangeExtended	CLFIE	0.444 \ 0.588
	interchangeExtendedK3	CLFIEK3	0.667 \ 5.118
	interchange	RSI	0.1 \ 0.188
	interchangeExtended	RSIE	0.455 \ 0.5
	interchangeExtendedK3	RSIEK3	0.583 \ 3.158
	interchange	LFI	0 \ 0.214
	interchangeExtended	LFIE	0.444 \ 0.533
	interchangeExtendedK3	LFIEK3	0.667 \ 4.625
	interchange	SLI	0.286 \ 0.461
	interchangeExtended	SLIE	0.857 \ 0.857
	interchangeExtendedK3	SLIEK3	1.143 \ 7.067
100	interchange	CLFI	0.154 \ 0.935
	interchangeExtended	CLFIE	1.0667 \ 1.541
	interchangeExtendedK3	CLFIEK3	1.2 \ 19.921
	interchange	RSI	0.111 \ 0.829
	interchangeExtended	RSIE	1.474 \ 1.317
	interchangeExtendedK3	RSIEK3	1.526 \ 11.786
	interchange	LFI	0.0833 \ 0.767
	interchangeExtended	LFIE	1.267 \ 1.46
	interchangeExtendedK3	LFIEK3	1.333 \ 18.73
	interchange	SLI	0.9 \ 1.354
	interchangeExtended	SLIE	1.917 \ 1.972
	interchangeExtendedK3	SLIEK3	2 \ 23.658
150	interchange	CLFI	0.25 \ 2.0208
	interchangeExtended	CLFIE	1.905 \ 2.8
	interchangeExtendedK3	CLFIEK3	1.952 \ 43.286
	interchange	RSI	0.167 \ 1.887
	interchangeExtended	RSIE	2.667 \ 2.446
	interchangeExtendedK3	RSIEK3	2.613 \ 28.938
	interchange	LFI	0.167 \ 1.688
	interchangeExtended	LFIE	2.181 \ 2.655
	interchangeExtendedK3	LFIEK3	2.181 \ 44.119
	interchange	SLI	1.429 \ 2.837
	interchangeExtended	SLIE	3.471 \ 3.661
	interchangeExtendedK3	SLIEK3	3.278 \ 54.3

Pokračování tabulky 7.8			
|V|	Varianta interchange	Algoritmus	Průměrný čas v ms jednoho zavolání metody interchange (řídké grafy \ husté grafy)
200	interchange	CLFI	0.333 \ 3.848
	interchangeExtended	CLFIE	2.815 \ 4.56
	interchangeExtendedK3	CLFIEK3	2.889 \ 99.393
	interchange	RSI	0.241 \ 3.547
	interchangeExtended	RSIE	4.342 \ 4.021
	interchangeExtendedK3	RSIEK3	4.714 \ 63.966
	interchange	LFI	0.2 \ 3.439
	interchangeExtended	LFIE	2.885 \ 4.366
	interchangeExtendedK3	LFIEK3	2.923 \ 106.120
	interchange	SLI	2.187 \ 4.958
	interchangeExtended	SLIE	4.727 \ 5.663
	interchangeExtendedK3	SLIEK3	4.772 \ 112
250	interchange	CLFI	0.391 \ 6.223
	interchangeExtended	CLFIE	4.233 \ 6.67
	interchange	RSI	0.265 \ 5.968
	interchangeExtended	RSIE	6.756 \ 5.958
	interchange	LFI	0.24 \ 5.477
	interchangeExtended	LFIE	4.032 \ 6.324
	interchange	SLI	2.842 \ 7.815
	interchangeExtended	SLIE	5.961 \ 8.241
300	interchange	CLFI	0.370 \ 9.358
	interchangeExtended	CLFIE	5.947 \ 9.5
	interchange	RSI	0.268 \ 9.131
	interchangeExtended	RSIE	9.54 \ 8.5
	interchange	LFI	0.259 \ 8.524
	interchangeExtended	LFIE	5.914 \ 8.575
	interchange	SLI	3.476 \ 11.593
	interchangeExtended	SLIE	7.893 \ 11.218

7.3 Obarvování náhodných grafů

Předposlední experiment se týká obarvování náhodných grafů, jehož cílem je zjistit chování jednotlivých algoritmů na velkém počtu náhodných grafů.

Bylo celkem nagenerováno přes 135 000 souvislých grafů a 2 500 000 obarvení. Grafy byly generovány s počtem vrcholů 1 až 300, kde počet vygenerovaných grafů s daným počtem vrcholů je roven $3 \times |V|$.

Grafy byly obarveny pravděpodobnostními algoritmy (tj. RS, RSI, RSIE, RSIEK3, Gen1 a Gen2) 10x.

Některé grafy nebyly pro větší časovou složitost obarveny některými algoritmy. Přesně řečeno, grafy byly obarveny genetickým algoritmem s exponentem 2, pokud měly méně než 100 vrcholů a grafy byly obarveny algoritmy využívající rozšířenou metodu interchange s přebarvením K3, pokud měly méně než 200 vrcholů.

Poznámka. Tento experiment zabral přes měsíc výpočetního času na čtyřech počítačích.

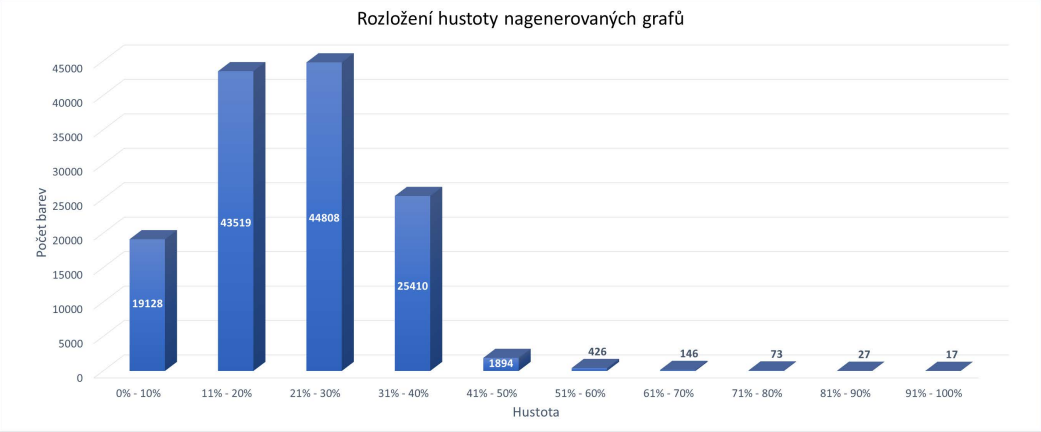
Protože u nagenerovaných grafů neznáme jejich chromatická čísla, tak se nám bude hodit nadefinovat pozměněný *performance guarantee*.

Definice 46. *Pro algoritmus A a algoritmy A_i, kde $1 \leq i \leq k$, definujeme* performance guarantee 2 $\hat{A}(n)$ *následovně:* $\hat{A}(n) = \max\{A(G)/\min\{A_i(G) : pro\ všechna\ i\}: G\ je\ graf\ s\ n\ vrcholy\}$.

Performance guarantee 2 tedy nepoměřuje algoritmus A vzhledem k chromatickému číslu, ale vzhledem k algoritmům A_i. Ve zbytku kapitoly budeme za algoritmy A_i považovat všechny algoritmy, kterými byly dané grafy obarveny. Množina těchto algoritmů vyplyne přímo z textu nebo grafů.

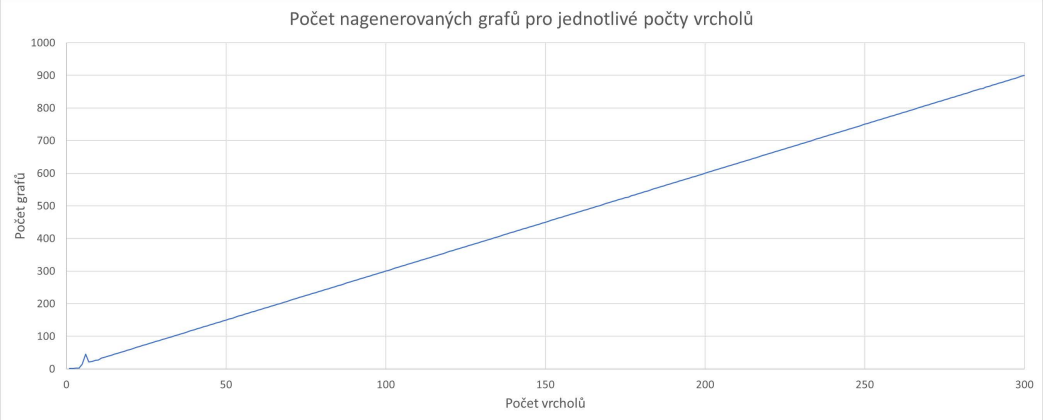
7.3.1 Grafy

Graf 7.3 zobrazuje rozložení hustoty nagenerovaných grafů. Generovaly se grafy s menšími hustotami (0% - 40%) z důvodu rychlejšího obarvení daných grafů. Lze očekávat, že následující výsledky by dopadly obdobně i pro hustší grafy.



Graf 7.3: Rozložení hustoty nagenerovaných grafů

Graf 7.4 zobrazuje počet nagenerovaných grafů pro jednotlivé počty vrcholů.

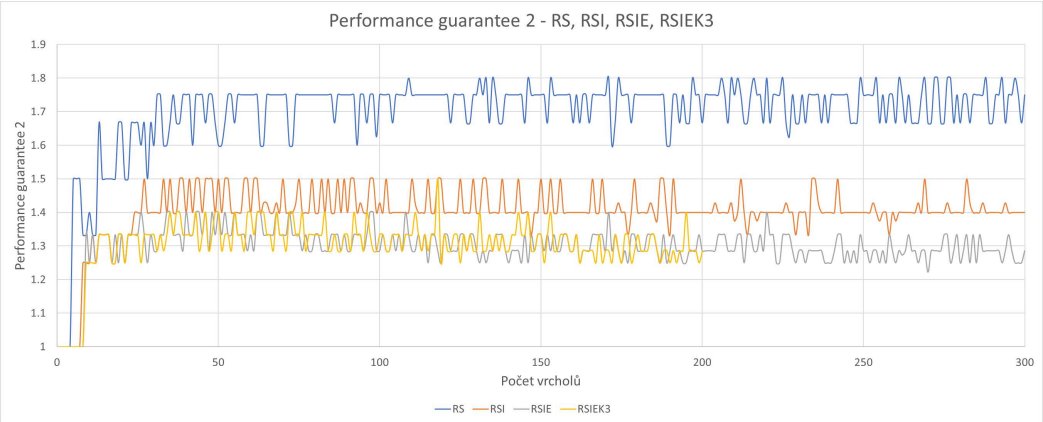


Graf 7.4: Počet nagenerovaných grafů pro jednotlivé počty vrcholů

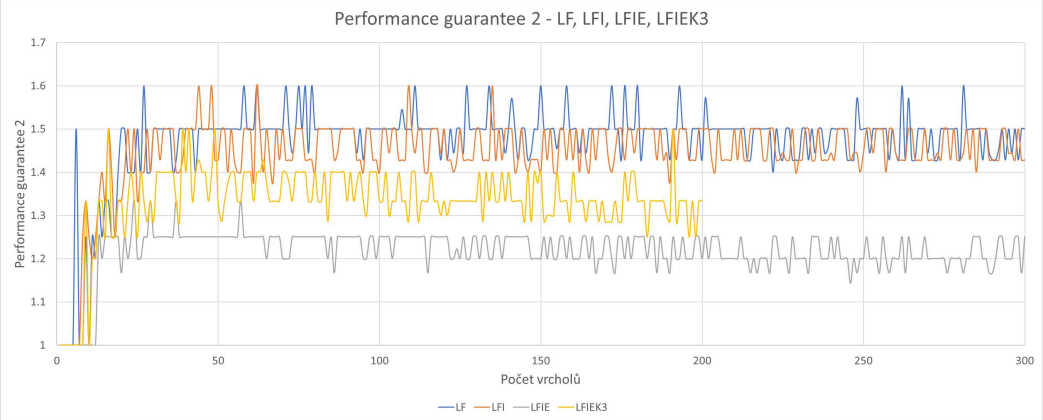
Porovnávání algoritmů na základě performance guarantee 2

V této podkapitole se budou algoritmy porovnávat na základě performance guarantee 2.

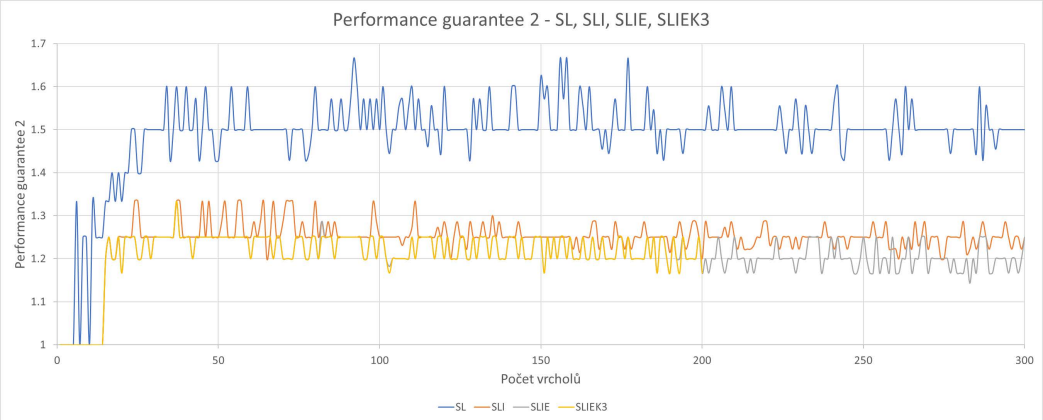
Grafy 7.5 - 7.8 porovnávají různé varianty metody interchange (standardní, rozšířená a rozšířená s přebarvením K3) na algoritmech RS, LF, SL a CLF. Z grafů je patrné, že algoritmy využívající libovolnou variantu metody interchange jsou lepší než ty samé algoritmy nevyužívající žádnou variantu metody interchange (výjimka je pouze u algoritmu LF, kde LF a LFI jsou obdobně dobré). Dále si všimněme, že rozšířená metoda interchange a rozšířená metoda interchange s přebarvením K3 jsou lepší než standardní metoda interchange (u algoritmů CLF a SL je tento rozdíl nepatrný). Překvapující je, že rozšířená metoda interchange a rozšířená metoda interchange s přebarvením K3 jsou obdobně dobré (výjimka zase nastává pouze u algoritmu LF, kde LFIEK3 je dokonce horší než LFIE).



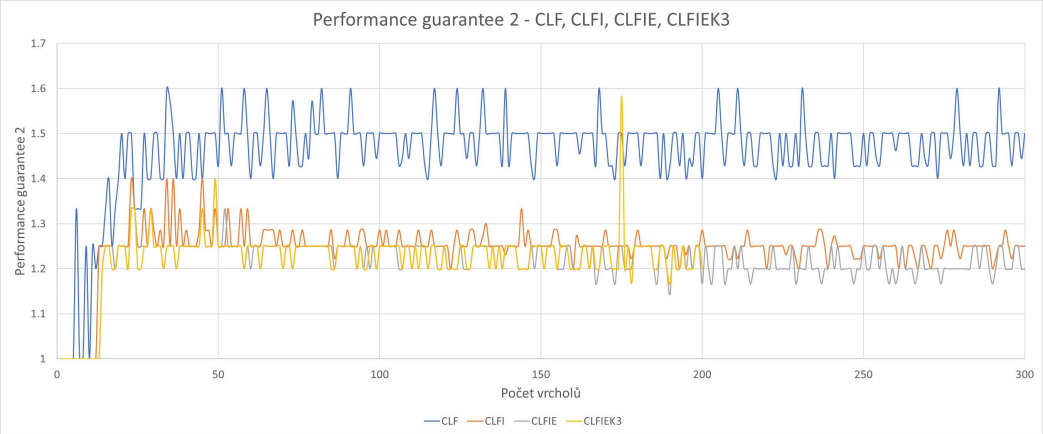
Graf 7.5: Performance guarantee 2 - RS, RSI, RSIE, RSIEK3



Graf 7.6: Performance guarantee 2 - LF, LFI, LFIE, LFIEK3

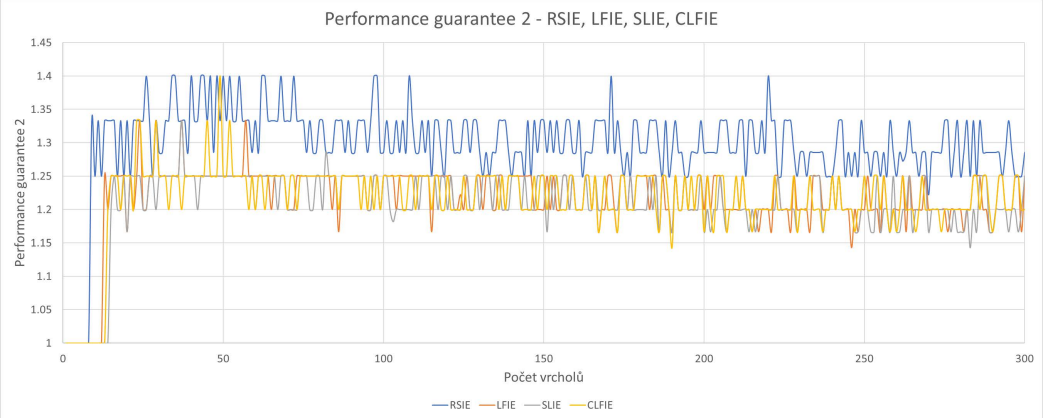


Graf 7.7: Performance guarantee 2 - SL, SLI, SLIE, SLIEK3



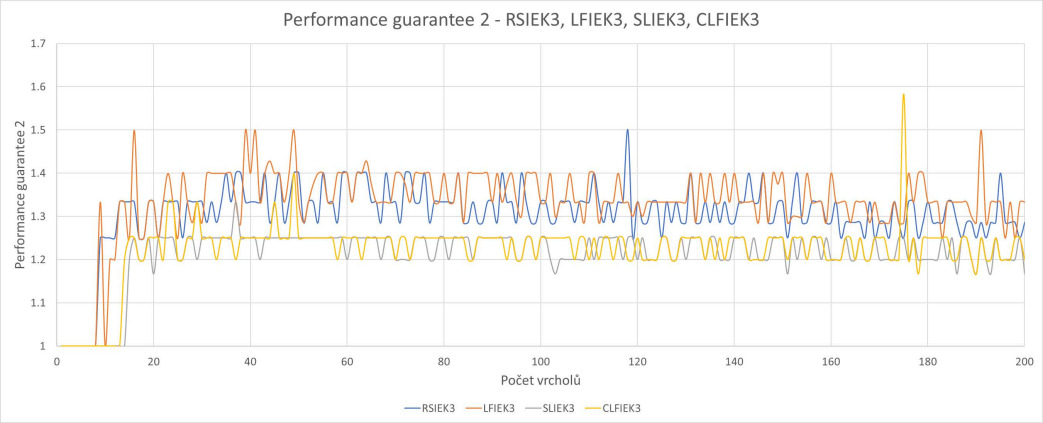
Graf 7.8: Performance guarantee 2 - CLF, CLFI, CLFIE, CLFIEK3

Graf 7.9 porovnává rozšířenou metodu interchange na algoritmech RSIE, LFIE, SLIE a CLFIE. Zde si můžeme všimnout toho, že LFIE, SLIE a CLFIE jsou obdobně dobré a RSIE je očividně horší, což odpovídá naší hypotéze, že rozšířená metoda interchange potřebuje alespoň nějakou jednoduchou heuristiku pro vybírání pořadí vrcholů pro obarvení.



Graf 7.9: Performance guarantee 2 - RSIE, LFIE, SLIE, CLFIE

Graf 7.10 porovnává rozšířenou metodu interchange s přebarvením K3 na algoritmech RSIEK3, LFIEK3, SLIEK3 a CLFIEK3. Zde si můžeme všimnout rozdělení algoritmů do dvou skupin. První (lepší) skupina obsahuje algoritmy SLIEK3 a CLFIEK3, které mají složitější heuristiky a druhá (horší) skupina obsahuje algoritmy RSIEK3 a LFIEK3, které obsahují jednodušší heuristiky. Vystává otázka, zda rozšířená metoda interchange s přebarvením K3 nepotřebuje dokonce nějakou silnější heuristiku na vybírání pořadí vrcholů pro obarvení.



Graf 7.10: Performance guarantee 2 - RSIEK3, LFIEK3, SLIEK3, CLFIEK3

Graf porovnávající všechny algoritmy zde není zobrazen, neboť jeho hustota velmi snižuje jeho přehlednost.

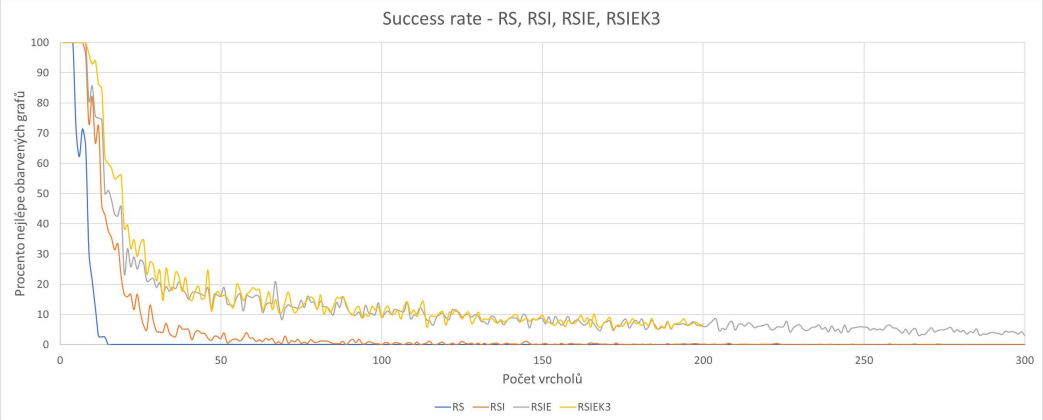
Nejlepších 5 algoritmů vzhledem k naměřeným datům jsou SLIE, SLIEK3, CLFIE, CLFIEK3 a LFIE.

Porovnávání algoritmů na základě míry úspěchu

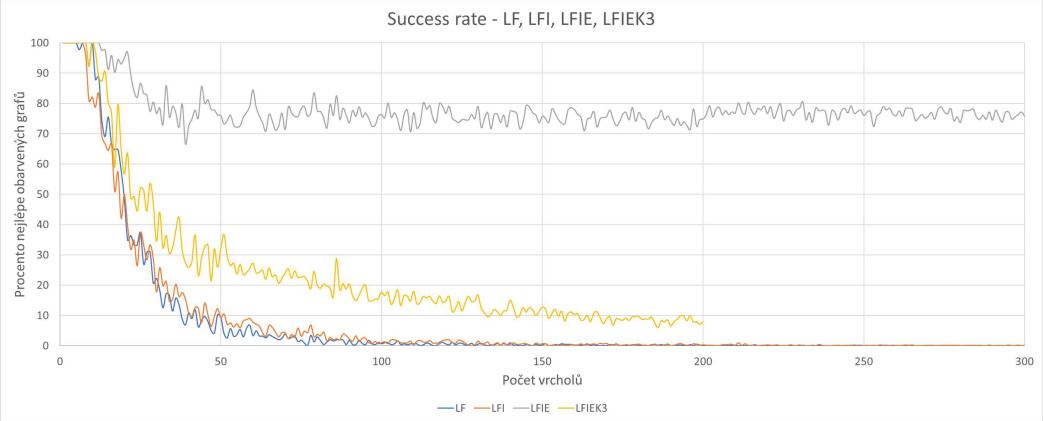
V této podkapitole se budou algoritmy porovnávat na základě míry úspěchu (success rate), tzn. že se algoritmy porovnávají na základě procentuálního počtu grafů, které obarvily nejlépe ze všech ostatních algoritmů.

Grafy 7.11 - 7.14 porovnávají různé varianty metody interchange (standardní, rozšířená a rozšířená s přebarvením K3) na algoritmech RS, LF, SL a CLF. Všim-

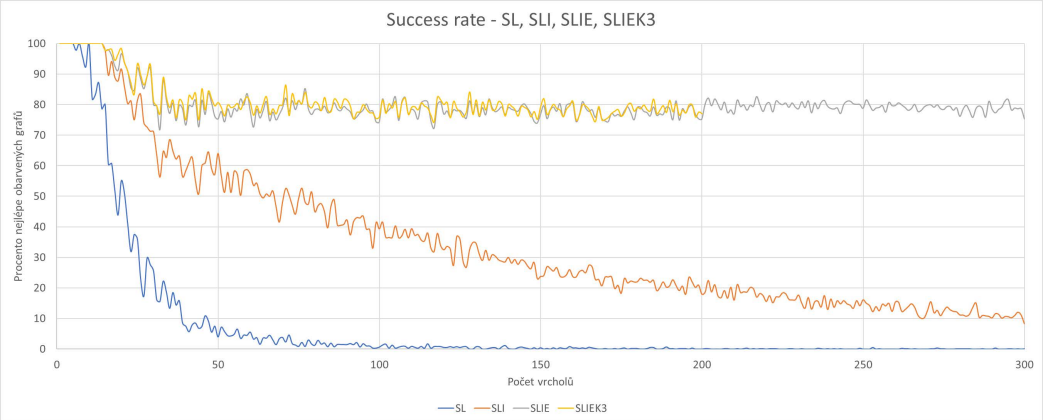
něme si, že z grafů plynou stejné vlastnosti jako u grafů 7.5 - 7.8 v podkapitole 7.3.1.



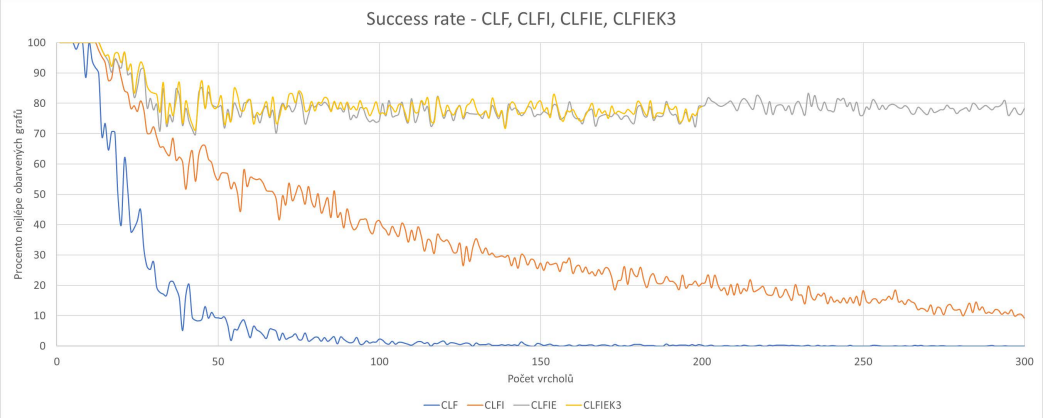
Graf 7.11: Success rate - RS, RSI, RSIE, RSIEK3



Graf 7.12: Success rate - LF, LFI, LFIE, LFIEK3

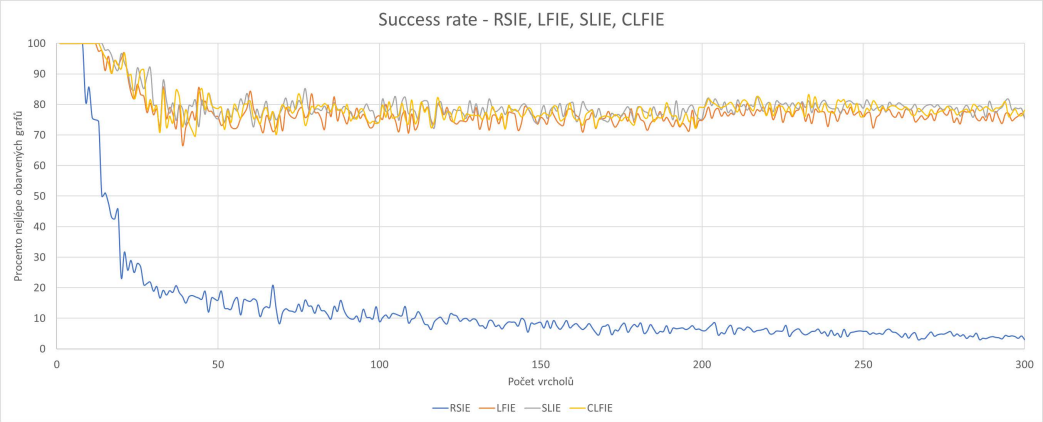


Graf 7.13: Success rate - SL, SLI, SLIE, SLIEK3

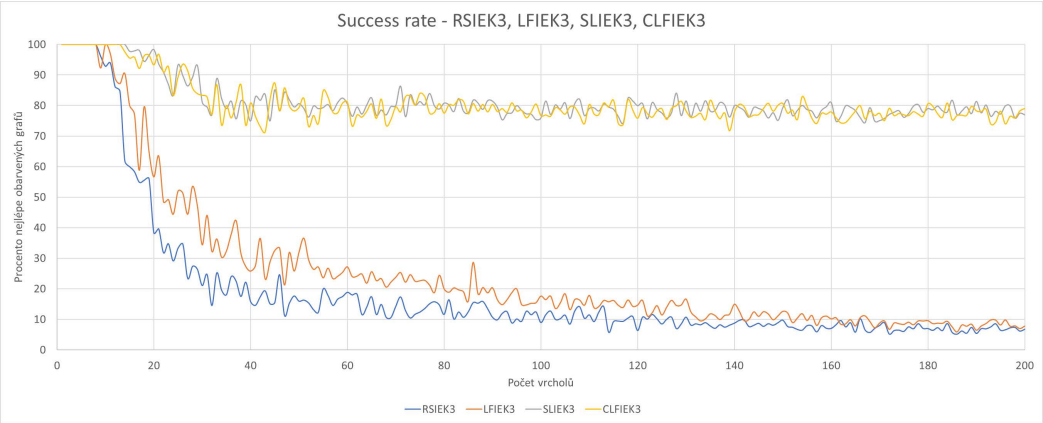


Graf 7.14: Success rate - CLF, CLFI, CLFIE, CLFIEK3

Grafy 7.15 a 7.16 porovnávají rozšířenou metodu interchange a rozšířenou metodu interchange s přebarvením K3 na algoritmech RSIE, LFIE, SLIE, SLFIE a RSIEK3, LFIEK3, SLIEK3, CLFIEK3. I tyto grafy mají stejné vlastnosti jako grafy 7.9 a 7.10 v podkapitole 7.3.1.



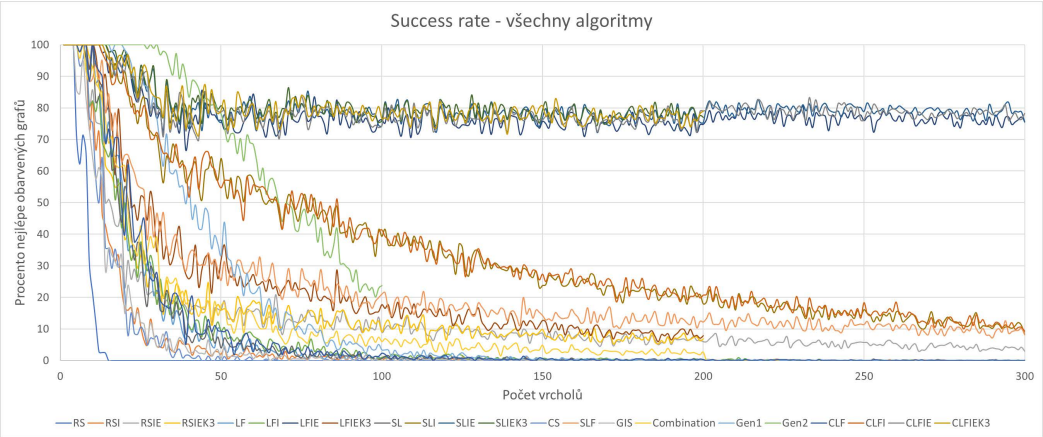
Graf 7.15: Success rate - RSIE, LFIE, SLIE, CLFIE



Graf 7.16: Success rate - RSIEK3, LFIEK3, SLIEK3, CLFIEK3

Graf 7.17 porovnává všechny algoritmy. Z tohoto grafu je patrné, že nejlepších 5 algoritmů vzhledem k naměřeným datům jsou SLIE, SLIEK3, CLFIE,

CLFIEK3 a LFIE.



Graf 7.17: Success rate - všechny algoritmy

7.4 Meta-algoritmus

Nyní změříme experimentálně efektivitu našeho meta-algoritmu.

Bylo celkem nagenerováno přes 3 400 náhodných grafů a 69 000 obarvení. Grafy byly generovány s počtem vrcholů 60 až 300 s rovnoměrným rozložením. Dolní mez je vybrána z toho důvodu, že drtivou většinu grafů s menším stupněm nejlépe obarví genetický algoritmus s exponentem 2 a tento model je pro tuto oblast nejsilnější, takže lze předpokládat, že by téměř vždy volil právě genetický algoritmus s exponentem 2. Horní mez měření je 300, neboť modely byly trénovány na grafy s počtem vrcholů max. 300, a proto lze očekávat, že efektivita pro větší grafy se bude postupně zhoršovat.

Grafy byly obarveny pravděpodobnostními algoritmy (tj. RS, RSI, RSIE, RSIEK3, Gen1 a Gen2) 10x.

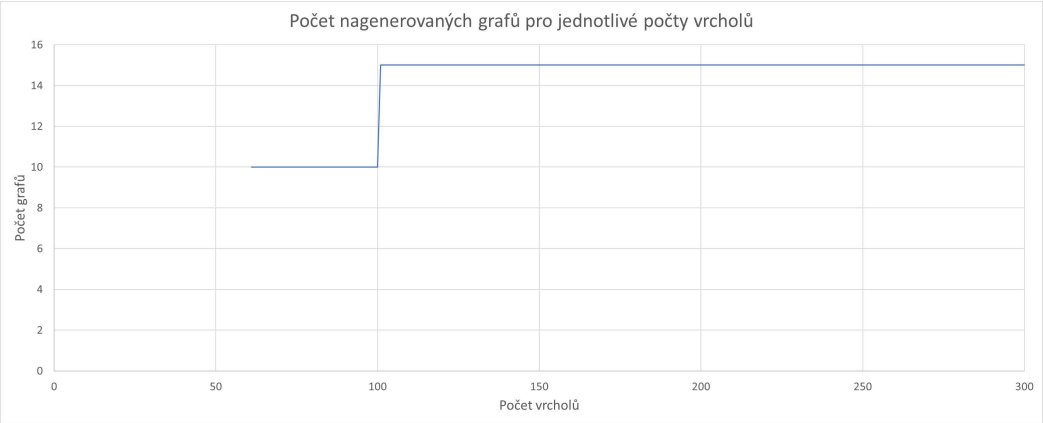
Některé grafy nebyly pro větší časovou složitost obarveny některými algoritmy. Přesně řečeno, grafy byly obarveny genetickým algoritmem s exponentem 2, pokud měly méně než 100 vrcholů a grafy byly obarveny algoritmy využívající rozšířenou metodu interchange s přebarvením K3, pokud měly méně než 200 vrcholů.

Meta-algoritmus obsahuje pouze modely pro následující algoritmy: CLFIE, CLFIEK3, LFIE, SLIE, SLIEK3 a Gen2. Důvod tohoto omezení je zřejmý z předchozích experimentů, kde se ukázalo, že výše zmíněné algoritmy obarví drtivou většinu grafů nejlépe a není tedy potřeba do meta-algoritmu přidávat další "slabé" algoritmy.

Poznámka. Tento experiment zabral jeden den výpočetního času na dvou počítačích.

7.4.1 Grafy

Graf 7.18 zobrazuje počet nagenerovaných grafů pro jednotlivé počty vrcholů.

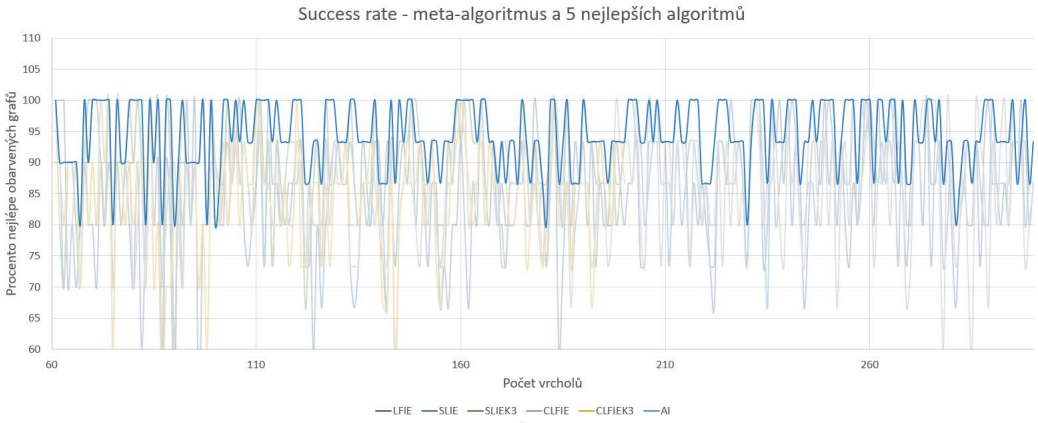


Graf 7.18: Počet nagenerovaných grafů pro jednotlivé počty vrcholů

Meta-algoritmus budeme porovnávat s ostatními algoritmy na základě míry úspěchu (success rate), tzn. že se algoritmy porovnávají na základě procentuálního počtu grafů, které obarvily nejlépe ze všech ostatních algoritmů.

Graf 7.19 porovnává jednotlivé algoritmy s meta-algoritmem. Protože jednotlivých algoritmů je hodně, tak je zobrazeno pouze 5 nejlepších algoritmů (LFIE,

SLIE, SLIEK3, CLFIE a CLFIEK3). Můžeme si všimnout, že meta-algoritmus je až na drobné výjimky mezi prvními. Dále si všimněme, že meta-algoritmus není nějakým výrazným způsobem lepší než zbylé algoritmy, což může být způsobené tím, že výše zmíněné algoritmy obarví nejlépe drtivou většinu grafů, a pokud náhodou nějaký graf není dobře obarven nějakým výše zmíněným algoritmem, tak algoritmus zpravidla použije o jednu barvu navíc.



Graf 7.19: Success rate - meta-algoritmus (AI), LFIE, SLIE, SLIEK3, CLFIE, CLFIEK3

Závěr

Zhodnocení

Naimplementovali jsme dvě aplikace, jednu s uživatelským rozhraním (*GraphColoring*), která umožňuje uživateli pracovat s grafy (vizualizovat, obarvovat, modifikovat apod.) a druhou (*GraphColoringConsole*), která slouží pro práci s velkým počtem grafů a je určená především k vytváření dat k experimentům. Obě aplikace jsou navrženy tak, aby mohly být jednoduše modifikovatelné (např. přidání nové vlastnosti grafu, nového barvícího algoritmu apod.).

Povedlo se nám naimplementovat meta-algoritmus, který pro daný graf nejdříve zjistí, zda neexistuje nějaký algoritmus, který daný graf obarví optimálně. Pokud takový algoritmus neexistuje, tak na základě vlastností grafu je schopen odhadnout nejvhodnější algoritmus pro obarvení daného grafu. Na základě jednoho experimentu jsme ukázali, že je lepší (až na malé výjimky) než ostatní algoritmy, ale ne o tolik. Možných vysvětlení je několik:

- málo grafových vlastností,
- málo vytvořených dat,
- nejpravděpodobnější je však následující: z experimentů vyplynulo, že nejlepšími algoritmy jsou CLFIE, CLFIEK3, LFIE, SLIE a SLIEK3, které využívá i meta-algoritmus. Všechny tyto algoritmy jsou obdobně dobré (tzn. že v průměru pouze dva algoritmy z výše uvedených obarví graf hůře (zpravidla o jednu barvu) než zbytek algoritmů uvedených výše), a proto meta-algoritmus nemůže být až o tolik lepší než jednotlivé algoritmy zmíněné výše.

V kapitolách 1 - 3 jsme čtenáře provedli základní a pokročilejší teorií grafů, která je zaměřená na barvení grafů s řadou příkladů k lepšímu pochopení definic.

Dokázali jsme vymyslet nový algoritmus *CLF*, který se vyrovnává ostatním barvícím algoritmům, které nepoužívají žádnou variantu metody interchange. Rozšířili jsme metodu interchange pojmenovanou rozšířená metoda interchange a rozšířená metoda interchange s přebarvením K3, které zefektivňují algoritmy, které tyto metody používají, což bylo ukázáno na třech experimentech. Ukázali jsme navíc, že algoritmy se standardní metodou interchange a rozšířenou metodou interchange běží pro husté grafy přibližně stejně rychle. Algoritmy využívající rozšířenou metodu interchange s přebarvením K3 jsou obdobně dobré jako ty samé algoritmy využívající rozšířenou metodu interchange. Algoritmy s rozšířenou metodu interchange s přebarvením K3 mají ale obrovské časové složitosti, a tudíž z celkového hlediska se jedná o horší metodu než rozšířená metoda interchange.

Možná rozšíření

Vypozorovali jsme, že algoritmy s libovolnou heuristikou pro vybírání pořadí vrcholů pro obarvení využívající rozšířenou metodu interchange jsou stejně dobré. Bylo by tedy vhodné implementovat rozšířenou metodu interchange i na další algoritmy (např. CS algoritmus) a ověřit, zda toto tvrzení bude platit i pro další

algoritmy. Navíc jsme vypozorovali, že rozšířená metoda interchange s přebarvením K3 k dobré funkčnosti potřebuje algoritmus se složitější heuristikou pro vybírání pořadí vrcholů pro obarvení, i tuto hypotézu by nebylo špatné ověřit naimplementováním více algoritmů využívající rozšířenou metodu interchange s přebarvením K3

V případě většího množství času by mohlo být zajímavé vytváření různých heuristik pro rozšířenou metodu interchange (k čemuž by mohlo pomoci nalezení HC a SHC grafů pro jednotlivé algoritmy využívající rozšířenou metodu interchange), neboť bichromatických podgrafů, ve kterých lze prohodit barvy může být více a nemusí být vždy dobré vybrat daný bichromatický podgraf náhodně. Obdobně pro vybírání artikulace a jejího přebarvení v bichromatickém podgrafu. Dále by bylo dobré se neomezovat jenom na přebarvování sousedů artikulace, ale rekurzit se i na sousedy sousedů apod. a zjistit poměr zlepšení metody vzhledem k době trvání.

Dalším možným rozšířením by mohlo být naimplementování další metody, která by mohla rozšiřovat stávající algoritmy a byla by založená na úplně jiné myšlence než metoda interchange, a následného porovnávaní jednotlivých metod a kombinací těchto metod.

Pro vylepšení meta-algoritmu by neškodilo rozpoznávat více vlastností grafu jako například vrcholová a hranová souvislost, rovinnost grafu apod.

Do aplikace *GraphColoring* by bylo vhodné implementovat ukazatel průběhu obarvování, díky čemuž by uživatel mohl odhadnout dobu skončení obarvení grafu.

Aplikace *GraphColoringConsole* by mohla uživateli umožňovat měřit pro jednotlivé barvící algoritmy dobu trvání obarvování grafů s pevným počtem vrcholů a s měnící se hustotou grafů.

Seznam použité literatury

[1] Jiří Matoušek, Jaroslav Nešetřil. *Kapitoly z diskrétní matematiky*. Karolinum, Praha, 2009. 4. vydání. ISBN: 978-80-246-1740-4.

[2] Adrian Kosowski and Krzysztof Manuszewski. Classical coloring of graphs. 2004.

[3] Artem S. Novozhilov. Erdős–Rényi random graphs. `https://www.ndsu.edu/pubweb/~novozhil/Teaching/767Data/chapter_3.pdf`. [Online; accessed 13-May-2019].

[4] Amites Sarkar. Brooks' Theorem. `http://faculty.wwu.edu/sarkara/brooks.pdf`. [Online; accessed 13-May-2019].

[5] Anne Berry, Jean R. S. Blair, Pinar Heggernes, and Barry W. Peyton. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica*, 39:287–298, 2004.

[6] Martin Mareš, Tomáš Valla. *Průvodce labyrintem algoritmů*. CZ.NIC, Praha, 2017. s. 123-126. 978-80-88168-19-5.

[7] Michael Trick. Graph Coloring Instances. `https://mat.gsia.cmu.edu/COLOR/instances.html`. [Online; accessed 13-May-2019].

[8] Michael Trick. Graph Coloring Instances. `https://mat.gsia.cmu.edu/COLOR03/`. [Online; accessed 13-May-2019].

Seznam obrázků

1.1 Graf . . . 6
1.2 3-regulární graf . . . 8
1.3 Komplementární grafy . . . 8
1.4 Hranový graf . . . 9
1.5 Diskrétní graf $\bar{K}_4$ . . . 10
1.6 Úplný graf K_4 . . . 10
1.7 Cesta P_4 . . . 10
1.8 Kružnice C_4 . . . 10
1.9 Bipartitní graf . . . 11
1.10 Úplný bipartitní graf $K_{3,3}$ . . . 11
1.11 Hvězda S_6 . . . 12
1.12 Kolo W_6 . . . 12
1.13 Kaktus . . . 12
1.14 Dva stejně vypadající grafy . . . 13
1.15 Isomorfní grafy . . . 13
1.16 Přidání vrcholu *11* a odebrání vrcholu *9* v grafu na obr. 1.1 . . . 14
1.17 Kontrakce vrcholu *1* v grafu na obr. 1.1 . . . 14
1.18 Potlačení vrcholu *9* v grafu na obr. 1.1 . . . 15
1.19 Expanze vrcholu *4* v grafu na obr. 1.1 . . . 15
1.20 Přidání hrany $e = \{9, 5\}$ do grafu na obr. 1.1 . . . 15
1.21 Odebrání hrany $e = \{1, 3\}$ z grafu na obr. 1.1 . . . 15
1.22 Kontrakce hrany $e = \{1, 3\}$ v grafu na obr. 1.1 . . . 16
1.23 Dělení hrany $e = \{2, 3\}$ v grafu na obr. 1.1 . . . 16
1.24 Graf s artikulacemi a mosty . . . 18
1.25 Graf . . . 19
1.26 Seznam sousedů pro graf z 1.25 . . . 20
1.27 Strom . . . 21
1.28 Kostra grafu (červené hrany) . . . 22
1.29 Příklad náhodného grafu G(1 000; 0,00095), kde p = $\frac{0,95}{n}$[3] . . . 26
1.30 Příklad náhodného grafu G(1 000; 0,0015), kde p = $\frac{1,5}{n}$[3] . . . 27
1.31 Příklad náhodného grafu G(1 000; 0,007), kde p = $2,\overline{33} \times \frac{log(n)}{n}$[3] 27

2.1 Příklad optimálního vrcholového obarvení grafu s $\chi(G) = 3$ . . . 28
2.2 Příklad optimálního hranového obarvení grafu s $\chi'(G) = 4$ . . . 32

3.1 Nejmenší slighty hard-to-color graf pro RS metodu . . . 36
3.2 Nejmenší slighty hard-to-color graf pro LF metodu . . . 36
3.3 Nejmenší hard-to-color graf pro LF metodu . . . 37
3.4 Nejmenší slighty hard-to-color graf pro SL metodu . . . 38
3.5 Nejmenší hard-to-color graf pro SL metodu . . . 39
3.6 Nejmenší slighty hard-to-color graf pro CS metodu . . . 40
3.7 Nejmenší hard-to-color graf pro CS metodu . . . 40
3.8 Příklad metody interchange při obarvování vrcholu *5* . . . 42
3.9 Nejmenší slighty hard-to-color graf pro RSI metodu . . . 42
3.10 Nejmenší slighty hard-to-color graf pro LFI metodu . . . 43

3.11 Nejmenší hard-to-color graf pro LFI metodu 43
3.12 Nejmenší slighty hard-to-color graf pro SLI metodu 44
3.13 Nejmenší hard-to-color graf pro SLI metodu 44
3.14 Nejmenší slighty hard-to-color graf pro SLF metodu 45
3.15 Nejmenší hard-to-color graf pro SLF metodu 45
3.16 Nejmenší slighty hard-to-color graf pro GIS metodu 46
3.17 Nejmenší hard-to-color graf pro GIS metodu 46
3.18 Příklad křížení dvou jedinců, kde jedinec je reprezentován posloupností vrcholů 47
3.19 Příklad rozšířené metody interchange při obarvování vrcholu *5* . . 52
3.20 Příklad rozšířené metody interchange s rozšířením K3 při obarvování vrcholu *1* 54
3.21 Benchmark pro algoritmy *LF* a *SL* 55
3.22 Benchmark pro algoritmy *LF*, *SL* a *SLF* 55
3.23 Benchmark pro algoritmy *LF*, *SL*, *SLF* a *GIS* 56
3.24 Benchmark pro algoritmy *LF*, *SL*, *SLF*, *LFI* a *SLI* 56
3.25 Weak benchmark pro algoritmy *LF*, *SL*, *SLF*, *LFI*, *SLI* a *GIS* . . 56

5.1 Aplikace *GraphColoring* 58
5.2 Hlavní panel 59
5.3 Panel pro zobrazení grafových vlastností 60
5.4 Panel pro modifikace grafu 61
5.5 Panel pro grafové operace 62
5.6 Tvary vrcholů 62
5.7 Vizualizace grafu 63
5.8 Menu aplikace *GraphColoringConsole* 66
5.9 Generování grafů do souboru - GraphColoringConsole 68
5.10 Generování grafů do databáze - GraphColoringConsole 69
5.11 Vkládání grafů ze souborů do databáze - GraphColoringConsole . 70
5.12 Převod grafů z *.col* na *.graph* - GraphColoringConsole 70
5.13 Měření časových složitostí - GraphColoringConsole 71
5.14 Obarvování grafů - GraphColoringConsole 71

6.1 Komunikace mezi vrstvami 72
6.2 Schéma databáze 73
6.3 Hierarchie namespacu *Graph* 75
6.4 Hierarchie dědění v namespacu *Graph* 76
6.5 Hierarchie namespacu *GraphColoringAlgorithm* 81
6.6 Hierarchie dědění v namespacu *GraphColoringAlgorithm* 82
6.7 Hierarchie namespacu *GenerateGraph* 83
6.8 Hierarchie dědění v namespacu *GenerateGraph* 84
6.9 Hierarchie namespacu *GraphVisualization* 84
6.10 Hierarchie dědění v namespacu *GraphVisualization* 84
6.11 Hierarchie namespacu *ReaderWriter* 85
6.12 Hierarchie dědění v namespacu *ReaderWriter* 85
6.13 Hierarchie namespacu *Tests* 86
6.14 Hierarchie dědění v namespacu *Tests* 86
6.15 Hierarchie namespacu *GUI* 88
6.16 Hierarchie namespacu *ML* 89

7.1 Počet nejlépe obarvených grafů pro jednotlivé algoritmy 103
7.2 Počet použitých barev na obarvení všech grafů pro jednotlivé algoritmy 104
7.3 Rozložení hustoty nagenerovaných grafů 111
7.4 Počet nagenerovaných grafů pro jednotlivé počty vrcholů 112
7.5 Performance guarantee 2 - RS, RSI, RSIE, RSIEK3 112
7.6 Performance guarantee 2 - LF, LFI, LFIE, LFIEK3 113
7.7 Performance guarantee 2 - SL, SLI, SLIE, SLIEK3 113
7.8 Performance guarantee 2 - CLF, CLFI, CLFIE, CLFIEK3 113
7.9 Performance guarantee 2 - RSIE, LFIE, SLIE, CLFIE 114
7.10 Performance guarantee 2 - RSIEK3, LFIEK3, SLIEK3, CLFIEK3 114
7.11 Success rate - RS, RSI, RSIE, RSIEK3 115
7.12 Success rate - LF, LFI, LFIE, LFIEK3 115
7.13 Success rate - SL, SLI, SLIE, SLIEK3 115
7.14 Success rate - CLF, CLFI, CLFIE, CLFIEK3 116
7.15 Success rate - RSIE, LFIE, SLIE, CLFIE 116
7.16 Success rate - RSIEK3, LFIEK3, SLIEK3, CLFIEK3 116
7.17 Success rate - všechny algoritmy 117
7.18 Počet nagenerovaných grafů pro jednotlivé počty vrcholů 118
7.19 Success rate - meta-algoritmus (AI), LFIE, SLIE, SLIEK3, CLFIE, CLFIEK3 119

Seznam tabulek

7.1 Grafy z *DIMACS* kolekce grafů. 93
7.2 Obarvení grafů z *DIMACS* kolekce - 1. část 95
7.3 Obarvení grafů z *DIMACS* kolekce - 2. část 97
7.4 Obarvení grafů z *DIMACS* kolekce - 3. část 99
7.5 Obarvení grafů z *DIMACS* kolekce - 4. část 101
7.6 Časové složitosti jednotlivých algoritmů 105
7.7 Průměrné časy pro jednotlivé algoritmy 106
7.8 Průměrné časy jednoho zavolání metody interchange 109

A.1 Ohodnocení AI modelů obsažených v aplikaci *GraphColoring* . . . 129

Seznam použitých zkratek

$G + e$	přidání hrany e do grafu G
$G \% e$	dělení hrany e v grafu G
$G - e$	odebrání hrany e v grafu G
$G + v$	přidání vrcholu v do grafu G
$G - v$	odebrání vrcholu v v grafu G
$G.e$	kontrakce hrany e v grafu G
$G_1 \cong G_2$	grafy G_1 a G_2 jsou isomorfní
H $\leq$ G	graf H je indukovaným podgrafem grafu G
$\alpha(G)$	velikost maximální nezávislé množiny v grafu G
$\delta(G)$	minimální stupeň grafu G
$\Delta(G)$	maximální stupeň grafu G
$\rho(v)$	nasycenost vrcholu v
$\chi(G)$	chromatické číslo grafu G
$\omega(G)$	velikost maximální kliky v grafu G
A(G)	počet barev použitých při obarvení grafu G algoritmem A
A(n)	performance guarantee
BE	backend
C_n	kružnice s n vrcholy
CLF	connected largest first metoda
CLFI	connected largest first metoda rozšířená standardní metodou interchange
CLFIE	connected largest first metoda rozšířená rozšířenou metodou interchange
CLFIEK3	connected largest first metoda rozšířená rozšířenou metodou interchange s přebarvením K3
CS	connected sequence metoda
DB	databáze
$deg_G(v)$	stupeň vrcholu v v grafu G
E(G)	množina hran grafu G
FE	frontend
G	graf
$\bar{G}$	komplementární graf ke grafu G
g(G)	hustota grafu G
Gen1	genetický algoritmus s exponentem 1
Gen2	genetický algoritmus s exponentem 2
GIS	greedy independent sets
$G(n, p)$	Erdős–Rényiho náhodný graf
HC	hard-to-color graf
K_n	úplný graf s n vrcholy
$\bar{K_n}$	diskrétní graf s n vrcholy
$K_{n,m}$	úplný bipartitní graf
LF	largest first metoda
LFI	largest first metoda rozšířená standardní metodou interchange

LFIE	largest first metoda rozšířená rozšířenou metodou interchange
LFIEK3	largest first metoda rozšířená rozšířenou metodou interchange s přebarvením K3
L(G)	hranový graf ke grafu G
$N_G(v)$	sousedé vrcholu v v grafu G
P_n	cesta délky n
RS	random sequence metoda
RSI	random sequence metoda rozšířená standardní metodou interchange
RSIE	random sequence metoda rozšířená rozšířenou metodou interchange
RSIEK3	random sequence metoda rozšířená rozšířenou metodou interchange s přebarvením K3
S_n	hvězda s n vrcholy
SHC	slightly hard-to-color graf
SL	smallest last metoda
SLF / DSATUR	saturation largest first metoda
SLI	smallest last metoda rozšířená standardní metodou interchange
SLIE	smallest last metoda rozšířená rozšířenou metodou interchange
SLIEK3	smallest last metoda rozšířená rozšířenou metodou interchange s přebarvením K3
V(G)	množina vrcholů grafu G
W_n	kolo s n vrcholy

A. Příloha - Ohodnocení AI modelů

Tabulka A.1 zobrazuje, jaké modely pro jednotlivé algoritmy obsahuje aplikace *GraphColoring* včetně jejich ohodnocení. Modely pro některé algoritmy nebyly vytvořeny pro nedostatek dat.

Tabulka A.1: Ohodnocení AI modelů obsažených v aplikaci *GraphColoring*

Model	Přesnost modelu[a]	Negative precision[b]	Positive precision[c]	Negative recall[d]	Positive recall[e]
~~CLFI~~	~~0.775~~	~~0.772~~	~~0.778~~	~~0.779~~	~~0.77~~
CLFIE	0.64	0.61	0.694	0.781	0.498
CLFIEK3	0.925	0.999	0.85	0.83	0.999
~~RSIE~~	~~0.804~~	~~0.778~~	~~0.833~~	~~0.836~~	~~0.774~~
~~RSIEK3~~	~~0.853~~	~~0.858~~	~~0.849~~	~~0.84~~	~~0.865~~
LFIE	0.637	0.603	0.74	0.78	0.483
~~LFIEK3~~	~~0.849~~	~~0.875~~	~~0.827~~	~~0.813~~	~~0.886~~
~~SLI~~	~~0.792~~	~~0.793~~	~~0.792~~	~~0.787~~	~~0.798~~
SLIE	0.636	0.6	0.76	0.82	0.483
SLIEK3	0.928	0.999	0.834	0.816	0.999
~~SLF~~	~~0.779~~	~~0.775~~	~~0.782~~	~~0.767~~	~~0.79~~
Gen2	0.97	0.983	0.957	0.955	0.984
CLF RS RSI LF LFI SL CS GIS Combination Gen1	modely nebyly vytvořeny pro nedostatek dat				

[a] Přesnost se počítá následovně: $\frac{true\ positive + true\ negative}{true\ positive + true\ negative + false\ positive + false\ negative}$

[b] Negative precision se počítá následovně: $\frac{true\ negative}{true\ negative + false\ negative}$

[c] Positive precision se počítá následovně: $\frac{true\ positive}{true\ positive + false\ positive}$

[d] Negative recall se počítá následovně: $\frac{true\ negative}{true\ negative + false\ positive}$

[e] Positive recall se počítá následovně: $\frac{true\ positive}{true\ positive + false\ negative}$

Poznámka. Modely, které jsou přeškrtnuty, jsou vytvořeny, ale nevyužívá je meta-algoritmus. Je to z toho důvodu, že dané algoritmy jsou příliš slabé a tedy nepotřebné.

B. Příloha - Obsah CD

- *GraphColoring.pdf* - tato práce v elektronické podobě,
- složka *GraphColoringDatabase* obsahuje skripty pro vytvoření schématu databáze,
- složka *GraphColoring\GraphColoring* obsahuje zdrojové kódy k aplikaci *GraphColoring*,
- složka *GraphColoring\GraphColoringConsole* obsahuje zdrojové kódy k aplikaci *GraphColoringConsole*,
- binární soubor aplikace *GraphColoring* se nachází v *GraphColoring\ \GraphColoring\bin\Release\GraphColoring.exe*,
- binární soubor aplikace *GraphColoringConsole* se nachází v *GraphColoring\ \GraphColoringConsole\bin\Release\GraphColoringConsole.exe*,
- složka *Data\TimeComplexity* obsahuje data z měření dob trvání algoritmů,
- složka *Data\DIMACS* obsahuje grafy z DIMACS kolekce grafů ve formátu *.col* a *.graph*,
- složka *Data\HC* obsahuje hard-to-color grafy ve formátu *.graph*,
- složka *Data\Benchmarks* obsahuje benchmarky ve formátu *.graph*,
- složka *Data\ColoredGraphs* obsahuje obarvené grafy (benchmark, hard-to-color grafy a grafy z kolekce DIMACS),
- složka *Data\GeneratedGraphs* obsahuje náhodně vygenerované a obarvené grafy a
- složka *Data\Models* obsahuje AI modely pro jednotlivé algoritmy.