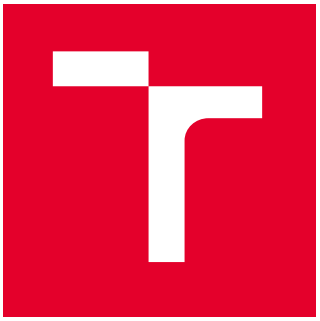


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY

INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

BARVENÍ GRAFU, KLIKA V GRAFU, ALGORITMY A APLIKACE

GRAPH COLOURING, GRAPH CLIQUE, ALGORITHMS AND APPLICATIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Alžbeta Valachová

VEDOUCÍ PRÁCE
SUPERVISOR

prof. RNDr. Ing. Miloš Šeda, Ph.D.

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Studentka:	**Bc. Alžbeta Valachová**
Studijní program:	Aplikovaná informatika a řízení
Studijní obor:	bez specializace
Vedoucí práce:	**prof. RNDr. Ing. Miloš Šeda, Ph.D.**
Akademický rok:	2022/23

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Barvení grafu, klika v grafu, algoritmy a aplikace

Stručná charakteristika problematiky úkolu:

Problém obarvení map nejmenším počtem barev tak, aby sousední země byly barevně odlišitelné, patří k nejznámějším úlohám teorie grafů. Barvení grafů má však obecnější význam, stejně tak pojem klika grafu, a řadu aplikací. Oba problémy jsou NP–úplné a je nutné je řešit heuristickými metodami.

Cílem práce je popsat algoritmy pro řešení těchto problémů, implementovat je a ověřit pro reprezentativní instance a popsat vybrané aplikace.

Cíle diplomové práce:

Popsat algoritmy pro barvení grafů, implementovat je v některém z optimalizačním modelovacím nástroji a ověřit pro reprezentativní instance problému.
Uvést příklady aplikací.

Seznam doporučené literatury:

CORMEN, Thomas H., Charles E. LEISERON , Ronald L. RIVEST and Clifford STEIN. Introduction to Algorithms. Cambridge: MIT Press, 2009.

GAREY, Michael R. and David S. JOHNSON. Computers and Intractability: A Guide to the Theory of NP-Completeness. New York: W. H. Freeman and Company, 1997.

SEDGEWICK, Robert and Kevin WAYNE. Algorithms. New York: Addison-Wesley, 2011.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2022/23

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jiří Hlinka, Ph.D. děkan fakulty

ABSTRAKT

Táto diplomová práca sa zaoberá problematikou farbenia grafu a hľadaním kliky v grafe, s dôrazom na popis algoritmov a ich aplikácie. Farbenie grafu je proces priraďovania farieb jednotlivým vrcholom grafu tak, aby susedné vrcholy mali odlišné farby. Táto problematika je dôležitá pre riešenie rôznych optimalizačných úloh. Práca sa zameriava aj na hľadanie kliky v grafe, čo je dôležitý problém v analýze sociálnych sietí alebo rozpoznávaní obrazov. Cieľom práce je poskytnúť komplexný prehľad o farbení grafu, hľadaní kliky v grafe, predstaviť a analyzovať existujúce algoritmy a ukázať ich aplikácie v praxi.

ABSTRACT

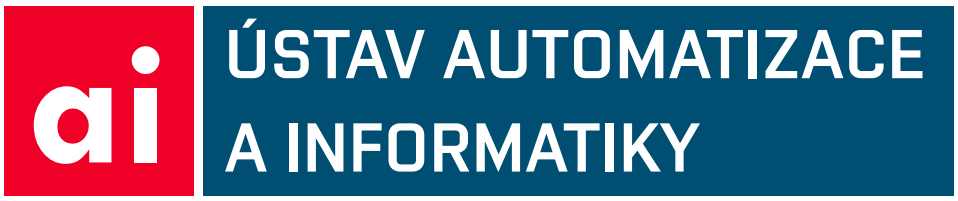
This thesis deals with the problem of graph coloring and finding cliques in a graph, with a focus on the description of algorithms and their applications. Graph coloring is the process of assigning colors to individual vertices of a graph so that adjacent vertices have different colors. This issue is important for solving various optimization problems. The work also focuses on finding cliques in a graph, which is an important problem in social network analysis or image recognition. The aim of this thesis is to provide a complex overview of graph coloring, finding cliques in a graph, introduce and analyze existing algorithms and show their applications in practice.

KĽÚČOVÉ SLOVÁ

Farbenie grafu, maximálna klika, algoritmus, heuristické metódy, NP-úplný problém, aplikácie

KEYWORDS

Graph coloring, maximal clique, algorithm, heuristic methods, NP-complete problem, applications



2023

BIBLIOGRAFICKÁ CITÁCIA

VALACHOVÁ, Alžbeta. *Barvení grafu, klika v grafu, algoritmy a aplikace*. Brno, 2023. Dostupné na: `https://www.vut.cz/studenti/zav-prace/detail/200982`. Diplomová práca. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, Vedúci práce: prof. RNDr. Ing. Miloš Šeda, Ph.D.

Vyhlásenie autora o pôvodnosti diela

Prohlašuji, že tato diplomová práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury.

Jako autor uvedené práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků.

V Brně dne 21. 5. 2023 .....

Alžbeta Valachová

Poďakovanie

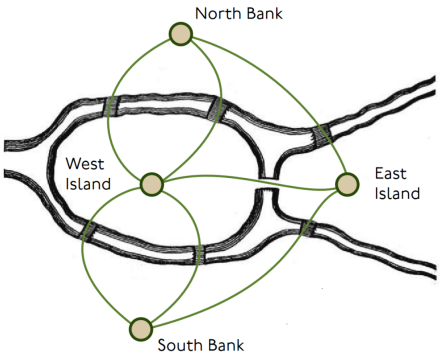
Rada by som poďakovala svojmu vedúcemu diplomovej práce prof. RNDr. Ing. Milošovi Šedovi, Ph.D. za vedenie, odborné konzultácie a pomocnú ruku pri písaní tejto práce. Ďalej by som poďakovala Ing. Petrovi Šoustkovi za odborné rady a mojej rodine a priateľovi za podporu.

Obsah

1 Úvod

Znalosť teórie grafov je neoddeliteľnou súčasťou mnohých oblastí modernej informatiky. Grafy slúžia na to, aby vyjadrovali súvislosti medzi objektami. Sú názorné a dajú sa jednoducho implementovať. Vďaka tomu sa presadili ako matematický model vhodný na popis rôznych problematík, jednoduchých aj komplikovaných.

Teóriu grafov vynašiel Leonhard Euler v roku 1736. Použil ju na problém mostov v prístave Königsberg[1]. Problém pozostával z otázky, či je možná pešia túra po meste tak, aby sa každým mostom prešlo práve raz. Na nasledujúcom obrázku je jednoduchý nákres mesta, na ktorom je znázornené, ako Euler pretransformoval hádanku na matematickú úlohu z oblasti grafov. Nezáležalo na vzdialenostiach ani na veľkostiach jednotlivých oblastí, ale iba na poradí, v akom mali byť navštívené. Oblasti označil ako body a spojil ich oblúkmi, ktoré prechádzali cez mosty. Odtiaľto pochádza slovné spojenie *Eulerovský ťah* [1].



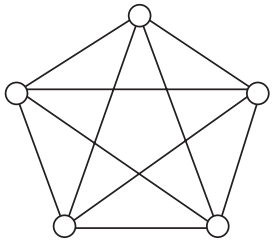
Obr. 1: Problém mostov v Königsbergu [2]

Farbenie grafov je jednou z disciplín teórie grafov. Jeho vznik sa datuje do 19. storočia, keď sa po prvýkrát objavil problém farbenia mapy. Tento problém spočíval v tom, ako priradiť farby jednotlivým regiónom mapy tak, aby susedné regióny nezdieľali rovnakú farbu. Matematici si lámali hlavy na otázke, aký počet farieb je postačujúci. Dnes už je známe, že na vyfarbenie každej mapy stačia štyri farby, ale dokazovanie tohto tvrdenia trvalo viac ako sto rokov. Farbenie grafov sa neskôr zovšeobecnilo a v dnešnej dobe sa aplikuje na rôzne problémy, ktoré sa dajú reprezentovať grafom a s geografickými mapami nemajú nič spoločné. Podobne aj klika v grafe má rad aplikácii, napríklad v lekárstve alebo počítačovom videní. Oba problémy sú NP-úplné, takže ich optimálne riešenie sa zatiaľ nedá získať v polynomiálnom čase. Pristupuje sa k nim heuristickými algoritmami, ktoré sú v polynomiálnom čase schopné nájsť postačujúce riešenie [3].

[1] dnešný Kaliningrad - mesto bolo založené na počesť českého kráľa Přemysla Otakara II.

2 Pojmy a definície

Je nevyhnutné najprv objasniť niektoré teoretické pojmy. Tými sa bude zaoberať táto kapitola. Nasledujúce definície a vety sú čerpané z [4, 5]. Graf všeobecne znázorňuje závislosti medzi prvkami.



Obr. 2: Prostý graf

Definícia 2.0.1. Nech V a E sú neprázdne konečné množiny. **Prostý graf** je potom usporiadaná dvojica $G = (V, E)$.

Poznámka. Prvok $v \in V(G)$ sa nazýva vrchol a prvok $e \in E(G)$ sa nazýva hrana. Hrana je obvykle označovaná dvojicou vrcholov, ktoré spája.

Graf môže byť **neorientovaný** alebo **orientovaný**. Rozdielom medzi nimi je, že v orientovanom grafe je hrana usporiadaná dvojica. Na obrázkoch sa značí šípkou. Hrany (a, b) a (b, a) sa v orientovanom grafe rozlišujú. Naopak v neorientovanom grafe by označovali jednu a tú istú.

Definícia 2.0.2. Množina susedov vrcholu v je množina všetkých vrcholov, ktoré sú s v spojené hranou. Označuje sa $N(v)$.

Definícia 2.0.3. Podgrafom grafu G je ľubovoľný graf H na podmnožine vrcholov $V(H) \subseteq V(G)$

Definícia 2.0.4. Stupňom vrcholu v v grafe G rozumieme počet hrán (resp. počet susedov) vychádzajúcich z v.

Najvyšší stupeň v grafe G sa označuje $\Delta(G)$. Najnižší zase $\delta(G)$.

Definícia 2.0.5. Sledom grafu G je alternujúca postupnosť jeho hrán a vrcholov, ktorá začína a končí vrcholom grafu. Musí platiť, že vrchol a hrana, ktoré nasledujú po sebe, sú incidentné.

Sled môže byť **uzavretý** alebo **otvorený**. Uzavretý sled má počiatočný a koncový vrchol rovnaké. Otvorený ich má rôzne.

Definícia 2.0.6. Ťah je sled, v ktorom sa neopakujú hrany.

Definícia 2.0.7. Neorientovaná cesta je sled, v ktorom sa neopakujú vrcholy.

Definícia 2.0.8. Kružnica je uzavretý ťah, v ktorom sa neopakujú vrcholy (okrem prvého a posledného).

Definícia 2.0.9. Slučka je typ hrany, ktorá je incidentná iba s jedným vrcholom.

Definícia 2.0.10. Graf sa nazýva **súvislý**, ak každé jeho dva vrcholy sú dosiahnuteľné neorientovanou cestou.

Definícia 2.0.11. Množina vrcholov K grafu G sa nazýva **klikou** v grafe G, pokiaľ každé dva rôzne vrcholy z množiny K sú spojené hranou a K je naviac maximálna množina s touto vlastnosťou.

Definícia 2.0.12. Klikovosť grafu je celé, číslo, ktoré udáva veľkosť najväčšej kliky v danom grafe. Klikovosť grafu sa značí $\omega(G)$.

3 Farbenie grafov

Táto kapitola vysvetľuje základné typy farbenia grafu [6, 7].

3.1 Typy farbenia grafov

3.1.1 Vrcholové farbenie

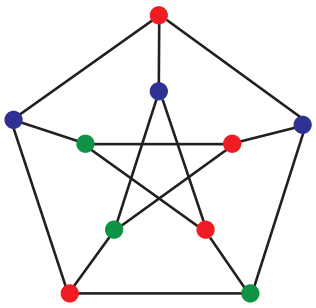
Definícia 3.1.1. Vrcholovým farbením grafu G sa myslí ľubovoľné zobrazenie

$$c : V(G) \to \{1, 2, ...\} \tag{1}$$

Poznámka. Množina $\{1, 2, ...\}$ predstavuje farby, pričom číselné hodnoty sa používajú pre zjednodušenie.

Definícia 3.1.2. K-farbenie grafu G je farbenie s použitím práve k farieb.

$$c : V(G) \to \{1, 2, ..., k\} \tag{2}$$



Obr. 3: Príklad vrcholového farbenia grafu

Definícia 3.1.3. Farbenie grafu sa nazýva **kompletné**, ak má každý vrchol pridelenú farbu.

Definícia 3.1.4. Ak každé dva susedné vrcholy majú rozdielnu farbu, ide o **riadne** farbenie grafu.

Definícia 3.1.5. Farbenie sa nazýva **uskutočniteľné**, pokiaľ je *kompletné* a *riadne* zároveň.

Definícia 3.1.6. Chromatickým číslom $\chi(G)$ grafu G sa nazýva najmenšie prirodzené číslo, pre ktoré existuje riadne farbenie grafu.

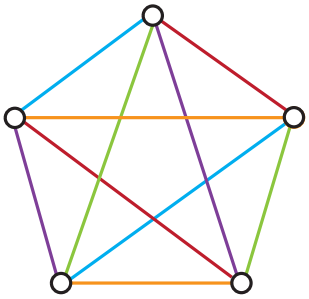
Definícia 3.1.7. Graf sa nazýva k-farebným $\leftrightarrow \chi(G) = k$

Definícia 3.1.8. Nezávislá množina vrcholov je taká množina $A \in V(G)$, kde žiadna hrana $e \in E(G)$ nespojuje vrcholy z množiny A. Počet prvkov **maximálnej nezávislej množiny vrcholov** sa označuje $\alpha(G)$.

3.1.2 Hranové farbenie

Definícia 3.1.9. Hranovým farbením grafu G sa myslí ľubovoľné zobrazenie

$$c : E(G) \to \{1, 2, ...\} \tag{3}$$

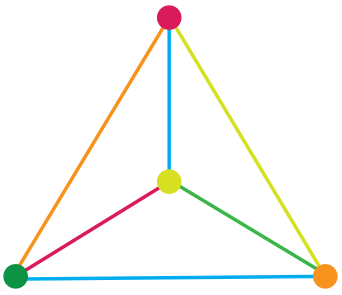


Obr. 4: Príklad hranového farbenia grafu

Definícia 3.1.10. Chromatickým indexom $\chi'(G)$ grafu G sa nazýva najmenšie prirodzené číslo, pre ktoré existuje riadne hranové farbenie grafu G.

3.1.3 Úplne farbenie

Úplné farbenie je spojenie vrcholového a hranového farbenia. Pri riadnom úplnom farbení musí platiť, že žiadne dva susedné vrcholy ani uzly nesmú mať pridelenú rovnakú farbu. Naviac, ani incidentné vrcholy a hrany nesmú mať pridelenú rovnakú farbu.



Obr. 5: Príklad úplného farbenia grafu

3.2 Vlastnosti chromatického čísla

V tejto kapitole budú uvedené chromatické čísla pre rôzne typy grafov [8]. Všeobecne pre každý graf $G = (V, E)$ platí:

$$\chi(G) \leq |V| \tag{4}$$

Ďalšie vlastnosti chromatického čísla:

Veta 3.2.1. *H je podgraf grafu G. Platí:*

$$\chi(H) \leq \chi(G) \tag{5}$$

Veta 3.2.2 (Brooksova veta)**.** *Nech G je súvislý graf rôzny of úplného grafu alebo kružnice. Potom platí:*

$$\chi(G) \leq \Delta(G) \tag{6}$$

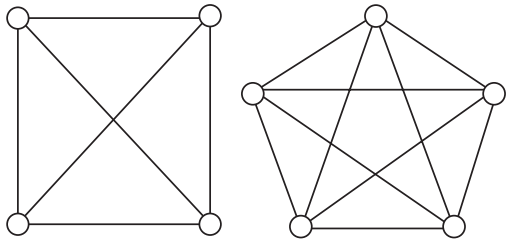
Zároveň, pre každý graf G platí:

$$\chi(G) \leq \Delta(G) + 1 \tag{7}$$

Veta 3.2.3 (Vizingova veta)**.** *Pre každý prostý graf platí:* $\Delta(G) \leq \chi'(G) \leq \Delta(G)+1$

3.2.1 Typy grafov a ich chromatické čísla

Úplný graf je taký neorientovaný graf, ktorého každé dva vrcholy sú prepojené hranou.

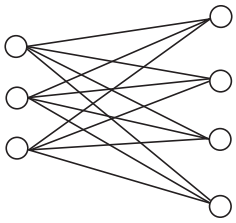


(a) (b)

Obr. 6: Úplné grafy

Pre úplný graf platí: $\chi(G) = |V|$

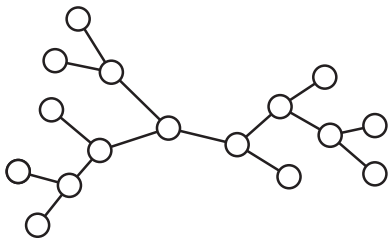
Biparitný graf je graf, ktorého množinu vrcholov V je možné rozdeliť na dve disjunktné množiny tak, že žiadne dva vrcholy z rovnakej množiny nie sú spojené hranou.



Obr. 7: Biparitný graf

Pre biparitný graf platí: $\chi(G) = \chi'(G) = 2$

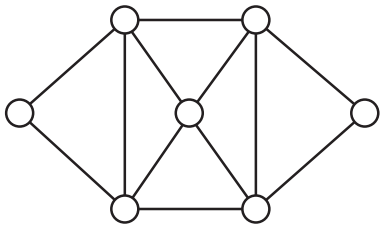
Strom je jednoduchý súvislý graf T bez kružníc.



Obr. 8: Stromový graf

Pre stromový graf platí: $\chi(G) = \chi'(G) = 2$

Planárny graf je graf, ktorý možno nakresliť tak, aby sa jeho hrany nekrížili.



Obr. 9: Planárny graf

Každá geografická mapa sa dá reprezentovať planárnym (rovinným) grafom. O chromatickom čísle plánárneho grafu hovorí slávna veta o štyroch farbách:

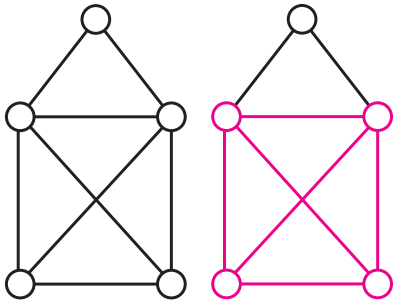
Veta 3.2.4 (Veta o štyroch farbách)**.**

$$\chi(G) \leq 4 \tag{8}$$

4 Klika v grafe

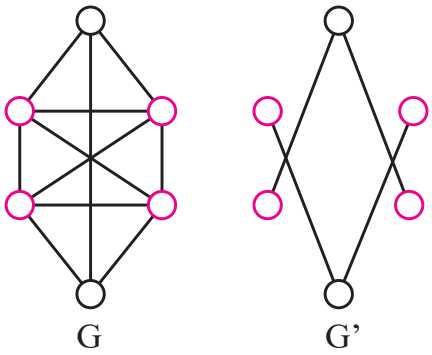
Podľa už spomenutej definície 2.0.11 a doplnenia znalostí o úplnom grafe a podgrafe, je maximálna klika v grafe G najväčší úplný podgraf. Kliky patria medzi základné problémy v teórii grafov a sú aplikovateľné do širokého spektra úloh. Vlastnosti kliky sa často používajú pri riešení problému farbenia grafu.

Na nasledujúcich obrázkoch je vyznačená maximálna klika v grafe.



Obr. 10: Maximálna klika v grafe

Zaujímavé je podotknúť, že vzťah kliky a maximálnej nezávislej množiny z Definície 3.1.8 je komplementný. Ak existuje klika s počtom vrcholov k v grafe G, tak v jeho hranovom doplnku G' existuje maximálna nezávislá množina tiež s počtom vrcholov k [9].



Obr. 11: Vzťah maximálnej kliky a maximálnej nezávislej množiny

Pre kliku a nezávislú množinu platí:

$$\omega(G) = \alpha(G') \tag{9}$$

Hodnota $\omega(G)$ predstavuje počet vrcholov maximálnej kliky v G a hodnota $\alpha(G')$ počet prvkov maximálnej nezávislej množiny G'.

Ďalším dôležitým pozorovaním je vzťah klikovosti $\omega(G)$ a chromatického čísla $\chi(G)$ [9]. Ak graf G obsahuje kliku s počtom vrcholov k, tak len na samotné farbenie kliky je nutné použiť k farieb. Tým pádom chromatické číslo $\chi(G)$ musí byť aspoň také veľké ako k. Pre graf s klikovosťou $\omega(G)$ platí:

$$\chi(G) \geq \omega(G) \tag{10}$$

Pokiaľ sa k tomuto zisteniu pridá Brooksova veta 3.2.2, tak obe hraničné hodnoty chromatického čísla $\chi(G)$ sú:

$$\omega(G) \leq \chi(G) \leq \Delta(G) + 1 \tag{11}$$

Zaujímavým doplnením tejto kapitoly je špeciálna trieda grafov – perfektné grafy. Perfektné grafy boli uvedené začiatkom 60-tych rokov minulého storočia Claude Bergeom. Spájajú rôzne matematické disciplíny zaujímavým spôsobom a majú mnoho uplatnení.

Definícia 4.0.1 (Lovászova veta)**.** Graf G sa nazýva perfektný, ak každý jeho indukovaný podgraf $H \subseteq G$ má chromatické číslo $\chi(H) = \omega(H)$.

Medzi perfektné grafy patria napríklad biparitné grafy, úplné grafy a chordálne grafy. Veľkou výhodou perfektných grafov je to, že úlohy farbenia, maximálnej kliky a nezávislej množiny môžu byť vyriešené v polynomiálnom čase [9].

5 Výpočtová náročnosť

Pojem algoritmus v informatike predstavuje konečnú postupnosť inštrukcií obvykle spúšťanú v počítači. Algoritmus slúži na vyriešenie nejakých problémov alebo úlohy. Na vyriešenie konkrétnej úlohy je potrebná postupnosť konkrétnych definovaných inštrukcií, ktorá je spustená na počítači, aby vykonala špecifickú úlohu. Inštrukcie môžu byť definované rôznymi spôsobmi, v rôznom poradí a stále plniť ten istý účel. Dôležitú úlohu hrá výber dátových štruktúr a programovacieho jazyka, vzhľadom na ktorý sa môže forma podania inštrukcií odlišovať napríklad i v syntaxe. V neposlednom rade samotný počítač – procesor, pamäte, operačný systém - ovplyvňuje výkonnosť algoritmu [10, 11].

Pomocou času a priestoru je možné definovať efektívnosť algoritmov podobne, ako môže byť definovaný akýkoľvek fyzický objekt v prírode. Pre rôzne typy problémov existujú rôzne algoritmy, ktoré sa odlišujú vo svojej optimálnosti, efektívnosti a ďalších vlastnostiach. Efektívnosť sa dá zhodnotiť pomocou časovej a pamäťovej náročnosti. Ich zohľadnenie môže dopomôcť k optimálnemu fungovaniu programu. Pre značenie náročnosti bola definovaná tzv. asymptotická notácia.

5.1 Asymptotická notácia

Náročnosť je vyjadrená prostredníctvom funkcie. Aby bolo možné algoritmy utriediť na základe ich náročností, je vhodné vyjadriť ich funkciami, medzi ktorými sú rozdiely zrejmé a funkcie sa dajú kategorizovať do jednotlivých tried. Funkcie $\log n, n, 2^n$ sú na pohľad veľmi rozdielne, avšak to platí aj pri funkciách $n, 10n, 100n$. Hlavná myšlienka za asymptotickou notáciou je nasledujúca predstava. V momente, kedy by sme sa z veľkej diaľky pozerali na skupinu funkcií, ktoré sa odlišujú len v multiplikatívnej konštante, rozdiely medzi nimi by v podstate zanikli. Rozlíšiť by sa dali iba tie funkcie, medzi ktorými je priepastný rozdiel – teda $\log n, n, 2^n$ [11].

Definícia 5.1.1 (Notácia $\mathcal{O}$)**.** Nech $f : \mathbb{N} \to \mathbb{N}$ a $g : \mathbb{N} \to \mathbb{N}$ sú funkcie. Potom: $f(n) = \mathcal{O}(g(n)) \leftrightarrow \exists$ konštanty $c > 0$ a $n_0 > 0$ také, že $\forall n \geq n_0$ platí: $f(n) \leq c \cdot g(n)$

Poznámka. Znamená to, že funkcia f nerastie rádovo rýchlejšie ako funkcia g.

Definícia 5.1.2 (Notácia Ω)**.** Nech $f : \mathbb{N} \to \mathbb{N}$ a $g : \mathbb{N} \to \mathbb{N}$ sú funkcie. Potom: $f(n) = \Omega(g(n)) \leftrightarrow \exists$ konštanty $c > 0$ a $n_0 > 0$ také, že $\forall n \geq n_0$ platí: $f(n) \geq c \cdot g(n)$

Poznámka. Notácia Ω slovami popisuje, že funkcia f rastie rádovo aspoň tak rýchlo ako funkcia g.

Definícia 5.1.3 (Notácia Θ)**.** Nech $f : \mathbb{N} \to \mathbb{N}$ a $g : \mathbb{N} \to \mathbb{N}$ sú funkcie. Potom: $f(n) = \Theta(g(n)) \leftrightarrow f(n) = \mathbb{O}(g(n))$ a $f(n) = \Omega(g(n))$

Poznámka. Notácia Θ vyjadruje, že f a g rastú rádovo rovnako rýchlo.

5.2 Časová náročnosť

Dôležitou vlastnosťou algoritmu je časová náročnosť jeho výpočtu. V teórii algoritmov sa časová náročnosť nezisťuje pomocou merania výpočtu pre rôzne dáta, ale analýzou samotného algoritmu. Neskúma celkovú dĺžku výpočtu programu, ale vyjadruje informáciu o zmene (zvýšenie alebo zníženie) v čase vykonania, keď sa počet operácií zvýši alebo zníži [10, 11].

Vyjadruje množstvo času, ktoré potrebuje algoritmus na výpočet. Množstvo času predstavuje počet krokov, ktoré algoritmus vykoná a kroky sú v tomto prípade operácie, napríklad aritmetické, ale aj priradenie hodnoty alebo vyhodnotenie podmienky. Časová náročnosť T je funkcia závislá od dĺžky vstupu n:

$$T(n) : \mathbb{N} \to \mathbb{N}$$

kde $T(n)$ je maximálny počet krokov, ktoré algoritmus vykoná pri vstupných dátach veľkosti n. Zvyknú sa uvažovať tri druhy časovej náročnosti: časová náročnosť v najlepšom, najhoršom a priemernom prípade. Časová náročnosť ukazuje priamu koreláciu medzi vstupnými dátami a časom výpočtu [10, 11, 12].

V nasledujúcej tabuľke sa nachádzajú niektoré názvy časových náročností a ich zápisy, kde n je veľkosť vstupu:

názov	**označenie**	$n = 10$	$n = 10^2$	$n = 10^3$	$n = 10^6$
logaritmický čas	$\mathcal{O}(log\ n)$	3.3	6.7	10	20
lineárny čas	$\mathcal{O}(n)$	10	10^2	10^3	10^6
kvadratický čas	$\mathcal{O}(n^2)$	10^2	10^4	10^6	10^{12}
exponenciálny čas	$\mathcal{O}(2^n)$	1024	$13 \cdot 10^{30}$	$11 \cdot 10^{302}$	∞
faktoriálny čas	$\mathcal{O}(n!)$	$36 \cdot 10^5$	$93 \cdot 10^{157}$	$40 \cdot 10^{2567}$	∞

Tab. 1: Tabuľka časových náročností [11]

Hodnoty času, ktoré algoritmy s danými náročnosťami pri zvyšujúcom sa n, potrebujú na výpočty, sú v tabuľke určené počtom operácií.

Najviac žiaduca je konštatná časová náročnosť $\mathcal{O}(1)$, pretože umožňuje výpočty s konštantným časom bez ohľadu na veľkosť vstupu. Ak má algoritmus časovú náročnosť $\mathcal{O}(1)$, jeho časová náročnosť by mala byť rovnaká pre akúkoľvek veľkosť vstupu. V praxi sa však dosahuje iba pre jednoduché operácie, ako je prístup k pamäti alebo aritmetické operácie s konštantami.

Náročnosti v tabuľke sú zoradené od najlepšej po najhoršiu. Konštantná časová náročnosť nezávisí od veľkosti vstupu, preto je najlepšia. Ako náhle sa v programe vyskytne cyklus, časová náročnosť sa zvýši. Príkladom logaritmickej časovej náročnosti sú napríklad funkcie binárneho hľadania. Príkladom exponenciálnej časovej náročnosti je rekurzívna Fibonacciho postupnosť [13].

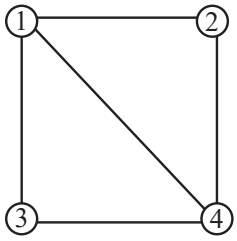
5.3 Pamäťová náročnosť

Pamäťová náročnosť je ďalší dôležitý aspekt na zváženie pri rozhodovaní o efektívnosti algoritmu [10, 11]. Program, ktorý vykonáva operácie, všeobecne potrebuje pamäť na to, aby mohol ukladať dočasné dáta (premenné a konštatné hodnoty), inštrukcie programu alebo koncový výsledok. Množstvo pamäte, ktoré je vyžadované algoritmom na výpočet konkrétnemu problému, sa nazýva pamäťová náročnosť algoritmu. Je vhodné zmieniť sa o tom, že pamäťová náročnosť závisí od programovacieho jazyka, kompilátoru, aj parametrov počítača, na ktorom program algoritmu beží. Pri prebiehaní výpočtu sa používa pamäť hlavne na tieto účely:

- pamäť inštrukcií – pamäť, do ktorej sa ukladá skomplilovaná verzia inštrukcia
- zásobník prostredia – pri volaní funkcie vo vnútri funkcie
- pamäť pre premenné a konštatny

6 Reprezentácia grafov

Všeobecne musí reprezentácia grafu $G = (V, E)$ popisovať vrcholy aj hrany medzi nimi. Preto vznikli jednoducho implementovateľné reprezentácie grafu, ktoré sa používajú v počítačoch [7, 14]. Reprezentácie budú znázornené na nasledujúcom grafe:



Obr. 12: Neorientovaný graf

Zoznam hrán

Jednoduchý typ reprezentácie grafu je zoznam hrán. Je intuitívny a prehľadný. Hrana je jeden prvok zoznamu a je zložená z dvoch vrcholov – počiatočného a koncového. V prípade neorientovaného grafu nezáleží na poradí vrcholov [5].

Matica susedstva

Matica susedstva grafu $G = (V, E)$ je štvorcová $|V| \times |V|$ matica. Pre jej prvky platí:

$$a_{i,j} = \begin{cases} 1 & \text{vrcholy } i \text{ a } j \text{ sú susedné} \\ 0 & \text{vrcholy } i \text{ a } j \text{ nie sú susedné} \end{cases}$$

Výhodou matice susedstva je jednoduchá implementácia. Odstránenie hrany zaberie $\mathcal{O}(1)$ času. Naopak nevýhodou je pamäťová náročnosť $\mathcal{O}(|V|^2)$. V prípade neorientovaného grafu bude matica susedstva symetrická. Matica susedstva grafu z Obr.(12) bude:

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Obr. 13: Matica susedstva grafu

Matica incidencie

Matica incidencie pre graf $G = (V, E)$ je obdĺžniková matica s dimenziou $|V| \times |E|$. Pre jej prvky platí:

$$a_{i,j} = \begin{cases} 1 & \text{vrchol } i \text{ je spojený s hranou } j \\ 0 & \text{vrchol } i \text{ nie je spojený s hranou } j \end{cases} \tag{12}$$

Pri orientovaných grafoch matica incidencie obsahuje prvky 0, 1 a -1. Rozšírenie upresňuje, či je vrchol počiatočným vrcholom hrany alebo koncovým. Matica incidencie pre graf z obrázku Obr.(12) bude:

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Obr. 14: Matica incidencie grafu

Výhodou matice incidencie je, že šetrí miesto - $\mathcal{O}(|V| + |E|)$, v najhoršom prípade $\mathcal{O}(|V|^2)$ [7, 14].

Zoznam následníkov

Ďalším typom reprezentácie grafu $G = (V, E)$ je zoznam. Jeho dĺžka je definovaná veľkosťou množiny vrcholov $|V|$. Zoznam teda obsahuje vrcholy a zároveň každý vrchol obsahuje ukazovateľ na množiny susedných vrcholov. Zoznam následníkov má pamäťovú náročnosť $\mathcal{O}(|V| + |E|)$.

7 Triedy zložitosti

V oblasti počítačovej náročnosti sa problémy uvádzajú ako rozhodovacie problémy, ktoré vyžadujú binárnu odpoveď „Áno/Nie“. Vďaka tomu je možné ich študovať z pohľadu formalizovaných jazykov, čo umožňuje vývoj matematických metód na analýzu a riešenie problémov [9, 10]. Okrem toho, rozhodovacie problémy sú ľahšie transformovateľné na iné typy problémov, ktoré sa v praxi častejšie vyskytujú. Napríklad, farbenie grafu G s použitím k farieb by sa mohlo transformovať na rozhodovací problém použitím otázky „Existuje uskutočniteľné k-farbenie grafu G?“

Algoritmus, ktorý rieši konkrétny rozhodovací problém sa nazýva „dobrý“, pokiaľ sa čas výpočtu dá vyjadriť polynómom. Pri týchto problémoch nedochádza ku kombinátorskej explózii - ich trieda sa nazýva P, ako polynomiálne-časové problémy. Patria sem jednoduché úlohy ako sčítanie dvoch čísel alebo triedenie zoznamu.

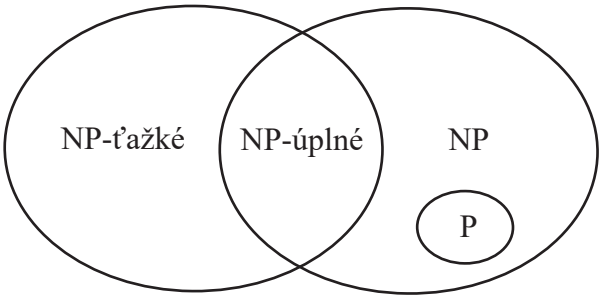
Nie všetky problémy sú tak jednoducho riešiteľné, a preto existuje trieda NP ako nedeterministické polynomiálne-časové problémy. NP problémy sú problémy, ktoré odpovedajú „Áno“ pre kandidátske riešenie a zároveň existuje algoritmus, ktorý v polynomiálnom čase dokáže potvrdiť túto odpoveď pre to dané riešenie. To však neznamená, že existuje algoritmus, ktorý by NP problémy dokázal riešiť v polynomiálnom čase.

Prirodzene, P $\subset$ NP, pretože pre P problém sa riešenie dokáže overiť algoritmom v polynomiálnom čase. Zatiaľ nie je známe, či platí P = NP, ale predpokladá sa opak, a to P $\neq$ NP.

Tým vzniká trieda NP-úplných problémov. Keďže NP-úplné $\subset$ NP, riešenie NP-úplných problémov by mohlo byť overené v polynomiálnom čase, ale zatiaľ neexistuje žiadny algoritmus, ktorý by efektívne dokázal riešenie nájsť. Stavové priestory NP-úplných problémov rastú exponenciálne s veľkosťou problému, takže nie je známy algoritmus, ktorý by získal riešenie v polynomiálnom čase [9]. Do tejto triedy patrí výpočet chromatického čísla grafu, maximálnej kliky v grafe alebo maximálnej nezávislej množiny v grafe, problém obchodného cestujúceho, prípadne problémy z iných matematických oblastí, ako napríklad knapsack problém.

Nakoniec existujú problémy, ktoré sú aspoň také ťažké ako NP-úplné, ale nemusia nutne patriť do NP-triedy. Nazývajú sa NP-ťažké a patrí sem problém súčtu podmnožín.

Na nasledujúcom obrázku sú znázornené množiny tried náročností za predpokladu, že P $\neq$ NP.



Obr. 15: Triedy náročnosti

8 Algoritmy vrcholového farbenia grafov

Najzákladnejšia otázka ohľadom problému farbenia grafov je, aký algoritmus je schopný ho vyriešiť. Slovo vyriešiť je použité v zmysle vyriešiť pre každý graf a vždy vrátiť optimálne riešenie - farbenie grafu s minimálnym počtom farieb [9]. Jedna z metód riešenia farbenia grafov je vyskúšať každé možné priradenie farieb a potom vrátiť to najlepšie z nich. Týmto spôsobom by farbenie bolo uskutočniteľné a zároveň by zistilo aj chromatické číslo vstupného grafu. Najprv je potrebné zistiť hranice chromatického čísla. Pre každý graf $G = (V, E)$ platí:

$$1 \leq \chi(G) \leq |V|$$

Prvým odhadom, ktorý by určite zaručil uskutočniteľné riešenie, by bolo postaviť $\chi(G)$ rovno jeho hornej hranici. Problémom tohto prístupu je to, koľko kombinácii by generoval. Pre graf s n vrcholmi s n možnosťami farieb by bolo možností kombinácii n^n. Toto číslo rastie prudko s rastúcim n. Tým pádom by táto úloha bola veľmi problematická na vyriešenie. O niečo lepší prístup by bolo rozdeliť množinu vrcholov V do množín k podľa farieb. Množina V by bola tvorená disjunktnými podmnožinami $V = \{V_1, ...V_k\}$. Tieto podmnožiny sa nazývajú nezávislými množinami. Problém by sa previedol na minimalizáciu hodnoty k. Bohužiaľ, ani tento prístup nezjednodušuje problematiku. V podstate žiadna metóda, ktorá by určovala farbenie na základe prehľadávania stavového priestoru, nie je vhodná, pretože jej uskutočnenie by zaberalo príliš veľa času. Avšak, existujú algoritmy farbenia grafu, ktoré obvykle nájdu uspokojivé riešenie v polynomiálnom čase. Využívajú heuristiky a prípadne aproximácie. Ďalšou dôležitou informáciou je, že všeobecne je farbenie grafov NP-úplný problém, ale nie pre všetky grafové inštancie. V kapitole 3.2 sú spomenuté chromatické čísla pre niektoré typy grafov. Dokonca pre konkrétne typy grafov heuristické algoritmy prinášajú presné riešenie [9].

8.1 Greedy algoritmus

Greedy alebo sekvenčný algoritmus je asi najzákladnejší algoritmus, ktorý dokáže určiť uskutočniteľné farbenie grafu. Je principiálne jednoduchý a vykazuje dobré výsledky. Je založený na intuitívnom farbení vrcholov grafu [9, 15]. Funguje tak, že prechádza vrcholy jeden za druhým a použije na ne prvú dostupnú (zatiaľ nepoužitú) farbu. Nech G je graf a postupnosť vrcholov $v_1, ..., v_n$. Farby budú označené číslami.

Postup

Algoritmus 1 Greedy algoritmus

$c(v_1) := 0$
for $v := v_2$ to v_n **do**
Priraď vrcholu v najnižšiu dostupnú farbu tak, aby nebola v konflikte medzi v a jeho susednými vrcholmi.
end for

Tento algoritmus pracuje tak, že vykoná prvé možné rozhodnutie bez ohľadu na možné dôsledky. Práve na tomto niekedy zlyháva – nemusí nájsť chromatické číslo. Avšak vždy nájde uskutočniteľné farbenie grafu.

8.1.1 Usporiadanie vrcholov grafu

Výsledné farbenie grafu závisí na tom, ako sú vrcholy zoradené v postupnosti [15]. Existuje niekoľko základných prístupov k ich zoradeniu:

Heuristika First Fit

First Fit je najjednoduchší prístup usporiadania vrcholov. Vrcholy sa spracovávajú v poradí, v akom boli načítané. V tomto prípade sa neuvažuje o sekundárnom usporiadaní vrcholov pred prideľovaním farieb.

Náhodné usporiadanie vrcholov grafu

Náhodné usporiadanie je ďalší prístup k spracovaniu vrcholov. Ide o jednoduchú a rýchlu stratégiu s časovou náročnosťou $\mathcal{O}(|V|)$.

Welsh-Powellova heuristika

Welsh-Powellov heuristický princíp funguje tak, že usporiada vrcholy v závislosti od ich stupňa. Prvý vrchol, ktorému sa pridelí farba, bude ten s najväčším stupňom. Táto heuristika je intuitívne lepšia ako náhodná, pretože vrcholy s veľkým množstvom susedov budú vyriešené ako prvé. Časová náročnosť Welsh-Powellovej heuristiky je $\mathcal{O}(|V|^2)$.

Heuristika DSatur

DSatur je iteratívna heuristika [16]. Pracuje tak, že vrchol, ktorému sa prideľuje farba, sa vyberie v každom kroku heuristicky. Výber závisí od hodnoty saturácie vrcholu v čiastočnom farbení grafu. Saturácia vrcholu je definovaná ako počet susedných vrcholov, ktoré majú pridelenú rôznu farbu. Jedna z vlastností tejto heuristiky je, že pokiaľ sa graf skladá z niekoľkých častí, tak vrcholy v rámci jednej časti budú mať pridelené farby pred zvyšnými časťami. Časová zložitosť je opäť $\mathcal{O}(|V|^2)$, ale

DSatur dosahuje lepšie výsledky ako Welsh-Powell. DSatur všeobecne poskytuje suboptimálne riešenie farbenia grafov, ale pre niektoré typy grafov je dokonca exaktný (biparitné, kružnice, kolesové grafy) [9].

8.2 Rekurzívny algoritmus nezávislých množín

V predošlej kapitole bolo spomenuté, že problém farbenia grafov sa dá previesť na problém nezávislých množín. Množina vrcholov V sa rozdelí na disjunktné nezávislé množiny V_i. Keďže vrcholy v jednotlivých nezávislých množinách nie sú susedné, môžu mať v rámci množiny pridelenú rovnakú farbu. Algoritmus Recursive Largest First bol vytvorený Leightonom v 1979 [17]. Algoritmus funguje tak, že pomocou heuristiky nájde maximálnu nezávislú množinu a potom ju odstráni a proces sa opakuje, až kým nie je graf prázdna množina [18].

Pre jednoduchosť bude *maximálna nezávislá množina vrcholov* ďalej označovaná skratkou *MIS*.

8.2.1 Rekurzívny *MIS*

Rekurzívny Algoritmus hľadania *MIS* iteruje cez všetky vrcholy grafu. Každý vrchol má dve možnosti: je alebo nie je obsiahnutý v *MIS*. Na začiatku je postupnosť $\{v_1, ..., v_n\}$ vrcholov. Vyberie sa vrchol v_i a zmaže sa z grafu. Tým sa vylúči z *MIS*. Rekurzívne sa pre zvyšok grafu nájde *MIS*. Druhá možnosť je, že vrchol sa v *MIS* nachádza. V tom prípade sa z grafu zmažú jeho susedné vrcholy. Z týchto dvoch možností sa vyberie tá, ktorá obsahuje viac vrcholov. Proces sa zopakuje pre všetky vrcholy [19].

Nech G je graf, v_i je aktuálny vrchol, $N(v_i)$ je množina susedných vrcholov v_i, $G(v_i)$ je graf s jediným vrcholom v_i. Potom rekurzívny algoritmus hľadania *MIS* má nasledujúci pseudokód:

Algoritmus 2 Rekurzívny algoritmus hľadania *MIS*

```
function RecursiveMIS(G)
    if size(G) = ∅ then
        return 0
    end if
    sol1 = RecursiveMIS(G − v_i):
    sol2 = G(v_i) + RecursiveMIS(G − v_i − N(v_n))
    return max{sol1, sol2}
end function
```

Problém hľadania *MIS* vrcholov je NP-úplný problém. Rekurzívny algoritmus má časovú náročnosť $\mathcal{O}(2^n)$.

8.2.2 Algoritmus farbenia grafu

Algoritmus farbenia pomocou *MIS* v každej iterácii nájde *MIS* a zmaže ho z grafu G. Výpočet sa opakuje, až kým nie je graf G prázdna množina.

Algoritmus 3 Farbenie grafu pomocou RecursiveMIS

```
while G ≠ ∅ do
    MIS = RecursiveMIS(G2)
    Priraď farbu množine MIS
    G := G - MIS
end while
```

V ideálnom prípade by pre vzniknuté k-farbenie $(V_1, V_2, ..., V_k)$ grafu malo platiť: ak V_1 je *MIS* grafu G, potom V_2 je *MIS* grafu $(G - V_1)$, V_3 je *MIS* grafu $(G - (V_1 \cup V_2))$ a podobne [9].

Algoritmus farbenia vrcholov grafu pomocou rekurzívnej *MIS* obvykle dosahuje lepšie výsledky ako Greedy algoritmus, avšak za cenu náročnejšej implementácie a horšej časovej náročnosti.

8.3 Zlepovací algoritmus

Farbenie grafu heuristickými algoritmami obvykle vedie na rozdelenie množiny vrcholov na nezávislé množiny. Predošlý algoritmus hľadal nezávislé množiny rekurzívne a nasledujúci využíva operáciu zlepovanie vrcholov [18].

Zlepovanie páru vrcholov grafu je operácia produkujúca nový graf, v ktorom sa dva vrcholy spoja do jedného. Výsledný vrchol je susedný so zjednotením susedných vrcholov každého z pôvodného páru vrcholov. Algoritmus farbenia pomocou zlepovania vrcholov funguje tak, že zlepí vrcholy, ktoré nemajú spoločných susedov, až dokým nevznikne úplný graf. Vrcholom v úplnom grafe sa pridelia rôzne farby a potom sa graf spätne rozloží na pôvodný, nezlepený graf. Tým pádom budú mať skupiny nesusedných vrcholov – nezávislé množiny – pridelené rovnaké farby.

Výber dvoch vrcholov, ktoré sa spoja sa môže zobrať náhodne, ale algoritmus dosahuje lepších výsledkov, ak sa vyberú dva vrcholy, ktoré majú najviac susedných vrcholov [20] a vrcholy sa usporiadajú podľa stupňa zostupne [17].

Veľmi dôležité je overiť úplnosť grafu. Keďže pri algoritmoch farbenia sa uvažujú neorientované prosté grafy, tak platí, že každá hrana má dva koncové vrcholy. S pomocou jednoduchých kombinátorských znalostí sa dá počet hrán neorientovaného grafu s n vrcholmi bez slučiek a multihrán spočítať nasledujúcim vzorcom:

$$|E| = C_2(n) = \binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2} \tag{13}$$

8.3.1 Algoritmus

Nech $G = (V, E)$ je graf, $V = v_1, ..., v_n$ je množina jeho vrcholov a $N(v)$ je množina všetkých susedných vrcholov vrcholu v. Zlepovací algoritmus má nasledujúci pseudokód:

Algoritmus 4 Zlepovací algoritmus

```
while G nie je úplný do
    for each v ∈ V do
        for each v' ∈ (V − N(v)) do
            Vyrob nový vrchol (v, v') tak, že N(v, v') = N(v') ∪ N(v)
            Zmaž pôvodné vrcholy v a v'
        end for
    end for
end while
Postupne priraď farby vrcholom úplného grafu
Rozlep graf na pôvodný
```

8.4 Tabu Col

Zatiaľ spomenuté algoritmy fungujú na základe heuristík. Nasledujúci algoritmus je metaheuristický a optimalizačný. Funguje na princípe Tabu search [21].

8.4.1 Tabu search

Tabu search patrí medzi metódy lokálneho vyhľadávania. Lokálne vyhľadávanie je princíp, ktorý kontroluje v okolí aktuálneho riešenia jeho najbližších susedov – v zmysle podobnosti – v nádeji, že niektoré zo susedných riešení môže byť o niečo lepšie ako aktuálne. Lokálne vyhľadávanie má tendencie uväzniť sa v lokálnych optimách.

Cieľom Tabu search je pohybovať sa iteračne od počiatočného riešenia obvykle kombinátorskej optimalizačnej úlohy k riešeniu, ktoré minimalizuje účelovú funkciu $f(s)$, kde $s \in S$ je riešenie z množiny riešení. Začne sa vygenerovaním počiatočného riešenia s a k nemu sa vygeneruje vzorka susedných riešení $N(s)$. $N(s)$ sú riešenia blízke s vo význame podobnosti. Z týchto riešení sa potom vyberie najlepšie riešenie $s*$. Výber riešenia závisí od kritéria akceptácie, ktoré sa obvykle odvíja od účelovej funkcie $f(s)$.

Aby sa výpočet nezastavil pri lokálnom optime, Tabu search má nasledujúcu vlastnosť: používa zoznam, ktorý obsahuje informácie o nedávno prejdených riešeniach a zakazuje návrat k nim. Aj toto kritérium sa zohľadňuje pri výbere $s*$. Výpočet pokračuje, pokiaľ sa nesplní nejaké ukončovacie kritérium. Môže to byť maximálny počet iterácii, dosiahnutie požadovanej hodnoty účelovej funkcie, dosiahnutie určenej kvality riešenia alebo akceptovanie určitého počtu zhoršení riešenia [21].

8.4.2 Popis Tabu Col

Tabu Col je optimalizačný algoritmus farbenia grafu, ktorý používa Tabu search. Počiatočné riešenie v tomto probléme je náhodne vygenerované farbenie grafu, ktoré dokonca nemusí byť uskutočniteľné.

Nech je $G = (V, E)$ graf. Počiatočné riešenie $s = \{V_1, V_2, ..., V_k\}$ je rozdelenie množiny vrcholov V na fixný počet k – tento počet sa neskôr môže znižovať. Vrcholy v jednotlivých V_i majú pridelenú rovnakú farbu. Ak $E(V_i)$ je množina hrán, ktoré sú spojené s V_i oboma koncami (majú rovnakú farbu), potom sa definuje účelová funkcia $f(s)$:

$$f(s) = \sum_{i=1}^{k} |E(V_i)| \tag{14}$$

Aby s bolo uskutočniteľné k-farbenie, musí platiť: $f(s) = 0$. Najlepšia hodnota tejto funkcie sa odhadne ako $f* = 0$. Z riešenia s sa vygeneruje množina susedov $N(s)$, napríklad tak, že sa v riešení s zmení farba jedného vrcholu na inú, ešte nepoužitú farbu, prípadne sa vyberie najmenejkrát použitá farba pri tomto vrchole. Tabu zoznam sa aktualizuje tak, že vždy pri zmene farby vrcholu x, sa pridá do zoznamu dvojica (x, i) vrcholu a jeho pridelenej farby. Vyberie sa najlepšie $s* \in N(s)$. Iteruje sa až kým sa nezíska riešenie s také, že $f(s) = f*$ alebo sa nepresiahne maximálny počet iterácií.

Vstupom do Tabu Col algoritmu je graf $G = (V, E)$, $k \in \mathbb{N}$ – počet farieb, rep – počet susedov vo vzorke a maximálny počet iterácií itmax [21].

Algoritmus 5 TABUCOL

Require: G=(V,E),k,$|T|$,rep,itmax
Ensure: k-farbenie grafu, ak $f(s) \geq 0$
Vygeneruj náhodne riešenie $s = \{V_1, ..., V_k\}$
it = 0
while $f(s) \geq 0$ and $it < itmax$ **do**
Vygeneruj susedov $N(s)$
Vyber $s*$ ako najlepšieho suseda
Aktualizuj zoznam T a zmaž najstaršiu tabu podmienku
$s := s*$
it ++
end while

8.5 Genetický algoritmus

Genetický algoritmus simuluje proces prirodzenej selekcie. Iba tí jedinci, ktorí sa dokážu prispôsobiť zmene prostredia, sú schopní prežiť a reprodukovať. Populácia sa skladá z jedincov, ktorí sú reprezentovaní chromozómami (skupinou génov) – napríklad reťazcami charov, integerov, floatov alebo bitov. Tieto kódy v rôznych optimalizačných problémoch predstavujú rôzne vlastnosti riešení. Na rozlíšenie silnejších jedincov od slabších je fitness funkcia. Každému jedincovi priradí hodnotu, na základe ktorej sa môže vybrať na párenie, pretože silnejší jedinci majú väčšiu šancu vytvoriť pomocou svojich génov lepších jedincov. Fitness funkcia pri optimalizácii problémov slúži na to, aby indikovala, ako blízko je chromozóm k žiadanemu riešeniu [22, 23].

Existuje niekoľko spôsobov selekcie jedincov na párenie. Turnajová selekcia funguje tak, že sa náhodne vyberie niekoľko jedincov a z nich sa vyberie ten s najlepšou hodnotou fitness. Ruletová selekcia je metóda, ktorá priradí každému jedincovi časť kolieska rulety podľa jeho fitness funkcie. Následne sa generuje náhodne číslo z intervalu (0,1) a vyberie sa jedinec, ktorému číslo spadá do jeho časti.

Ďalej nasleduje párenie. Keďže chromozóm je reprezentovaný poľom hodnôt, určí sa miesto kríženia, ktoré predstavuje index v poli. Na tomto indexe sa spojí časť genetickej informácie prvého jedinca a časť genetickej informácie druhého jedinca. Aby sa riešenie neuväznilo v lokálnom optime, tak ešte nasleduje procedúra mutácia. Noví jedinci môžu byť náhodne podrobení mutácii tak, že sa chromozóm zmení na jednom alebo viacerých miestach. Akonáhle sa párením silných jedincov nevytvára

lepšia populácia, populácia konverguje k riešeniu problému. Okrem toho zvykne byť použité aj nejaké ukončovacie kritérium. Zjednodušený pseudokód genetického algoritmu môže byť popísaný napríklad takto:

Algoritmus 6 Genetický algoritmus

```
Inicializácia náhodnej populácie
Vypočítanie fitness populácie
while Nesplnená ukončovacia podmienka do
    Selekcia najlepších jedincov
    Generácia nových jedincov pomocou párenia
    Mutácia nových jedincov
    Vypočítanie fitness nových jedincov
    Ak sú noví jedinci lepší, nahradia najhorších jedincov v populácii
end while
```

Problém farbenia grafov je tiež optimalizačný problém, pokiaľ sa prevedie na minimalizačnú úlohu, kde sa minimalizuje k v k-farbení [24, 25]. V inicializácii sa k označí ako horná hranica chromatického čísla a postupne sa znižuje, až kým sa neporuší uskutočniteľnosť farbenia. Cieľom algoritmu je nájsť také riešenie, kde žiadne dva susedné vrcholy nemajú pridelenú rovnakú farbu. Genetický algoritmus pokračuje, až kým také riešenie nenájde alebo sa prekročí maximálny počet iterácií.

Každý jedinec predstavuje nejaké k-farbenie grafu G, kde sa na začiatok k postaví rovné hornej hranici chromatického čísla podľa Vety 3.2.2: $k = \Delta(G) + 1$. Nech $G = (V, E)$ je graf s n vrcholmi a m hranami. Ak $c(i)$ je farba pridelená vrcholu i, potom $C = \{c(0), c(1), ..., c(n)\}$ je farbenie grafu. Prvá náhodná populácia je teda matica, kde sa v riadkoch nachádzajú farbenia C_j v tvare vektoru, ktorý obsahuje prirodzené čísla od 1 po k.

Nasleduje definícia fitness funkcie. Budú sa penalizovať farbenia grafu, ktoré obsahujú susedné vrcholy s rovnakými farbami. Najprv sa definuje funkcia $penalty(i, j)$, kde i, j sú dva rôzne vrcholy. Ak $c(i) = c(j)$ a vrcholy i, j sú susedné, potom $penalty(i, j) = 1$, v inom prípade $penalty(i, j) = 0$.

Fitness hodnota farbenia C sa spočíta ako súčet $penalty(i, j)$ cez všetky hrany grafu.

$$fitness(C) = \sum penalty(i, j) \tag{15}$$

Keďže cieľom algoritmu je nájsť uskutočniteľné farbenie grafu, silní jedinci sú v tomto prípade tí, ktorí majú čo najmenšiu hodnotu fitness funkcie a ideálne nulovú. Selekcia môže byť napríklad ruletová, teda na základe relatívnej fitness hodnoty medzi jednotlivcami a pravdepodobnosti.

Po selekcii jedincov nasleduje ich párenie. Na zvolenom indexe – vrchole – sa prehodia farby medzi jedincami a vznikne nové farbenie grafu. Mutácia nového jedinca zabezpečí, aby sa algoritmus nezasekol v lokálnom minime, takže s určitou pravdepodobnosťou sa vyberie vrchol, ktorému sa zmení farba. Šanca na mutáciu sa zvykne pohybovať do 5% alebo 10%. Pokiaľ sa podarí v iteračnom rozsahu nájsť uskutočniteľné k-farbenie (také, kde je hodnota fitness funkcie nulová), tak bola optimalizácia úspešná. Potom sa môže postupne k znižovať a hľadať dolnú hranicu chromatického čísla. Algoritmus nezaručuje nájdenie uskutočniteľného k-farbenia, ale upravením ukončovacieho kritéria sa za cenu prípadného dlhšieho výpočtu, dá šanca zvýšiť [24, 25].

8.6 Algoritmus lineárneho programovania

Lineárne programovanie je optimalizačná matematická metóda, ktorá umožňuje riešiť lineárne problémy s lineárnymi obmedzeniami.

Problém farbenia grafu sa môže previesť na problém lineárneho programovania, ktorý minimalizuje počet použitých farieb a zároveň nedovoľuje prideliť dvom susedným vrcholom rovnakú farbu [26, 27]. Nech $G = (V, E)$ je graf a $V = \{v_1, ..., v_n\}$ je množina jeho vrcholov. Ak vrchol v má pridelenú farbu i, potom premenná $x_{vi} = 1$, inak $x_{vi} = 0$. Použitie farby i reprezentuje premenná w_i. Ak je farba použitá, tak $w_i = 1$, inak $w_i = 0$. Najväčší potrebný počet farieb je označený H. Farbenie grafu $G = (V, E)$ sa dá upraviť na problém lineárneho programovania nasledujúcim spôsobom:

$$\min \sum\nolimits_{1 \leq i \leq H} w_i \tag{16}$$

$$\text{s.t.} \quad \sum\nolimits_{i=1}^{H} x_{vi} = 1 \qquad \forall v \in V \tag{17}$$

$$x_{ui} + x_{vi} \leq w_i \qquad \forall (u, v) \in E, i = 1, ..., H \tag{18}$$

$$x_{vi}, w_i \in \{0, 1\} \qquad \forall v \in V, i = 1, ..., H \tag{19}$$

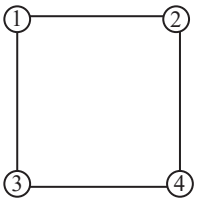
$$w_i \leq \sum\nolimits_{v \in V} x_{vi} \qquad i = 1, ..., H \tag{20}$$

$$w_i \leq w_{i-1} \qquad i = 2, ..., H \tag{21}$$

Objektová funkcia (16) predstavuje počet použitých farieb v grafe. Úlohou je ju minimalizovať, pričom riešenie musí spĺňať rad podmienok. Podmienka (17) slúži na to, aby sa zaručilo, že každý vrchol má pridelenú práve jednu farbu. Podmienka (18) zamedzuje, aby mali susedné vrcholy pridelenú rovnakú farbu. Podmienky (20) a (21) zaručujú to, aby farba i mohla byť pridelená vrcholu iba, ak farba $i - 1$ už je pridelená nejakému inému vrcholu [26, 27].

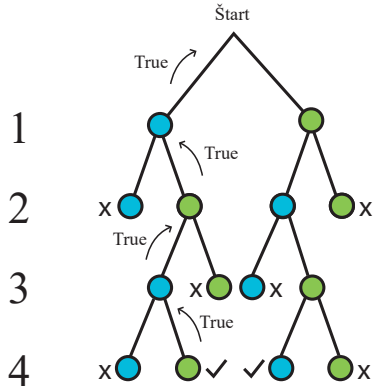
8.7 Backtracking algoritmus m-farbenia grafu

Backtracking je všeobecný postup, ktorý sa používa na hľadanie riešení niektorých výpočtových problémov. Prehľadáva stavový priestor. Stavový priestor je reprezentovaný stromovým grafom, kde jednotlivé vrstvy predstavujú rozhodnutia a koncové vrcholy samotné riešenia. Backtracking prehľadáva stavový priestor do hĺbky [9]. Ak zistí, že prehľadávaný stav nevedie na výpočet platného riešenia, tak ho upustí. Forma backtracking výpočtu bola uvedená aj pri rekurzívnom hľadaní maximálnej nezávislej množiny. Nasledujúci algoritmus poskytuje m-farbenie grafu, ak existuje [28]. Využíva na to prehľadávanie do hĺbky. Postupne skúša farby od 0 po m na vrcholoch tak, aby neboli v kolízii s farbami susedných vrcholov. Ak ku kolízii dôjde, tak sa farba zvýši, a pritom sa stále kontroluje, či už sa výpočet dostal k poslednému vrcholu. Pri dosiahnutí posledného vrcholu sa vráti `True` pre každé úspešné priradenie farby. V prípade, že sa počas farbenia zistí, že aktuálny vrchol nemôže byť ofarbený v rozsahu požadovaných farieb, zmení sa farba posledného vrcholu, kde je to možné. Prehľadávanie stavového priestoru bude naznačené na nasledujúcich obrázkoch.



Obr. 16: Vstupný graf

Bude sa kontrolovať, či existuje uskutočniteľné farbenie s použitím dvoch farieb. Stavový strom je naznačený na nasledujúcom obrázku, pričom vetvy, ktoré nevedú na uspokojivé riešenie, sú označené krížikom.



Obr. 17: Stavový priestor [28]

Čísla 1-4 predstavujú vrcholy, v ich vrstve je im priradená farba – buď modrá, alebo zelená. Z obrázka je zrejmé, že sú dve možné riešenia 2-farbenia grafu označené fajkou, pričom jediný rozdiel v nich je opačné priradenie farieb. Po získaní uskutočniteľného farbenia vráti algoritmus za každú vrstvu `True` a vráti sa z rekurzie.

9 Algoritmy hľadania kliky

Problém hľadania kliky sa delí na dva druhy problémov. Prvým problémom je výpočet maximálnej kliky v grafe a druhým problémom je zistiť, či pre $k \in \mathbb{N}$ existuje klika v grafe veľkosti aspoň k. Problémy majú rovnakú náročnosť, jediným rozdielom je pridaný parameter k. Problém kliky je tiež NP-úplný problém. Existujú rôzne metódy na určenie maximálnej kliky. Niektoré algoritmy budú predstavené v tejto kapitole [12].

9.1 Bron–Kerbosch algoritmus

Algoritmus Bron-Kerbosh bol navrhnutý v roku 1973 holandskými vedcami Coenraad Bron a Joep Kerbosch. Pracuje s troma množinami - P, R, X. P ako *possible candidates* je množina možných kandidátov vrcholov, ktoré môžu byť obsiahnuté v klike. R ako *temporary result* je množina aktuálneho výsledku – teda je to klika, ktorá ešte môže byť rozšírená o ďalšie vrcholy. Množina X obsahuje vrcholy, ktoré v klike obsiahnuté nebudú. Sú to napríklad vrcholy, ktoré už boli vylúčené z výpočtu kliky. Na začiatku množina P obsahuje všetky vrcholy grafu a zvyšné množiny sú prázdne. Funkcia Bron-Kerbosh pracuje tak, že prechádza postupne všetky vrcholy v P. Získa susedov aktuálneho vrcholu $N(v)$ a rekurzívne sa zavolá pre množiny $R = R \cup v, P = P \cap N(v), X = X \cap N(v)$. Ide o prehľadávanie do hĺbky, ktoré zisťuje, do akej vrstvy sú vrcholy susedné, pričom sa stále kontroluje, či sú množiny P a X prázdne [29, 30].

Algoritmus 7 Bron-Kerbosh algoritmus

```
function BRON-KERBOSH(R, P, X)
    if P = ∅ and X = ∅ then
        R je maximálna klika
    end if
    for each v ∈ P do
        BRON-KERBOSH(R ∪ v, P ∩ N(v), X ∩ N(v))
        P = P − v
        X = X ∪ v
    end for
end function
```

Bron-Kerbosh algoritmus existuje už pol storočia, počas ktorého bol obohatený heuristikami. Stále sa používa a dokonca ho niektorí znalci uprednostňujú pred alternatívami.

9.2 Bron–Kerbosch algoritmus s heuristikami

Základná heuristika, ktorá vylepšuje a zrýchľuje Bron-Kerbosh algoritmus, je výber vrcholu v cykle `for`, tzv. pivot. Pivot šetrí počet vnorení a tým aj potrebný čas na výpočet. Výber je založený na tom, že existujú dve možnosti pre každý vrchol v, a to: v sa nachádza v klike alebo jeden z jeho nesusedných vrcholov sa nachádza klike. Takže iba v a jeho nesusedné vrcholy musia byť otestované ako možnosti pre výber do kliky v rekurziách.

Ďalšie vylepšenie spočíva v tom, v akom poradí sa vrcholy spracujú. Na to sa využíva usporiadanie vrcholov podľa stupňa degenerácie (degeneracy ordering). Toto usporiadanie je v podstate usporiadanie vrcholov, ktoré sa získa opakovným odstránením vrcholu minimálneho stupňa v zostávajúcom podgrafe. Degenerácia grafu je najmenšie číslo d také, že každý vrchol má najviac d susedov, ktorí sú usporiadaní ďalej za ním. Idea, podľa ktorej je usporiadanie podľa stupňa degenerácie lepšie ako náhodné, spočíva v tom, že pri náhodom usporiadaní $|P|$ môže byť až tak veľké, ako maximálny stupeň grafu, ale pri usporiadaní podľa stupňa degenerácie nemôže byť $|P|$ väčšie ako degenerácia grafu. Degenerácia grafu je menšia ako maximálny stupeň grafu a dokonca býva táto hodnota nízka aj pri veľkých grafoch, ktoré predstavujú reálne problémy [29, 31].

Algoritmus 8 Bron-Kerbosh algoritmus

function BRON-KERBOSH2(R, P, X)
  if $P = \emptyset$ and $X = \emptyset$ **then**
    R je maximálna klika
  end if
  for each $v \in$ *degeneracyordering* **do**
    PIVOT($R \cup v, P \cap N(v), X \cap N(v)$)
    $P = P - v$
    $X = X \cup v$
  end for
end function

Bron–Kerbosch algoritmus na výpočet nezávislých množín

Vzhľadom na to, že vzťah kliky a *MIS* je komplementý, algoritmus Bron-Kerbosh sa dá jednoducho upraviť tak, aby jeho výstupom boli všetky maximálne nezávislé množiny v grafe. Stačí vstupný graf nahradiť jeho hranovým doplnkom. Algoritmus nájde množinu klík v G' a tým aj všetky nezávislé množiny v G.

9.3 Algoritmus lineárneho programovania

Problém maximálnej kliky sa dá tiež definovať pomocou lineárneho programovania s použitím modelu z článku [32]. Nech $G = (V, E)$ je graf, kde $V = \{v_1, ..., v_n\}$ je množina jeho vrcholov. Ak K je množina vrcholov kliky v G, potom sa definuje binárna rozhodovacia premenná x_i, ktorá nadobúda hodnotu 1, ak sa vrchol v_i nachádza v klike a 0, ak sa v nej nenachádza. Problém sa formuluje s počtom obmedzení $\mathcal{O}(n^2)$ týmto spôsobom:

$$\max \sum_{1 \leq i \leq n} x_i \tag{22}$$

$$\text{s.t.} \quad x_i + x_j \leq 1 \qquad \forall (v_i, v_j) \notin E \tag{23}$$

Formulácia je intuitívna a najbežnejšie používaná. Na definovanie nasledujúcej formulácie [33] s počtom obmedzení $\mathcal{O}(n)$ je nutné uviesť niekoľko notácii. Pre každý vrchol sa definuje množina všetkých nesusedov $NN(j)$ vrcholu v_j, pričom samotný vrchol v_j sa nepovažuje za nesuseda sebe samému. Veľkosť množiny $NN(j)$ sa označí:

$$h_j = \begin{cases} |NN(j)| & \text{ak } NN(j) \neq \emptyset \\ 1 & \text{ak } NN(j) = \emptyset \end{cases} \tag{24}$$

S využitím tejto premennej sa formuluje lineárny problém maximálnej kliky v grafe takto:

$$\max \sum_{1 \leq i \leq n} x_i \tag{25}$$

$$\text{s.t.} \quad h_j x_j + \sum_{i \in NN(j)} x_i \leq h_j \quad \forall j = 1, ..., n \tag{26}$$

10 Implementácia algoritmov

V tejto kapitole budú predstavené časti implementácie algoritmov riešiacich problémy vrcholového farbenia grafu a kliky v grafe v jazyku Python a GAMS. Pre všetky programové spracovania v Pythone bola použitá knižnica NetworkX [34], ktorá obsahuje dátové štruktúry a rôzne funkcie pre grafy. Vstupné grafy boli načítané z textových súborov ako zoznamy hrán alebo matice susedstva. Ďalej sa v implementáciach používa súbor graphtools.py, ktorý obsahuje niektoré funkcie, ktoré sa opakujú pri viacerých programoch. Kompletné implementácie sa nachádzajú v prílohe.

10.1 Greedy algoritmus

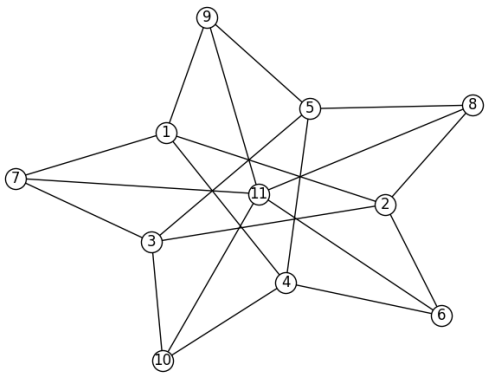
Heuristika DSatur je najzložitejšia zo spomínaných heuristík Greedy algoritmu, a preto je najvhodnejšie ukázať časť jej implementácie v jazyku Python. Vstupným parametrom je graf ako štruktúra knižnice NetworkX v Pythone. Výstupom je množina vrcholov s pridelenými farbami.

Výpis 1: DSatur Greedy coloring algoritmus

```
def GreedyColor(graph):

    lenColorArray = gt.maxNode(graph)+1
    minNode = gt.minNode(graph)

    colors = [NOT_DEF_COLOR]*lenColorArray

    list_of_nodes = list(graph.nodes) #pole vrcholov
    sat = [0]*lenColorArray #inicializacia pola saturácie
    sat[:minNode]=[NOT_DEF_COLOR]

    #vyberie sa prvý vrchol na farbenie náhodne:
    node = int(random.choice(list_of_nodes))

    while(True):
        #vrchol sa zmaže z množiny vrcholov
        list_of_nodes.remove(str(node))
        sat[node] = -1 #saturácia vrcholu sa označí ako nedostupná

        #pomocné pole na uloženie dostupných farieb
        available_colors = [True]*lenColorArray

        #nájde sa najnižšia volná farba, ktorú nemajú susedia
        #vrcholu
        for neighbor in nx.neighbors(graph,str(node)):
            if ((colors[int(neighbor)]) != NOT_DEF_COLOR):
```

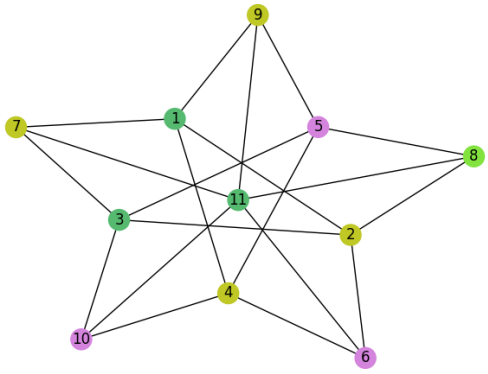
```
27                 available_colors[colors[int(neighbor)]] = False
28
29         cr = 0
30         #hladá sa najnižšia použitelná farba
31         while cr < lenColorArray:
32             if (available_colors[cr]):
33                 break
34             cr += 1
35
36         colors[(node)] = cr
37
38         if len(list_of_nodes)==0:
39             break
40         else: #výber dalšieho vrcholu
41             sat = Satur2(graph,colors,list_of_nodes,str(node),sat)
42             #vyberie sa vrchol s max saturáciou
43             node = np.argmax(sat)
44     return colors
```

Príkladom vstupu je nasledujúci 11-vrcholový graf s názvom myciel3 [35].



Obr. 18: Vstupný graf

Graf na obrázku bude definovaný zoznamom hrán. Inicializuje sa množina farieb a množina saturácie. Po načítaní grafu sa vyberie prvý vrchol náhodne a zmaže sa z množiny vrcholov. Pridelí sa mu najnižšia možná farba, v tomto prípade 0. Hodnoty saturácie susedov aktuálneho vrcholu sa aktualizujú a vyberie sa ten vrchol, ktorý má hodnotu najvyššiu. Opäť sa mu pridelí najnižšia možná farba a saturačné hodnoty sa aktualizujú. Proces pokračuje, až kým nie je farbenie kompletné. Na nasledujúcom obrázku sa nachádza výsledné farbenie grafu.



Obr. 19: Farbenie grafu s použitím DSatur

Výsledkom metódy je 4-farbenie grafu. Podľa [35] je chromatické číslo $\chi(G) = 4$. Algoritmus teda našiel minimálny počet farieb.

10.2 Rekurzívny algoritmus nezávislých množín

Algoritmus farbenia pomocou *MIS* obsahuje dve funkcie. Funkcia `getMisSet` obsahuje cyklus, ktorý postupne volá funkciu `getMis`, ktorá hľadá *MIS* z aktuálneho grafu. Nájdená *MIS* sa zmaže z aktuálneho grafu a výpočet sa opakuje, až kým graf neobsahuje žiadne vrcholy. Odporúča sa využiť náhodné prehádzanie vrcholov pred spracovaním a sledovať zmeny vo výsledkoch. Uvedený algoritmus je v jazyku Python. Vstupným parametrom je graf ako štruktúra NetworkX. Výstupom je množina vrcholov s pridelenými farbami.

Výpis 2: Rekurzívny algoritmus nezávislých množín

```
def getMis(graph):
    if len(graph) == 0:
        return []
    elif len(graph) == 1:
        return list(graph.nodes)

    graph2 = graph.copy() #kópia grafu

    allnodes = list(graph2.nodes)
    random.shuffle(allnodes) #pomiešanie vrcholov

    vertexCurrent = allnodes[0] #aktuálny vrchol

    #1.možnosť -> vrchol sa nenachádza v MIS
    graph2.remove_node(vertexCurrent) #vrchol sa zmaže
    sol1 = getMis(graph2) #1.riešenie

```

```
18      #2.možnosť -> vrchol sa nachádza v MIS -> zmazanie jeho susedov
19      for neighbors in graph.neighbors(vertexCurrent):
20          graph2.remove_node(neighbors)
21
22      sol2 = [vertexCurrent] + getMIS(graph2) #2.riešenie
23
24      #vyberie sa to riešenie, ktoré obsahuje viac vrcholov
25      return max(sol1, sol2, key=len)
26
27  def getMisSet(graph):
28
29      graphCopy = graph.copy()
30      misNodes = [] #inicializácia
31
32      #výpočet všetkých MIS
33      #cyklus, ktorý postupne nájde všetky MIS a zmaže ich
34      while True:
35          size = len(graphCopy)
36          if size == 1: #MIS je samotný vrchol
37              MIS = list(graphCopy.nodes)[0]
38              graphCopy.remove_node(MIS) #zmazanie vrcholu
39              misNodes.append([MIS]) #pridanie MIS do misNodes
40          elif size == 0:
41              break #koniec výpočtu
42          else:
43              MIS = getMis(graphCopy)
44              graphCopy.remove_nodes_from(MIS) #zmazanie MIS
45              misNodes.append(MIS) #pridanie MIS do misNodes
46
47      return misNodes
```

10.3 Zlepovací algoritmus

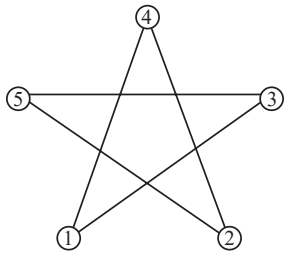
Nasledujúci algoritmus hľadá nezávislé množiny tak, že spája nesusedné vrcholy do skupín. V každej iterácii sa pre aktuálny vrchol vyberie nesusedný vrchol, ktorý má s aktuálnym najviac spoločných susedov. Vrcholy sa zlepujú, až kým nie je množina nesusedov aktuálneho vrcholu prázdna, potom sa prejde na ďalší vrchol. Výpočet skončí, keď je graf úplný. Program obsahuje funkciu `GetNextUnprocessedNod`, ktorá vyberá ďalší vrchol na spracovanie z tabuľky pravdivostných hodnôt. Vstupným parametrom je graf ako štruktúra NetworkX. Výstupom je množina vrcholov s pridelenými farbami.

Výpis 3: Zlepovací algoritmus

```
def ContractionColor(graph):
    graphCopy = graph.copy()
    #tabulka, podla ktorej sa vyberá nasledujúci vrchol na farbenie
    dispatchModeTable = [False]*(gt.maxNode(graph)+1)

    connections = []*len(graphCopy)
    #vrcholy sa zoradia podla stupňa zostupne
    allnodes = gt.DescendingDegree(graphCopy)

    while True :
        #aktuálny vrchol
        node = GetNextUnprocessedNod(allnodes,dispatchModeTable)
        if node == 'Nothing':
            break
        con = [] #pomocný list pre zápis
        con.append(node) #vloženie vrcholu do listu

        if(len(list(nx.non_neighbors(graphCopy,node)))) != 0:

            noNeighbor = 'init' #inicializačná hodnota
            while(noNeighbor != 'Nothing'):

                nonNeighbor = 'Nothing'
                commonNeighbors = []

                #vybrať najvhodnejšieho nesuseda - vyberie sa
                #nesused s najviac spoločnými susedmi
                max = 0
                for noNeighbor in nx.non_neighbors(graphCopy,node):
                    commonNeighbors = list(nx.common_neighbors(
    ↪ graphCopy, node, noNeighbor))
                    if len(commonNeighbors) > max:
                        max = len(commonNeighbors)
                        best = noNeighbor
                if max == 0 & len(commonNeighbors) == 0:
                    best = noNeighbor
                if noNeighbor == 'Nothing':
                    break #presun na ďalší vrchol
                else:
                    #susedia vrcholu best
                    bestNeighbors = list(nx.neighbors(graphCopy,
    ↪ best))
                    for neighbor in bestNeighbors:
                        #susedia best sa pridajú k vrcholu
                        graphCopy.add_edge(node, neighbor)

```

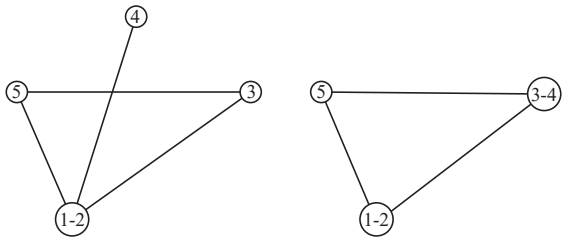
```
45                      #zmazanie vrcholu best
46                      graphCopy.remove_node(best)
47                      #zoradenia vrcholov podla stupňa
48                      allnodes = gt.DescendingDegree(graphCopy)
49                      con.append(best) #zlepenie vrcholov
50          connections.append(con)
51
52      #kontrola úplnosti grafu
53      if isCompleted(graphCopy):
54          return True, connections
55      else:
56          return False
```

Na nasledujúcom obrázku sa nachádza vstupný graf s 5 vrcholmi. Tento graf je pomerne jednoduchý, ale na ukážku fungovania algoritmu je vhodný.



Obr. 20: Graf hviezda

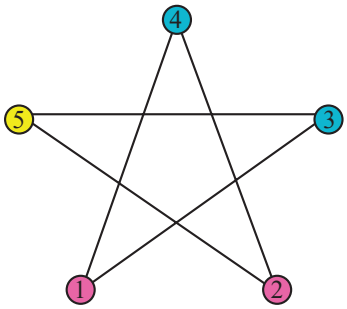
Nasledujú obrázky s postupom algoritmu. Pri zlepovaní vrcholov sa prioritne vyberajú tie nesusedné vrcholy, ktoré majú najviac spoločných susedných vrcholov.



(a) (b)

Obr. 21: Postup zlepovacieho algoritmu

Zlepené vrcholy predstavujú nezávislé množiny. Po dosiahnutí úplného grafu sa graf rozlepí na pôvodný a v rámci nezávislých množín sa vrcholom pridelia rovnaké farby.



Obr. 22: Farbenie grafu zlepovacím algoritmom

10.4 Genetický algoritmus

Genetické algoritmy sú skupinou optimalizačných algoritmov, ktoré sú založené na evolúcii. Zaoberajú sa minimalizáciou fitness funkcie, ktorá definuje konkrétny problém. V úlohe farbenia grafu je fitness funkcia postavená tak, aby penalizovala riešenia, v ktorých majú susedné vrcholy pridelené rovnaké farby. V prípade nulovej hodnoty fitness konkrétneho farbenia grafu G platí, že farbenie je uskutočniteľné. Na začiatku sa vygeneruje náhodná populácia riešení farbenia grafu pomocou k farieb, kde sa k nastaví ako horná hranica chromatického čísla. Ešte predtým sa však vyskúša položiť k rovné polovici hornej hranici. V prípade neúspechu sa k opäť nastaví na hornú hranicu a výpočet sa tým predĺži len o jednu iteráciu. Pomocou selekcie a mutácie sa vytvárajú nové riešenia tak, aby sa ich fitness hodnota vylepšovala. Ak sa nájde v populácii *individual*, pre ktorého platí $fitness(individual) = 0$, hodnota k sa nastaví na $k - 1$ a opäť sa použije genetický algoritmus. Pokiaľ sa za určitý počet iterácii nenájde individuál s nulovou fitness hodnotou pre farbenia s použitím k farieb, tak sa zistené chromatické číslo položí $\chi(G) = k+1$ [24]. Vstupným parametrom je graf ako štruktúra NetworkX. Výstupom je množina vrcholov s pridelenými farbami.

Výpis 4: Genetický algoritmus

```
def GeneticColoring(graph,max_num_colors,test_number):
   global popSize, selectionSize, maxIT #globálne premenné

    k = test_number

    valid = False
    bestInd  = {0}

    while(condition and k > 0):

        population = initPop(k, len(graph)) #init populácia
```

```
12          bestFitness = fitnessEval(graph, population[0]) #fitness
13          bestFitness = MAX_FITNESS_VAL #init najlepšieho jedinca
14          betterInd = population[0]
15          gen = 0
16
17          while(bestFitness != 0 and gen != maxIT):
18
19              gen += 1 #posun na noúu generáciu
20              #selekcia
21              population = rouletteSelections(graph, population)
22              newPop = []
23              random.shuffle(population)
24
25              for i in range(0, selectionSize-1, 2):  #párenie
26
27                  child1,child2 = crossover(graph,population[i],
    ↪ population[i+1])
28                  newPop.append(child1)
29                  newPop.append(child2)
30
31              for individual in newPop: #mutácia
32                  if(gen < 200):
33                      individual = mutation1(individual,k,len(graph))
34                  else:
35                      individual = mutation2(individual,k,len(graph))
36
37              population = newPop
38
39              #získanie jedinca s najlepšou hodnotou fitness
40              bestFitness = MAX_FITNESS_VAL
41              for individual in population:
42                  fitval = fitnessEval(graph, individual)
43                  if(fitval < bestFitness):
44                      bestFitness = fitval
45                      betterInd = individual
46                      #v populacii sa hladá uskutoč. k-farbenie
47                      if(bestFitness == 0):
48                          bestInd = betterInd
49                          valid = True #validita navr. hod
50                          break #výpočet nemusí pokračovať
51
52          #výpis
53          print("Počet farieb", k)
54          print("Generácia: ", gen)
55          print("Najlepsia hodnota fitness: ", bestFitness)
56          print("Najlepsi jedinec: ", bestInd)
57
```

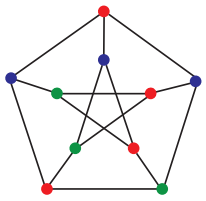
```
58          if(bestFitness != 0): #koniec výpočtu
59              break
60          else:
61              k -= 1 #zníženie počtu farieb
62
63      print("Validita: " ,valid, " Najlepsie farbenie: ",bestInd)
64      return valid,bestInd
```

10.5 Algoritmus lineárneho programovania

Problém farbenia grafu je možné implementovať v softvéri GAMS kvôli tomu, že sa dá upraviť na model lineárneho programovania [27]. V jednotlivých sekciách programu sú definované príslušné premenné a funkcie. V sekcii PARAMETERS je vstupný graf reprezentovaný pomocou matice susedstva $E(V1, V2)$. V sekcii VARIABLES je definovaná premenná $X(V1, C)$, pre ktorú platí:

$$X(V1, C) = \begin{cases} 1 & \text{vrchol } V1 \text{ má pridelenú farbu } C \\ 0 & \text{inak} \end{cases} \tag{27}$$

Ďalšou dôležitou premennou je $W(C)$, ktorá nadobúda hodnotu 1, ak je farba C použitá a 0, ak nie je. V sekcii EQUATIONS sú do prostredia GAMS implementované obmedzenia a objektová funkcia (16).



Obr. 23: Graf peterson

Výpis 5: GAMS

```
1  $TITLE Vertex colouring
2  $OFFSYMXREF
3  $OFFUELLIST
4  $OFFUELXREF
5
6  SETS
7      V1 vertices  /1*21/;
8      ALIAS(V1,V2);
9      ALIAS(V1,C);
10
11 PARAMETER
```

```
12      H;
13      H=CARD(V1);
14      * H = an upper bound of the number of the colours, at most |V|
15
16      TABLE E(V1,V2)
17          1  2  3  4  5  6  7  8  9  10
18      1   0  1  0  0  1  1  0  0  0  0
19      2   1  0  1  0  0  0  1  0  0  0
20      3   0  1  0  1  0  0  0  1  0  0
21      4   0  0  1  0  1  0  0  0  1  0
22      5   1  0  0  1  0  0  0  0  0  1
23      6   1  0  0  0  0  0  0  1  1  0
24      7   0  1  0  0  0  0  0  0  1  1
25      8   0  0  1  0  0  1  0  0  0  1
26      9   0  0  0  1  0  1  1  0  0  0
27      10  0  0  0  0  1  0  1  1  0  0
28
29  VARIABLES
30      X(V1,C)
31      * X(V1,C)=1 if vertex V1 is assigned to color C and X(V1,C) = 0
        ↪  otherwise
32      W(C)
33      * W(C)=1 if and only if color C is used in the colouring (i =
        ↪ 1,...,H).
34      Cmin objective function (minimal number of colours);
35      BINARY VARIABLE X;
36      BINARY VARIABLE W;
37
38  EQUATIONS
39      EQ2(V1)
40      EQ3(V1,V2,C)
41      EQ5(C)
42      EQ6(C)
43      OBJFCE number of colours;
44      EQ2(V1)                              ..SUM(C,X(V1,C))=E=1;
45      EQ3(V1,V2,C)$(ORD(V1) NE ORD(V2))..(X(V1,C)+X(V2,C)-W(C))*E(V1,
        ↪ V2)=L=0;
46      EQ5(C)                               ..W(C)-SUM(V1,X(V1,C))=L=0;
47      EQ6(C)$(ORD(C) GE 2)                 ..W(C)-W(C-1)=L=0;
48      OBJFCE                               ..Cmin=E=SUM(C,W(C));
49
50      MODEL COLOURING /ALL/;
51      SOLVE COLOURING USING MIP MINIMIZING Cmin;
52      DISPLAY X.L, W.L, Cmin.L;
```

Výstupom programu je matica X, ktorá reprezentuje, aké farby sú použité na vrcholy podľa (27), premenná W, ktorá vyjadruje použité farby a nakoniec počet farieb sa nachádza v minimalizovanej objektovej funkcii $Cmin$.

10.6 Algoritmus m-farbenia

Nasledujúci algoritmus nehľadá minimálny počet farieb, ale pomocou skúšania kombinácii zisťuje, či konkrétne m-farbenie grafu je uskutočniteľné.

Skladá sa z funkcie `backtrackColor`, ktorá volá funkciu `backtracking`, ktorá skúša prideliť vrcholom grafu m farieb tak, aby bolo farbenie uskutočniteľné. Na kontrolu používa funkciu `isSafe`. Tá kontroluje, či vrchol môže byť ofarbený konkrétnou farbou vzhľadom na svojich susedov.

Vstupným parametrom je graf ako štruktúra NetworkX a $m \in \mathbb{N}$. Algoritmus vráti farbenie grafu vo forme zoznamu vrcholov s priradenými farbami, ak existuje.

Výpis 6: Backtracking algoritmus

```
def isSafe(node, graph, color, c):
  for neighbor in nx.neighbors(graph,str(node)):
     if color[int(neighbor)] == c:
         return False
  return True

def backtracking(graph, m, color, v, firstNode):
  if v == len(graph) + firstNode:
        return True

  for c in range(1, m+1):
     if isSafe(v, graph, color, c):
         color[v] = c
         if backtracking(graph, m, color, v+1, firstNode):
             return True
         color[v] = 0
  return False

def backtrackColor(graph, m):
  allnodes = list(graph.nodes)
  firstNode = int(min(allnodes)) #nájdi najmenší vrchol:
  color = [0] * (len(graph) + (firstNode))
  if backtracking(graph, m, color, firstNode, firstNode):
        return color
  return None
```

10.7 Bron-Kerbosh algoritmus

Základný Bron-Kerbosh algoritmus je implementovaný podľa pseudokódu Algoritmus 7. Pracuje tak, že iteruje cez všetky možné podmnožiny vrcholov a kontroluje, či netvoria kliku. Jeho časová náročnosť je $O(3^{n/3})$. Vstupným parametrom je graf ako

štruktúra NetworkX. Výstupom funkcie je list, ktorý obsahuje všetky maximálne kliky v neorientovanom grafe a vykreslenie.

Výpis 7: Bron-Kerbosh algoritmus

```
def bron_kerbosch(graph):
  cliquesSET = []

  def cliques(R, P, X):
        if not P and not X:
            cliquesSET.append(R) #klika nájdená
          return

  for node in P.copy()
      neighbors = list(nx.neighbors(graph,node)) #susedia vrcholu
      cliques(R.union({node}), P.intersection(neighbors), X.
    ↪ intersection(neighbors))
      P.remove(node) #vrchol sa zmaže z množiny possible candates
      X.add(node) #vrchol sa pridá do množiny x

  cliques(set(), set(graph.nodes()), set()) #prvé zavolanie
  return cliques
```

10.7.1 Bron-Kerbosh algoritmus s heuristikou

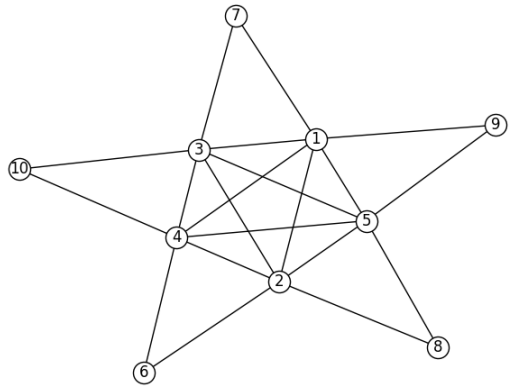
Jednoduchý Bron-Kerbosh algoritmus prehľadáva aj možnosti, ktoré môžu byť zbytočné. To samozrejme zvyšuje časovú náročnosť. Preto boli navrhnuté heuristické úpravy. Výber pivotu a nové usporiadanie vrcholov zmenšuje stavový priestor [36]. Vstupom je opäť graf ako štruktúra knižnice NetworkX a výstupom list maximálnych klík.

Výpis 8: Bron-Kerbosh algoritmus s heuristikou

```
def cliques(graph):

  P = set(graph.nodes) #possible candidates
  R = set() #temporary result
  X = set()
  cliques = []

    for node in degenOrdering(graph):
        neighbors = list(nx.neighbors(graph,node))
        pivot(graph, R.union([node]), P.intersection(neighbors), X.
    ↪ intersection(neighbors), cliques)
        P.remove(node)
        X.add(node)
  return sorted(cliques, key = lambda x: len(x), reverse=True)

def pivot(graph, R, P, X, cliques):

    if not P and not X:
        cliques.append(R) #našla sa klika
    else:
        u = next(iter(P.union(X)))
        Pdif = P.difference(list(nx.neighbors(graph,u)))
        for node in Pdif:
          neighbors = list(nx.neighbors(graph,node))
          pivot(graph, R.union([node]), P.intersection(neighbors),
    ↪ X.intersection(neighbors), cliques)
          P.remove(node)
          X.add(node)

def degenOrdering(graph):
    graphCopy = graph.copy()
    ordering_set = []

    nodes = sorted(graphCopy.degree, key=lambda x: x[1], reverse=
    ↪ False)
    while len(nodes)>0: #zmaže sa vrchol s najmenším stupňom
        min = nodes[0]
      graphCopy.remove_node(min[0])
      ordering_set.append(min[0])
      nodes = sorted(graphCopy.degree, key=lambda x: x[1], reverse=
    ↪ False)

  ordering_set.reverse()
  return ordering_set
```

10.8 Algoritmus lineárneho programovania

Algoritmus lineárneho programovania je implementovaný v prostredí GAMS. Vstupný graf je reprezentovaný maticou susedstva. Maximálna klika v grafe má 5 vrcholov.



Obr. 24: Vstupný graf

V sekcii PARAMETERS sa podľa modelu (25) získajú počty nesusedov pre každý vrchol a v sekcii VARIABLES sa implementujú obmedzenia a objektová funkcia. Úlohou je maximalizovať objektovú funkciu za použitých podmienok.

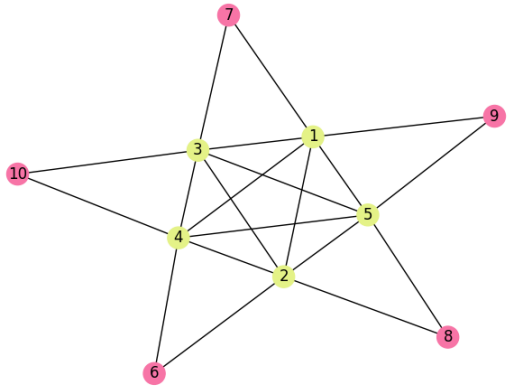
Výpis 9: GAMS

```
$TITLE Maximum clique
$OFFSYMXREF
$OFFUELLIST
$OFFUELXREF

SETS
    J vertices  /1*5/;
    ALIAS(K,J);
    ALIAS(I,J);

TABLE E(J,K)
        1   2   3   4   5   6   7   8   9   10
    1   1   1   1   1   1   0   1   0   1   0
    2   1   1   1   1   1   1   0   1   0   0
    3   1   1   1   1   1   0   1   0   0   1
    4   1   1   1   1   1   1   0   0   0   1
    5   1   1   1   1   1   0   0   1   1   0
    6   0   1   0   1   0   1   0   0   0   0
    7   1   0   1   0   0   0   1   0   0   0
    8   0   1   0   0   1   0   0   1   0   0
    9   1   0   0   0   1   0   0   0   1   0
    10  0   0   1   1   0   0   0   0   0   1;

```

```
TABLE NN(J,K) non-neighbours of vertex j;

PARAMETERS
    H(J) cardinality of non-neighbours set of vertex j;

    LOOP(J,
            H(J)=0;
            LOOP(K,
                    IF (E(J,K)=0,
                           NN(J,K)=1;
                           H(J)=H(J)+1;
                    ELSE
                           NN(J,K)=0;
                       );
                );
        );
    LOOP(J,
            NN(J,J)=0;
            H(J)=H(J)-1;
            IF (H(J)=0,
                 H(J)=1;
               );
         );
VARIABLES
      X(J)
      Cmax objective function (number of vertices in the maximum
   ↪ clique);
      BINARY VARIABLE X;

EQUATIONS
  EQ1(J)
  OBJFCE number of verices in the maximu clique;

  EQ1(J) .. H(J)*X(J)+SUM(I,X(I)*NN(J,I))-H(J) =L= 0;
  OBJFCE                          .. Cmax =E= SUM(J,X(J));

MODEL CLIQUE /ALL/;
SOLVE CLIQUE USING MIP MAXIMIZING Cmax;
DISPLAY NN, X.L, Cmax.L;
```

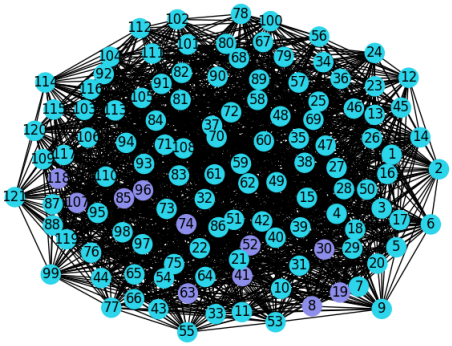
Výstup programu pozostáva z premennej X, ktorá obsahuje vrcholy, ktoré sa nachádzajú v maximálnej klike a výslednej hodnoty objektovej funkcie $Cmax$. Pre daný vstupný graf $Cmax = 5$, čo súhlasí s Obr.(24). Výsledkom výpočtu je jedna klika s maximálnym počtom vrcholov.

Na nasledujúcom obrázku je vstupný graf s vyznačenou klikou. Žlté vrcholy patria do maximálnej kliky. Maximálna klika má 5 vrcholov, čo súhlasí s hodnotou objektovej funkcie a premenná X obsahuje vrcholy {1,2,3,4,5}.



Obr. 25: Maximálna klika v grafe

Na nasledujúcom obrázku sa nachádza graf queen11_11 [35] so 121 vrcholmi. Maximálna klika nájdená Bron-Kerbosh algoritmom s heuristikami má 11 vrcholov a je vyznačená fialovou. Bron-Kerbosh algoritmus riešenie získal za 0.5 sekúnd, s heuristikami 0.15 sekúnd a Algoritmus lineárneho programovania za 1.6 sekundy.



Obr. 26: Graf queen11_11

Bron-Kerbosh algoritmus využíva rekurziu. Prechádza vrcholy grafu tak, aby našiel čo najviac navzájom susedných vrcholov. Výstupom algoritmu je množina maximálnych klík grafu. Použitím heuristiky sa niektoré vetvy stavového priestoru neberú do úvahy a tým sa skráti čas výpočtu. Pri počte vrcholov do 100 dokáže vypočítať riešenie do niekoľkých sekúnd a pri veľkých grafoch sa neúmerne zvyšuje časová náročnosť výpočtu. Rýchlosť výpočtu všetkých algoritmov na výpočet klík sa odvíja aj od hustoty grafu (28), pričom grafy s väčšou hustotou sú náročnejšie pre výpočet. Výpočet Algoritmu lineárneho programovania maximálnej kliky pozostáva z maximalizácie počtu vrcholov, ktoré môžu byť v klike. Výstupom je jedna klika s najväčším počtom vrcholov.

11 Porovnanie algoritmov farbenia

V tejto kapitole budú porovnané algoritmy farbenia grafov s použitím benchmark grafov [37, 35]. Merania boli vykonané na notebooku s CPU Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz a s RAM 16GB. Všetky algoritmy okrem Algoritmu lineárneho programovania používajú Python, takže výpočet sa predvolene vykonáva na jednom CPU jadre.

Dôležitý pojem je hustota grafu, ktorá vyjadruje pomer počtu hrán grafu k maximálnemu možnému počtu hrán – rovnica (13). Pre graf G s n vrcholmi a m hranami sa jeho hustota vyjadrí týmto vzťahom:

$$d(G) = \frac{2m}{n(n-1)} \tag{28}$$

Najprv sa porovnajú časovo menej náročné algoritmy na nasledujích grafoch:

Číslo	**Graf**	**Počet vrcholov**	$\chi(G)$	**d(G)**
1	flat1000_76_0	1000	76	0.49
2	le450_25	450	25	0.081
3	flat300_28_0	300	28	0.48
4	myciel7	191	8	0.13
5	miles1500	128	73	0.64
6	queen11_11	121	11	0.27

Tab. 2: Tabuľka benchmark grafov I

Algoritmus Greedy s heuristikami Random, Welsh-Powell a DSatur a zlepovací algoritmus boli použité na farbenie grafov z tabuľky Tab. 2. Získané počty použitých farieb k a časy výpočtu v sekundách sa nachádzajú v nasledujúcej tabuľke.

Graf G	$\chi(G)$	**Greedy R**		**Greedy W-P**		**Greedy DS**		**Zlep Algo**	
		k	čas	k	čas	k	čas	k	čas
1	76	127	0.485	123	0.404	114	94.7	104	181.1
2	25	28	0.018	25	0.032	25	0.555	25	21.6
3	28	47	0.014	45	0.035	42	2.798	39	6.456
4	8	9	0.005	8	0.005	8	0.128	8	2.739
5	73	74	0.011	73	0.01	73	0.428	73	0.348
6	11	17	0.005	19	0.004	16	0.099	14	0.667

Tab. 3: Tabuľka meraní I

Algoritmus Greedy s heuristikou Random a Welsh-Powell výpočet farbenia grafu zvládol najrýchlejšie. Vzhľadom na to, že vrcholom prideľuje farby za radom, nemusí sa priblížiť optimálnemu farbeniu, ako to aj vyplýva z meraní. Pri 1000 vrcholovom grafe sa počet použitých farieb k veľmi odlišuje $\chi(G)$. Pri grafe le450_25 s nízkou hustotou sa všetkým algoritmom podarilo nájsť chromatické číslo, okrem Greedy Random, ktorý sa mu ale veľmi priblížil.

DSatur a Zlepovací algoritmus vykazujú lepšie výsledky v rámci počtu farieb. Samotný Zlepovací algoritmus má najbližšie výsledky k chromatickému číslu, ale jeho výpočet je časovo náročnejší, pretože obsahuje veľa cyklov. Iba výber vrcholu na ďalšie spracovanie prechádza celý graf, pretože sa zakaždým vrcholy preusporiadajú zostupne podľa stupňa.

Na ďalšie testovanie boli využité grafy s menším počtom vrcholov:

Číslo	**Graf**	**Počet vrcholov**	$\chi(G)$	**d(G)**
1	myciel6	95	7	0.27
2	david	87	11	0.11
3	queen9_9	81	10	0.326
4	queen6_6	36	7	0.46
5	queen5_5	25	5	0.53
6	myciel4	23	5	0.28

Tab. 4: Tabuľka benchmark grafov II

Nasledujúca tabuľka obsahuje testovanie grafov z Tab. 4 na algoritme Greedy s heuristikami Random, Welsh-Powell a DSatur.

Graf G	$\chi(G)$	**Greedy R**		**Greedy W-P**		**Greedy DS**	
		k	čas	k	čas	k	čas
1	7	7	0.003	7	0.007	7	0.038
2	11	12	0.001	11	0.001	11	0.006
3	10	15	0.003	15	0.001	13	0.069
4	7	10	0.001	9	0.001	8	0.017
5	5	8	0.001	6	0.001	5	0.003
6	5	5	0.001	5	0.001	5	0.004

Tab. 5: Tabuľka meraní II/1

Greedy algoritmus pre všetky heuristiky zvládol výpočet farbenia týchto grafov veľmi rýchlo, pri nízkej hustote grafov sú počty farieb blízke až rovnaké chroma-

tickému číslu. Nevýhodou Greedy algoritmu je jeho nespoľahlivosť v hľadaní chromatického čísla. Najlepšia heuristika vzhľadom na počet farieb je DSatur.

Nakoniec budú porovnávané algoritmy Zlepovací, Algoritmus farbenia pomocou nezávislých množín, Genetický algoritmus a Algoritmus lineárneho programovania.

Graf G	$\chi(G)$	**Zlep Algo**		**Rec MIS**		**Gen Algo**		**Lin algo**	
		k	čas	k	čas	k	čas	k	čas
1	7	7	0.355	-	-	8	13633.7	-	-
2	10	11	0.300	-	-	12	7663.2	11	2.895
3	10	11	0.216	10	134.7	13	8596.8	10	545.8
4	7	8	0.025	10	0.442	8	865.8	7	11.3
5	5	5	0.014	5	0.079	5	154.2	5	3.281
6	5	5	0.009	5	0.293	5	59.9	5	3.240

Tab. 6: Tabuľka meraní II/2

Zlepovací algoritmus vykazuje dobré a rýchle výsledky pre väčšie aj menšie grafy. Algoritmus farbenia pomocou nezávislých množín je rekurzívny a pri väčších grafoch v reálnom čase riešenie nemusí nájsť. Jeho presnosť je ovplyvnená aj tým, že v každom vnorení vyberá väčšiu z dvoch množín. Pokiaľ sú obe množiny rovnako veľké, jedna z nich je odignorovaná, aj keď môže viesť nakoniec k lepšiemu riešeniu. Počet vrcholov nie je jediná vlastnosť, ktorú je nutné zvážiť pred použitím rekurzívneho algoritmu – grafy stromového typu sú menej vhodné.

Genetický algoritmus je možné prispôsobiť počtom iterácií a veľkosťou populácie a tým zvýšiť šancu na lepší výsledok, ale na druhej strane počet iterácií zvyšuje časovú náročnosť výpočtu. Jednotlivé k-farbenia generuje náhodne, optimalizuje ich selekciou, krížením a mutáciou, ale aj tak nemusia viesť k požadovanému výsledku. Algoritmus lineárneho programovania bol implementovaný v prostredí GAMS. Algoritmus nemal problémy s menšími grafmi, pri hustejších bol výpočet časovo náročnejší. Pri grafe myciel6 ani v zvýšenom časovom limite na 10 000 s nezískal konečný výsledok.

Grafy s nízkou hustotou nie sú výzvou takmer pre žiadne algoritmy. Výnimkou je Genetický algoritmus, ktorý riešenia generuje a optimalizuje, takže hustota grafu neovplyvňuje jeho výsledok. Podobne rekurzívny algoritmus vzhľadom na prehľadávanie veľkého stavového priestoru. Pred riešením problému farbenia grafu sa odporúča zistiť počet vrcholov a hustotu grafu. Na veľké grafy s nízkou hustotou stačí jednoduchý Greedy algoritmus – ideálne Welsh-Powell alebo DSatur. Na ostatné grafy je potrebné použiť zložitejší algoritmus a očakávať dlhší výpočet na získanie uskutočniteľného riešenia.

12 Aplikácie farbenia grafov

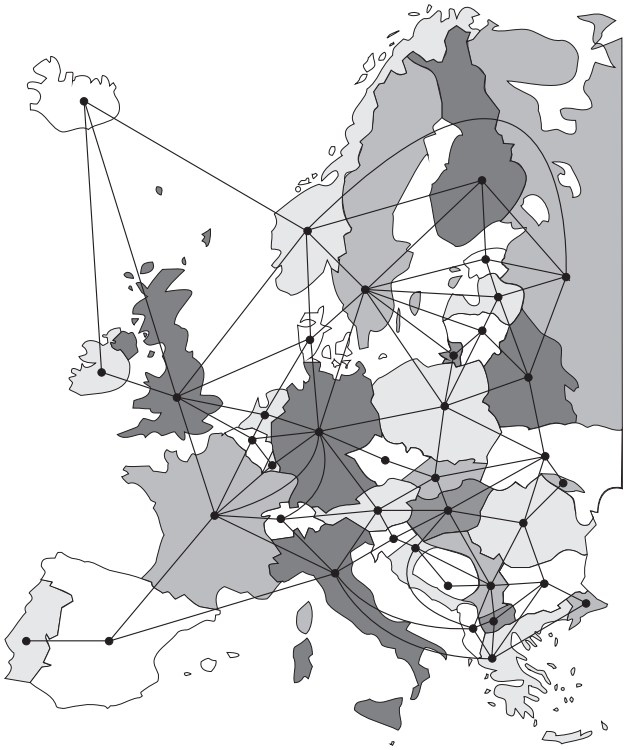
Všeobecne sa grafy používajú na modelovanie prekvapivo veľkého množstva problémových oblastí. Farbenie grafov sa okrem problému máp využíva v chémii, elektrickom inžinierstve, počítačových sieťach, satelitnej navigácii, ale aj pri donáškach balíkov, sociálnych sieťach a všeobecne v plánovaní.

12.1 Farbenie máp

Farbenie mapy je najpriamejšia aplikácia farbenia grafu. Začalo sa skúmať v 19. storočí v súvislosti s problémom "farebných máp". Tento problém bol spočiatku riešený manuálne, ale neskôr sa začali používať matematické prístupy. Cieľom bolo vytvoriť mapu, kde každá oblasť bude mať svoju vlastnú farbu, pričom susedné oblasti budú mať rôzne farby. Najprv sa jednotlivé krajiny alebo regióny v krajine nahradia vrcholmi umiestnenými v ich strede. Regióny, ktoré zdieľajú hranicu, sa spoja hranou. Zistilo sa, že hrany sa dajú nakresliť vždy tak, aby sa nekrížili – každá mapa sa dá previesť na planárny (rovinný) graf. Podľa vety 3.2.4 platí, že na farbenie akejkoľvek mapy stačia štyri farby, ale cesta k tomuto tvrdeniu bola pomerne dlhá.

Datuje sa späť do roku 1852. Francis Guthrie bol anglický matematik, ktorý sa jedného dňa snažil vyfarbiť mapu Anglicka. Všimol si, že na to stačili štyri farby. Spolu so svojím bratom si nad tým lámali hlavy. Neskôr v roku 1878 bol problém uvedený pre verejnosť ako hádanka. O rok neskôr Alfred Kempe vydal prvý dôkaz, ktorý nikto nespochybnil až 10 rokov. V 1880 Peter Tait vydal iný dôkaz, ktorý ale bol vyvrátený. Jeho aj Kempeho dôkazy však mali prínosy (napríklad Kempeho reťaze). O viac ako 40 rokov neskôr sa americkému matematikovi Phillipovi Franklinovi podarilo dokázať vetu o štyroch farbách pre mapy s najviac 25 plochami. Ďalší matematici sa pomocou jedného zo starších neúspešných dôkazov dostali k myšlienke redukovateľnosti a vybíjania náboja, ktoré sú základnými myšlienkami konečného dôkazu. V 1976 Kenneth Appel a Wolfgang Haken publikovali svoj dôkaz vety o štyroch farbách, ktorý doteraz nebol spochybnený. Niektoré časti dôkazu sú príliš náročné a zdĺhavé a na ich overenie je nutný počítač [3].

Napríklad mapa Európy sa dá previesť na graf takto:



Obr. 27: Mapa Európy [9]

12.2 Plánovanie

Farbenie grafu sa dá aplikovať na rôzne úlohy plánovania. Môžu to byť problémy priraďovania úloh rôznym členom tímu, plánovanie odchodov vlakov, autobusov, odletov lietadiel. V grafovom modeli plánovania vrcholy grafu predstavujú úlohy, ktoré sa majú plánovať, a hrany reprezentujú závislosti medzi nimi. Cieľom je priradiť farbu (napríklad pracovníka) každej úlohe tak, aby sa zohľadnili možné kolízie úloh. Algoritmy farbenia grafov sú schopné efektívne riešiť tieto problémy, minimalizovať čas a náklady a zlepšiť celkovú efektivitu plánovania [38].

12.3 Plánovanie rozvrhov

Plánovanie školských rozvrhov je ďalšia aplikácia farbenia grafov. V problémoch tohto typu je daná množina udalostí ako napríklad prednášok, testov alebo triednych schôdzok s množinou časových úsekov. Problémom je priradiť udalosti časovým úsekom v súlade s obmedzeniami tak, aby nedošlo k nežiadúcemu stretu udalostí. Napríklad, ak má aspoň jeden študent zapísané dva predmety, nesmie sa stať, že

budú prebiehať v rovnaký čas. Problém rozvrhu sa dá previesť na problém farbenia grafu jednoducho. Vrcholy grafu budú predstavovať udalosti a spojené budú tie udalosti, ktoré nemôžu prebiehať súčasne. Následne je úlohou nájsť uskutočniteľné farbenie grafu s čo najnižším počtom farieb. Farby predstavujú časové úseky. Skupiny vrcholov s rovnakou farbou sú teda tie udalosti, ktoré môžu nastať v rovnakom čase bez kolízií. Problém je variabilný, nemusí sa zaoberať práve kolíziou študentov, ale tiež miestností, kde môžu dané udalosti prebiehať, prípadne kolíziou učiteľov [9].

12.4 Uskladnenie chemických látok

Okrem problému máp je ďalšia veľmi známa aplikácia farbenia grafov uskladnenie chemických látok. Nech firma vyrába n chemikálií označených $C_1, ...C_n$. Niektoré dvojice chemikálií sú nekompatibilné a nesmú byť umiestnené pri sebe, nakoľko by mohlo dôjsť k nežiaducim reakciám. Aby k nim nedošlo, firma potrebuje navrhnúť sklad na tieto látky tak, aby bol rozdelený do niekoľkých častí, kde môžu byť chemikálie bezpečne uložené. Problémom je zistiť, aký je najmenší počet priehradok v sklade, aby sa firma vyvarovala možným rizikám. Modelom tejto problematiky bude graf G s vrcholmi $V = \{v_1, ...v_n\}$, kde vrcholy predstavujú chemické látky. Vrcholy v_i a v_j sú susedné, ak látky C_1 a C_2 sú nekompatibilné a nesmú byť umiestnené pri sebe. Chromatické čislo grafu G predstavuje počet skupín, do ktorých sa musia rozdeliť chemické látky tak, aby nedošlo k nežiaducim následkom. Tieto skupiny sú vlastne nezávislé množiny vrcholov. Úloha by sa týmto pozmenila na hľadanie najmenšieho počtu nezávislých množín [8].

12.5 Sudoku

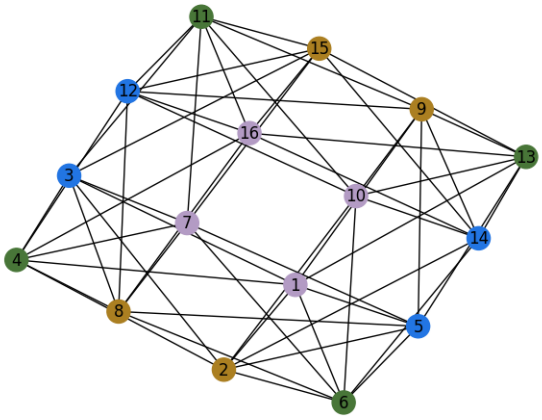
Zaujímavou matematickou oblasťou, ktorá môže využiť farbenie grafov, sú latinské štvorce. Latinský štvorec je mriežka $n \times n$ obsahujúca n symbolov, ktoré sa v riadkoch ani stĺpcoch neopakujú. Samotné latinské štvorce sa môžu použiť na dizajn experimentov, generáciu hesiel, pri plánovaní alebo v sudoku. [9].

Sudoku je známa logická hra pre jedného hráča. Hrací plán sudoku sa skladá z tabuľky o veľkosti 9×9 políčok, do ktorých sa dopisujú čísla od 1 po 9. Niektoré políčka sú predvyplnené a cieľom hráča je doplniť zvyšné tak, aby v každom riadku, stĺpci a podoblastiach 3×3 bola každá číslica práve jedenkrát. Riešiť sudoku je možné aj pomocou vrcholového farbenia grafu. Hádanka obsahuje niekoľko vyznačených hodnôt tak, aby riešenie bolo vždy jednoznačné, ale ukážkový príklad bude zameraný len na vyplnenie prázdnej tabuľky veľkosti 4×4. Jednotlivé políčka budú označené indexami od 1 po 16 ako na nasledujúcom obrázku:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Obr. 28: Sudoku tabuľka 4×4

Prevod sudoku tabuľky na graf je jednoduchý. Vytvorí sa graf tak, aby indexy z Obr.28 predstavovali vrcholy. Hranami budú prepojené práve tie vrcholy, ktoré sa nachádzajú v rovnakých riadkoch, stĺpcoch a 2×2 vyznačených podoblastiach. Algoritmus farbenia pomocou zlepovania vrcholov následne rozdelí vrcholy do štyroch skupín tak, aby v žiadnej skupine neboli dva susedné vrcholy. Skupinám sa potom pridelia farby. Výsledné farbenie grafu je uskutočniteľné a vzniknutý graf je znázornený na nasledujúcom obrázku:



Obr. 29: Graf sudoku

Políčka v sudoku sa nafarbia podľa grafu a farby sa nakoniec nahradia číslicami, tak, ako sme zvyknutí sudoku riešiť. Riešenie 4×4 sudoku tabuľky je znázornené na nasledujúcom obrázku.

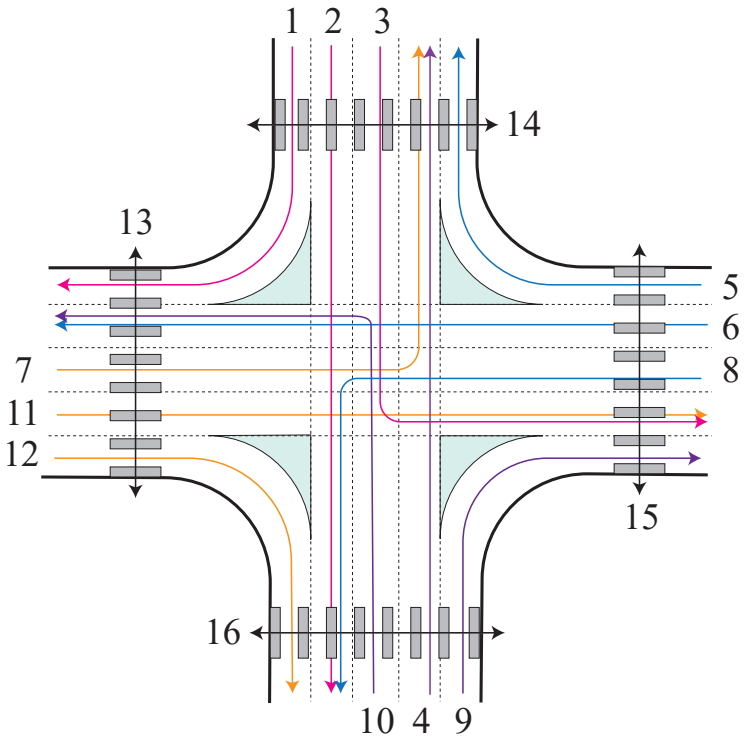
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Obr. 30: Jedno z možných riešení sudoku

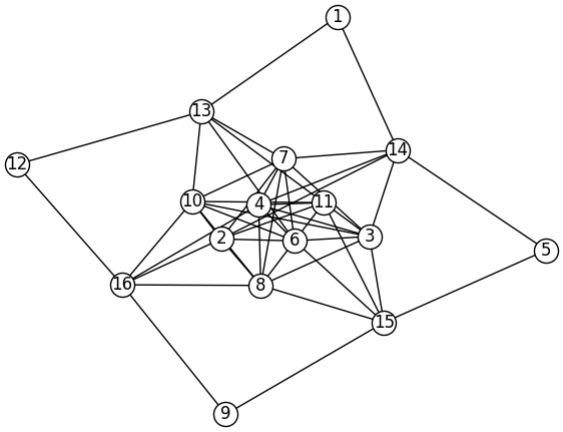
12.6 Fázovanie svetelne riadenej križovatky

S narastajúcim počtom vozidiel sa dopravná premávka stáva problémom každej krajiny. Z tohoto dôvodu je potrebné optimalizovať algoritmy regulácie svetelných križovatiek, aby sa zamedzilo zdržiavaniu premávky. Neraz sa vo veľkých mestách nachádzajú zložité svetelné križovatky s veľa prúdmi a možnosťami pohybu. Pokiaľ križovatka pretína štvorprúdové cesty, ktoré naviac obsahujú električkové trate, na prvý pohľad nie je zrejmé, ako ju čo najlepšie usmerniť. Prúdy sa rozdelia do niekoľkých fázových skupín tak, aby postupne dostávali zelenú. Zároveň, v rámci jednej skupiny prúdov musí byť pohyb vozidiel bezkolízny. Medzi ukončením jednej fázy a začatím druhej musí existovať nejaká doba, ktorá slúži na vyprázdnenie križovatky. Optimalizácia tohto problému by sa skladala z minimalizácie počtu fáz a zároveň minimalizácie času, ktorý je potrebný na vyprázdnenie križovatky po každej fáze [39, 40]. Keďže opäť ide o kombinátorskú úlohu s určitými kolíziami, vrcholové farbenie grafu sa javí ako užitočné. Vrcholy grafu sú v tomto prípade jednotlivé prúdy a spojené su tie, medzi ktorými existuje relácia kolíznosti - dva prúdy sú kolízne, ak ich dráhy majú spoločný bod. Vozidlá nemôžu vchádzať do kolíznych prúdov naraz, pretože by mohlo dôjsť k nárazu. Riešením je rozdeliť prúdy do fáz - skupiny prúdov, ktoré v semafóre naraz dostanú zelenú. Na nasledujúcom obrázku je uvedený príklad križovatky, ktorej riešenie je jednoduché a na znázorňovacie účely postačí.



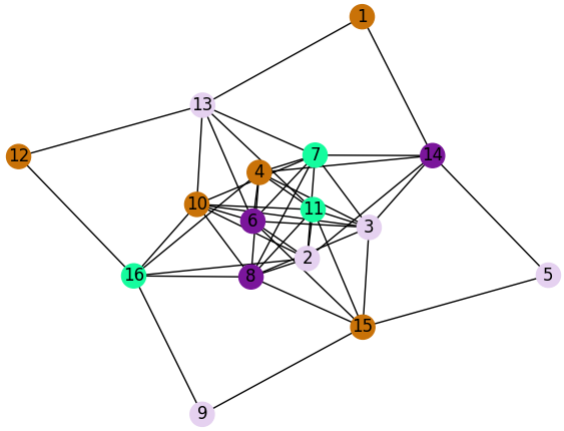
Obr. 31: Križovatka

V prvom kroku sa križovatka prevedie na model grafu, kde vrcholy predstavujú prúdy a hrany predstavujú reláciu kolíznosti. Na nasledujúcom obrázku sa nachádza graf popisujúci danú križovatku:



Obr. 32: Model križovatky

Na graf sa použije algoritmus farbenia, ktorý prúdy rozdelí do fáz. Prúdy v rámci fázi môžu mať naraz zelenú, pretože nie sú kolízne. Na nasledujúcom obrázku sa nachádza vrcholové farbenie grafu s použitím 4 farieb.



Obr. 33: Výsledné rozdelenie prúdov

Samotný algoritmus farbenia grafu nezaručuje, že sa križovatka správne vyrieši. Pokiaľ program nie je špeciálne prispôsobený tomuto problému, je nutné výsledné nezávislé množiny skontrolovať, aby sa zistilo, či sa riešenie môže použiť. Ak by totiž naraz zelenú dostali len chodci, prípadne nejaká fáza by bola tvorená len jedným prúdom, riešenie by síce z matematického hľadiska bolo správne, ale v realite by sa určite nepoužilo. Jedno z možných riešení je pridať hrany medzi všetky

prúdy, ktoré by nemali ísť naraz, prípadne ošetriť niektoré možnosti. Okrem toho, pri riadení križovatky musí byť ešte zohľadnený čas nutný na vyprázdnenie križovatky medzi jednotlivými fázami a najlepšie poradie, v akom by mali dostať fázy zelenú. Táto úloha sa zvykne previesť na úlohu obchodného cestujúceho.

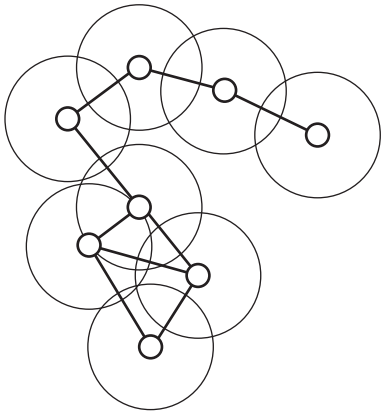
12.7 Alokácia registrov

Kompilátor je programovací nástroj, ktorý slúži na prekladanie zdrojového kódu vo vyššom programovacom jazyku na strojový kód. Vyššie programovacie jazyky umožňujú použitie veľkého až neobmedzeného počtu premenných, zatiaľ čo typický mikroprocesor umožňuje efektívny prístup len k malému počtu premenných, nazývaných registre, a potom menej efektívny prístup do hlavnej pamäte. Z dôvodov efektívnosti je motiváciou umiestniť čo najviac premenných do registrov a čo najmenej do pamäte, pretože prístup k registrom a aktualizácia ich hodnôt sú oveľa rýchlejšie. Procesory majú iba obmedzený počet registrov. Niektoré premenné sa v počítačovom programe používajú len v určitom čase. Pokiaľ sa to dá, je výhodné priradiť jednému registru viacero premenných tak, aby sa navzájom nerušili. Registrový konflikt nastane, ak sa dve premenné používajú v rovnakej inštrukcii a boli priradené do rovnakého registra. V tomto prípade musí procesor uložiť jeden z výsledkov do pamäte, čo môže spomaliť výkon programu.

Modelom tohto problému je graf, kde vrcholy predstavujú jednotlivé premenné. Hrany grafu reprezentujú závislosti medzi premennými a inštrukciami. Ak hrana vedie z vrcholu A do vrcholu B, znamená to, že premenná reprezentovaná vrcholom A sa používa v inštrukcii, ktorá používa premennú reprezentovanú vrcholom B. Cieľom problému je minimalizovať počet registrov, ktoré sú potrebné na uloženie všetkých premenných a medzivýsledkov [9].

12.8 Prideľovanie frekvencií

Prideľovanie frekvencií je aplikácia farbenia grafu, ktorá sa zaoberá rádiovými stanicami. Predpokladom je, že existuje niekoľko rádiových staníc na ploche so súradnicami x a y. Problémom je prideliť frekvencie každej stanici tak, aby stanice, ktoré sú blízko seba, mali rôzne vysielacie frekvencie. V opačnom prípade by mohol nastať šum v prijímači kvôli interferencii. Tento problém sa podobá na problém farbenia máp, avšak s menším rozdielom. Ak by bolo veľa rádiových staníc v malom regióne, boli by všetky príliš blízko seba a tým pádom by tvorili kliku. Preto sa na riešenie frekvencií používa jednotkový diskový graf. Každý vrchol má okolo seba znázornený kruh s jednotkovým priemerom, ktorý naznačuje dosah frekvencie. Ak sa dva kruhy staníc prelínajú, tak sú regióny, v ktorých sa nachádzajú, susedné [38].



Obr. 34: Jednotkový diskový graf

13 Aplikácie kliky v grafe

Klika je úplný podgraf. Všeobecne, ak vrcholy nejakého grafu predstavujú dáta a hrany reláciu podobnosti, klika môže reprezentovať skupinu predmetov, ľudí alebo udalostí s podobnými vlastnosťami. Vrcholové farbenie sa zvykne zaoberať úlohami, ktoré sa snažia minimalizovať kolízie a klika naopak hľadá podobnosť. Má mnoho aplikácii v problémoch selekcie, klasifikácie, počítačovom videní, ekonómii, sociálnych sieťach aj v biológii [41].

13.1 Clustering

Dátový clustering alebo zhlukovanie sa zaoberá delením dát na podmnožiny (zhluky), pričom objekty v rovnakých podmnožinách sú podobné vo význame nejakej skúmanej vlastnosti. Využitie kliky na zhlukovanie dát je užitočné najmä v prípadoch, keď sú dáta reprezentované grafom a je nutné identifikovať skupiny podobných dát. Vrcholy sú reprezentované dátovými bodmi a algoritmus kliky identifikuje podobné skupiny. Veľa zhlukovacích algoritmov je založených na hľadaní kliky konkrétnej veľkosti [42]. Kliky však môžu byť veľké a zložité a keďže podstata väčšiny algoritmov je backtracking, výpočet to značne spomaľuje.

13.2 Počítačové videnie

Klika v grafe sa používa v počítačovom videní ako jeden z nástrojov pre detekcie a rozpoznanie objektov v obraze. Segmentácia pomocou Markovských náhodných polí je metóda automatického rozdelenia obrazu na základe pravdepodobnosti. Obraz sa vníma ako mriežka bodov, kde každý bod reprezentuje pixel obrázka. Každý pixel je klasifikovaný do jednej z niekoľkých tried, ktoré zodpovedajú rôznym oblastiam obrázka. Cieľom je určiť pravdepodobnosti priradenia každého pixelu do určitej triedy na základe pravdepodobností priradenia susedných pixelov do rovnakej triedy. Kliky sa môžu použiť na definovanie podmienenej pravdepodobnosti, ktorá popisuje vzťahy medzi pixelmi a ich okolím. Využitie klík umožňuje modelovať komplexnejšie vzťahy medzi pixelmi, ako napríklad závislosti na textúre alebo farbe [43, 44].

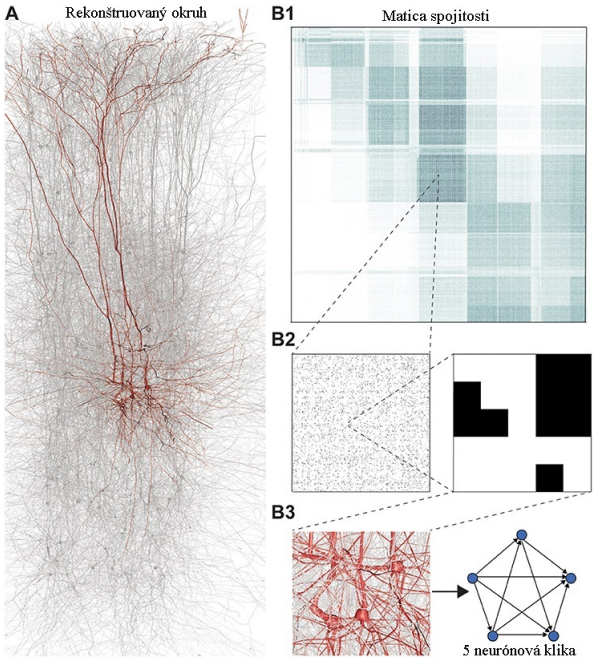
13.3 Kliky v biológii a lekárstve

Kliky majú rôzne aplikácie v biológii, napríklad sa dajú použiť pri porovnávaní proteínov. Práca [45] sa zaoberá vytvorením zásuvného modulu pre softvér Cytoscape. Proteíny reprezentované grafmi sa porovnávajú pomocou algoritmu založenom na hľadaní maximálnej spoločnej kliky. Porovnávanie proteínov môže biológom pomôcť

skúmať predpokladané funkcie neznámych proteínov, prípadne ich použitie v liečivách.

V práci [46] je navrhovaný algoritmus, ktorý by mohol automatizovať hľadanie mozgových nádorov pomocou klík. Obvykle sa na túto problematiku využívajú neurónové siete, ktoré sa natrénujú na veľkom množstve snímok z magnetickej rezonancie. Algoritmus, ktorý používa kliky, sa zaoberá segmentáciou obrazu na graf. Prvým krokom algoritmu je zostrojenie susedstiev na základe vzdialenosti a podobných vlastností. Potom sa hľadajú všetky kliky s konkrétnou veľkosťou k na zistenie korelácii v pixeloch, podľa ktorých by sa mohol identifikovať nádor.

Doteraz nie je úplne známe, ako presne mozog spracováva informácie. Práca [47] využíva grafovú topológiu na reprezentovanie smeru toku informácií v synapsiách. Graf nervovej siete je orientovaný, jeho vrcholy predstavujú neurónové bunky a hrany synaptické spojenia, po ktorých sa šíria vzruchy. Nervové siete sa často analyzujú z hľadiska vrcholov, ktoré sú navzájom prepojené, teda klík. Keď sa orientované kliky vhodne naviažu zdieľaním neurónov, ale nevytvoria veľké kliky kvôli chýbajúcim spojeniam, vytvárajú tzv. dutiny. Orientované kliky opisujú tok informácií v sieti lokálne a dutiny poskytujú globálnu mieru toku informácií. Na nasledujúcom obrázku sa nachádza prevod nervovej sústavy na grafovú štruktúru.



Obr. 35: Nervová sústava [47]

Charakterizácia chemických zložiek komplexných roztokov je veľmi dôležitá v oblastiach biochémie. Nukleárna magnetická rezonancia je nástroj, ktorý sa používa na analýzu roztokov bez nutnosti fyzickej separácie ich zložiek. Jej použitie

sa ukázalo ako veľmi užitočné na identifikáciu známych metabolitov, ale aj neznámych metababolitov. Výsledné spektrá sú komplikované a ich analýzu je výhodné automatizovať. Pre tento účel sa dá použiť algoritmus hľadania klík v grafe, ktorý reprezentuje spinové systémy [48].

13.4 Kliky v sociálnych sieťach

Sociálna sieť predstavuje množiny vzťahov medzi členmi sociálnych systémov vo všetkých významoch. Analýza sociálnych sietí skúma interakcie medzi individuálmi a organizáciami. Veľkým medzníkom v študovaní sociálnych sietí bol experiment amerického psychológa Stanleyho Milgrama. Experiment pozostával z posielania listov od ľudí z Nebrasky ľuďom do Bostonu. Listy sa mali posielať iba z ruky do ruky cez známych. Keď listy dosiahli svoju cieľovú destináciu, prešli priemerne cez šesť ľudí. Tento experiment viedol k zavedeniu idey o šiestich stupňoch odlúčenia. Táto myšlienka tvrdí, že pre každého človeka je každý iný človek vzdialený približne šesť sociálnych kontaktov [49].

Okrem toho, analýza sociálnych sietí sa zaoberá prieskumom človeka o jeho priateľoch, zverejňovaním dát s ich dovolením, ukladaním a sledovaním. Aplikácie, ktoré využívajú koncept tvorenia spojov medzi priateľmi a priateľmi priateľov ako Facebook a LinkedIn, tieto dáta poskytujú. Sociálne aplikácie využívajú grafy na reprezentáciu pracovných skupín, politických skupin, prípadne športových tímov alebo radikálnych skupín. Pokiaľ je sociálna sieť reprezentovaná grafom, ktorého vrcholy predstavujú ľudí a hrany medziľudské vzťahy, klika je potom skupina ľudí, kde sa každý pozná s každým.

14 Záver

Diplomová práca sa zaoberá vrcholovým farbením grafu, klikou v grafe, ich algoritmami a aplikáciami. Vrcholové farbenie grafu je zobrazenie, ktoré prideľuje jednotlivým vrcholom farby, obvykle reprezentované číslami. Úlohou je nájsť také farbenie grafu, ktoré každým dvom susedným vrcholom prideľuje rôzne farby. Optimálne farbenie grafu je farbenie s minimálnym počtom farieb, obvykle značeným $\chi(G)$. Klika v grafe predstavuje úplný podgraf. Úlohy farbenia grafu a kliky v grafe sú úzko spojené, v istom slovazmysle komplementné.

Oba problémy sú NP-úplné, takže neexistujú algoritmy, ktoré by ich vyriešili v polynomiálnom čase. Pristupuje sa k nim heuristickými alebo optimalizačnými metódami, ktoré dokážu priniesť uskutočniteľné riešenie v reálnom čase. Najzákladnejším algoritmom farbenia grafu je Greedy algoritmus, ktorý postupne priraďuje farby vrcholom, pričom ich pred spracovaním usporiada podľa nejakej heuristiky. Iný prístup je rozdeliť množinu vrcholov na disjunktné množiny, ktoré neobsahujú žiadne susedné vrcholy - na tomto princípe funguje Algoritmus farbenia pomocou nezávislých množín a Zlepovací algoritmus. Problém farbenia grafu sa dá definovať ako lineárny problém, a preto je lineárne programovanie ďalším možným prístupom. V neposlednom rade sú optimalizačné metódy, ktoré generujú náhodné riešenie a postupne sa ho snažia vylepšiť pomocou fitness funkcie.

Najznámejší algoritmus na výpočet maximálnej kliky v grafe je algoritmus Bron-Kerbosh. Prešiel mnohými úpravami a vylepšeniami, avšak využíva rekurziu, a preto je časovo náročnejší pre veľké grafy. Problém maximálnej kliky v grafe sa dá tiež definovať ako lineárny problém a riešiť pomocou maximalizácie objektovej funkcie.

Vrcholové farbenie grafu vzniklo, keď sa prvýkrát ľudia snažili farbiť geografické mapy tak, aby každé dva susedné regióny mali rôzne farby. Dnes sa táto problematika využíva všeobecnejšie a má rôzne aplikácie v problémoch, ktoré majú cieľ vyhnúť sa kolíziam. Používa sa pre rôzne typy úloh plánovania, v riadení premávky, pri riešení latinských štvorcov, pri priraďovaní frekvencií rádiovým staniciam a ďalších problémoch. Klika v grafe má tiež všeobecnejšie použitie ako len podgraf, napríklad sa používa pri zhlukovaní dát, pri spracovaní obrazu, v sociálnych sieťach a v lekárstve.

LITERATÚRA

[1] Paoletti, T. Leonard Euler's Solution to the Konigsberg Bridge Problem - Konigsberg. In:. Dostupné z: https://www.maa.org/press/periodicals/convergence/leonard-eulers-solution-to-the-konigsberg-bridge-problem-konigsberg.

[2] Sr, I., Halskau, Ø. a Laporte, G. The Bridges of Kőnigsberg - A Historical Perspective. *Networks*. 2007, zv. 49, s. 199–203. DOI: 10.1002/net.20159.

[3] Brun, Y. The four-color theorem. *Undergraduate Journal of Mathematics*. 2002, s. 21–28.

[4] Pokorný, M. *Diskrétna matematika* [online]. 2013 [cit. 2023-10-04]. Dostupné z: https://pdfweb.truni.sk/e-ucebnice/diskretna-matematika/.

[5] Hliněný, P. *Základy teorie grafů [online]*. 1. vyd. Masarykova univerzita, 2010 [cit. 2023-05-22]. Elportál. Dostupné z: http://is.muni.cz/elportal/?id=878389.

[6] Formanowicz, P. a Tanaś, K. A survey of graph coloring - Its types, methods and applications. *Foundations of Computing and Decision Sciences*. September 2012, zv. 37. DOI: 10.2478/v10209-011-0012-y.

[7] Polínková, E. *Aplikace barvení grafů s využitím omezujících podmínek*. Brno, 2013. Diplomová práca. Mendelova univerzita v Brně.

[8] Bondy, J. A. a Murty, U. S. R. *Graph Theory with Applications*. Piate. Elsevier Science Publishing Co . • Inc., 1982. ISBN 0-444-19451-7.

[9] Lewis, R. *A Guide to Graph Coloring*. Springer Publishing Company, 2016. ISBN 978-3-319-25728-0.

[10] Demel, J. *Grafy a jejich aplikace*. 2019. Dostupné z: https://kix.fsv.cvut.cz/~demel/grafy/gr.pdf.

[11] Černý, J. Základní grafové algoritmy. In:. Dostupné z: https://docplayer.cz/4802558-Zakladni-grafove-algoritmy.html.

[12] Neapolitan, R. E. *Foundations of algorithms*. Jones & Bartlett Learning, 2015. ISBN 978-1-284-04919-0. Dostupné z: https://dl.ebooksworld.ir/motoman/JBL.Foundations.Of.Algorithms.5th.Edition.www.EBooksWorld.ir.pdf.

[13] Garey, M. R. a Johnson, D. S. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman Co., 1990. ISBN 0716710455.

[14] *Graph and its representations* [online]. [cit. 2023-15-05]. Dostupné z: https://www.geeksforgeeks.org/graph-and-its-representations/.

[15] Hasenplaugh, W., Kaler, T., Schardl, T. B. a Leiserson, C. E. Ordering heuristics for parallel graph coloring. s. 166–177. DOI: https://doi.org/10.1145/2612669.2612697. Dostupné z: http://supertech.csail.mit.edu/papers/HasenplaughKaSc14.pdf.

[16] Brelaz, D. New Methods to Color the Vertices of a Graph. 1979. DOI: https://doi.org/10.1145/359094.359101.

[17] Leighton, F. T. A Graph Coloring Algorithm for Large Scheduling Problems. *JOURNAL OF RESEARCH of the Notional Bureau of Standards*. 1979.

[18] Klotz, W. Graph coloring algorithms. *Mathematik-Bericht*. Január 2002, zv. 5, s. 1–9.

[19] Robson, J. Algorithms for maximum independent sets. *Journal of Algorithms*. 1986, zv. 7, č. 3, s. 425–440. DOI: https://doi.org/10.1016/0196-6774(86)90032-5. ISSN 0196-6774. Dostupné z: https://www.sciencedirect.com/science/article/pii/0196677486900325.

[20] Dutton, R. D. a Brigham, R. C. A new graph colouring algorithm. *The Computer Journal*. Január 1981, zv. 24, č. 1, s. 85–86. DOI: 10.1093/comjnl/24.1.85. ISSN 0010-4620. Dostupné z: https://doi.org/10.1093/comjnl/24.1.85.

[21] Hertz, A. a Werra, D. Werra, D.: Using Tabu Search Techniques for Graph Coloring. Computing 39, 345-351. *Computing*. December 1987, zv. 39. DOI: 10.1007/BF02239976.

[22] Kumar, A. *Genetic Algorithms* [online]. [cit. 2023-10-04]. Dostupné z: https://www.geeksforgeeks.org/genetic-algorithms/.

[23] Katoch, S., Chauhan, S. S. a Kumar, V. A Review on Genetic Algorithm: Past, Present, and Future. *Multimedia Tools and Applications*. Február 2021, zv. 80. DOI: 10.1007/s11042-020-10139-6.

[24] Goenka, H. *Genetic Algorithms for Graph Colouring* [online]. [cit. 2023-05-15]. Dostupné z: https://www.geeksforgeeks.org/project-idea-genetic-algorithms-for-graph-colouring/.

[25] Hindi, M. a Yampolskiy, R. Genetic Algorithm Applied to the Graph Coloring Problem. *Midwest Artificial Intelligence and Cognitive Science Conference*. Január 2012, s. 60.

[26] Jabrayilov, A. a Mutzel, P. *New Integer Linear Programming Models for the Vertex Coloring Problem*. 2017.

[27] Šeda, M. Integer Programming Approach to Graph Colouring Problem and Its Implementation in GAMS. *WSEAS Transactions on Systems*. 2023. accepted for publication.

[28] *Graph Coloring Algorithm using Backtracking* [online]. [cit. 2023-05-15]. Dostupné z: `https://pencilprogrammer.com/algorithms/graph-coloring-problem/`.

[29] *Bron–Kerbosch algorithm*. San Francisco (CA): Wikimedia Foundation, 2001. Dostupné z: `https://en.wikipedia.org/wiki/Bron%E2%80%93Kerbosch_algorithm`.

[30] Antoro, S., Sugeng, K. a Handari, B. Application of Bron-Kerbosch algorithm in graph clustering using complement matrix. In:. Júl 2017, sv. 1862, s. 030141. DOI: 10.1063/1.4991245.

[31] Eppstein, D., Löffler, M. a Strash, D. Listing All Maximal Cliques in Sparse Graphs in Near-Optimal Time. In: *Algorithms and Computation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, s. 403–414. ISBN 978-3-642-17517-6.

[32] Kardos, D., Patassy, P., Szabo, S. a Zaválnij, B. Numerical experiments with LP formulations of the maximum clique problem. *Central European Journal of Operations Research*. September 2021, zv. 30. DOI: 10.1007/s10100-021-00776-z.

[33] Croce, F. D. a Tadei, R. A multi-KP modeling for the maximum-clique problem. *European Journal of Operational Research*. 1994, zv. 73, s. 555–561.

[34] *NetworkX* [online]. [cit. 2023-05-16]. Dostupné z: `https://networkx.org/`.

[35] *Graph Coloring Instances* [online]. [cit. 2023-05-16]. Dostupné z: `https://mat.tepper.cmu.edu/COLOR/instances.html#XXDSJ`.

[36] Sarkar, A. *Finds all maximal cliques in a graph using the Bron-Kerbosch algorithm* [GitHub Gist]. 2020. Dostupné z: `https://gist.github.com/abhin4v/8304062`.

[37] *DIMACS Graphs: Benchmark Instances and Best Upper Bounds* [online]. [cit. 2023-05-16]. Dostupné z: https://cedric.cnam.fr/~porumbed/graphs/.

[38] Ahmed, S. Applications of Graph Coloring in Modern Computer Science. Január 2013, zv. 3, s. 01–07.

[39] Marpaung, F. a Ritonga, A. Application of graf coloring for optimization of traffic light settings in Medan. *Journal of Physics: Conference Series*. Marec 2019, zv. 1188, s. 012012. DOI: 10.1088/1742-6596/1188/1/012012.

[40] Palúch, S. Algoritmická Teória grafov. In:. S. 238–241. Dostupné z: https://frcatel.fri.uniza.sk/users/paluch/grafy.pdf.

[41] Pardalos, P. a Xue, J. The Maximum Clique Problem. *Journal of Global Optimization*. Apríl 1994, zv. 4, s. 301–328. DOI: 10.1007/BF01098364.

[42] Ouyang, G., Dey, D. K. a Zhang, P. Clique-based Method for Social Network Clustering. In:. Dostupné z: https://arxiv.org/pdf/1708.07609.pdf.

[43] Kato, Z. *Markov Random Fields in Image Segmentation*. Dostupné z: https://inf.u-szeged.hu/~ssip/2008/presentations2/Kato_ssip2008.pdf.

[44] Ballard, D. H. a Brown, C. M. *Computer vision*. PrenticeHall , Inc., 1982. ISBN 0-13-165316-4. Dostupné z: https://homepages.inf.ed.ac.uk/rbf/BOOKS/BANDB/Ballard__D._and_Brown__C._M.__1982__Computer_Vision.pdf.

[45] Čechová, A. *Porovnávaní proteinu reprezentovaných obecným grafem BP*. Brno, 2010. Bakalárska práca. Masarykova univerzita.

[46] Liu, S., Song, Y., Zhang, F., Feng, D., Fulham, M. et al. Clique Identification and Propagation for Multimodal Brain Tumor Image Segmentation. In: *Brain Informatics and Health: International Conference, BIH 2016, Omaha, NE, USA, October 13-16, 2016 Proceedings*. Cham: Springer International Publishing, 2016, s. 285–294. ISBN 978-3-319-47103-7.

[47] Reimann, M. W., Nolte, M., Scolamiero, M., Turner, K., Perin, R. et al. Cliques of Neurons Bound into Cavities Provide a Missing Link between Structure and Function. DOI: https://doi.org/10.3389/fncom.2017.00048. Dostupné z: https://www.frontiersin.org/articles/10.3389/fncom.2017.00048/full.

[48] Li, D.-W., Wang, C. a Brüschweiler1, R. Maximal clique method for the automated analysis of NMR TOCSY spectra of complex mixtures. DOI: 10.1007/s10858-017-0119-4. Dostupné z: https://europepmc.org/article/med/28573376.

[49] Cavique, L., Mendes, A. a Santos, J. Clique communities in social networks. In:. December 2012, s. 469–490. DOI: 10.1142/9789814407724_0020.

ZOZNAM SYMBOLOV A SKRATIEK

MIS	maximálna nezávislá množina
$d(G)$	hustota grafu G
$N(v)$	množina susedov vrcholu v
$\Delta(G)$	najvyšší stupeň v grafe G
$\delta(G)$	hustota grafu G
$d(G)$	najnižší stupeň v grafe G
$\omega(G)$	klikovosť grafu G
$\alpha(G)$	počet prvkov maximálnej nezávislej množiny grafu G
$\chi(G)$	chromatické číslo grafu G
$\mathbb{N}$	množina prirodzených čísel
G'	hranový doplnok grafu G
$\mathcal{O}$	notácia “Veľké O”

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ZOZNAM PRÍLOH

A Prvá príloha

1: Zdrojové kódy: Codes_200982.zip

- obsahuje implementácie algoritmov