Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

Optimální algoritmy pro problém pokrytí bipartitního grafu biklikami

2021

Bc. Adam Beneš

Vedoucí práce: Mgr. Petr Osička, Ph.D.

Studijní obor: Aplikovaná Informatika, prezenční forma

Bibliografické údaje

Autor:	Bc. Adam Beneš
Název práce:	Optimální algoritmy pro problém pokrytí bipartitního grafu biklikami
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2021
Studijní obor:	Aplikovaná Informatika, prezenční forma
Vedoucí práce:	Mgr. Petr Osička, Ph.D.
Počet stran:	62
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Bc. Adam Beneš
Title:	Optimal algorithms for biclique cover problem
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2021
Study field:	Applied Computer Science, full-time form
Supervisor:	Mgr. Petr Osička, Ph.D.
Page count:	62
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Diplomová práce pojednává o algoritmech pro nalezení optimálního řešení problému pokrytí bipartitního grafu biklikami. Práce se zaměřuje zejména na algoritmus publikovaný v článku Ene, Alina et al. „Fast exact and heuristic methods for role minimization problems“. Součástí diplomové práce je implementace algoritmu v programovacím jazyce C.

Synopsis

This master's thesis deals with algorithms for finding an optimal solution for biclique cover problem. Thesis focuses on algorithm from article Ene, Alina et al. "Fast exact and heuristic methods for role minimization problems". The work also includes implementation of algorithm in the C programming language.

Klíčová slova: bikliky; bipartitní graf; pokrytí biklikami

Keywords: bicliques; bipartite graph; biclique cover

Rád bych poděkoval Mgr. Petru Osičkovi, Ph.D., za užitečné rady a připomínky při vedení této diplomové práce. Dále bych chtěl poděkovat rodině za podporu a trpělivost.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

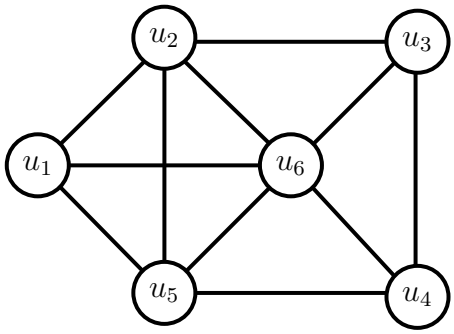
1 Teorie

V této kapitole jsou definovány pojmy, které budou potřeba v následujících kapitolách.

1.1 Graf (neorientovaný)

Graf je dvojice $G = (V, E)$, kde V je množina vrcholů (uzlů) a E je množina hran, pro kterou platí $E \subseteq V^2$. Každá hrana se tedy skládá ze dvou uzlů a jelikož zde nezáleží na orientaci, reprezentujeme hrany jako neuspořádanou dvojici (dvouprvkovou množinu). Množiny uzlů a hran jsou disjunktní ($V \cap E = \emptyset$).

Na obrázku 1 můžeme vidět příklad neorientovaného grafu. Množina V obsahuje celkem 6 uzlů ($\{u_1, \ldots, u_6\}$) a množina E obsahuje 11 hran.



Obrázek 1: Graf (neorientovaný)

1.1.1 Doplňkový graf

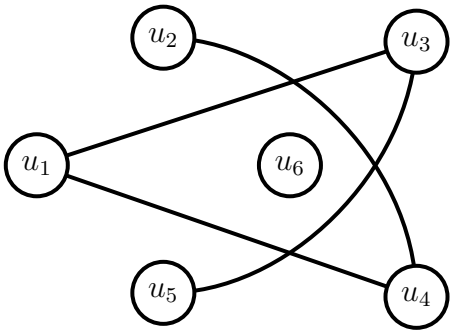
Doplňkový graf ke grafu $G = (V, E)$ je $G' = (V, V^2 \setminus E)$. To znamená že doplňkový graf G' obsahuje stejné uzly jako G a obsahuje jen ty hrany, které v grafu G chybí.

Na obrázku 2 můžeme vidět doplňkový graf ke grafu z obrázku 1. Zde si můžeme ověřit správný počet hran. Úplný graf s 6 uzly má $\binom{6}{2} = 15$ hran. Graf na obrázku 1 má 11 hran, tudíž doplňkový graf na obrázku 2 má $15 - 11 = 4$ hrany.

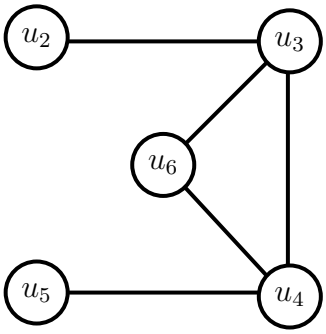
1.1.2 Podgraf

Podgraf grafu $G = (V, E)$ je graf $G' = (V', E')$, kde $V' \subseteq V$, $E' \subseteq E$ a všechny hrany v E' musí mít oba koncové uzly v množině uzlů V'. Podgraf tedy vznikne z původního grafu vymazáním některých (i žádných) uzlů a hran. Pokud se vymazaly pouze hrany, které měly alespoň jeden koncový uzel ve $V \setminus V'$, mluvíme o *indukovaném podgrafu*. [18]

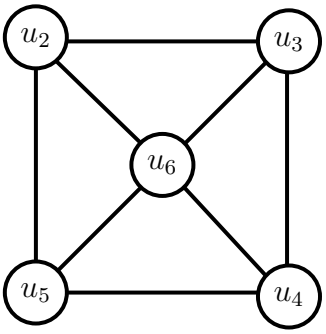
Na obrázku 3 můžeme vidět dva příklady podgrafů grafu z obrázku 1. Vlevo graf vznikl vynecháním uzlu u_1 a šesti hran. Vpravo graf vznikl pouze vynecháním uzlu u_1 a hran, které z tohoto uzlu vedly, podgraf je proto indukovaný.



Obrázek 2: Doplňkový graf



(a) Podgraf



(b) Indukovaný podgraf

Obrázek 3: Podgrafy

1.1.3 Klika

Klika v grafu $G = (V, E)$ je podgraf $G' = (V', E')$ grafu G, který je úplným grafem, což znamená, že pro každou dvojici uzlů z V' existuje hrana v E', která je spojuje. [2]

Na obrázku 4 můžeme vidět příklady klik. Vlevo je klika tvořena třemi uzly (u_1, u_2, u_5). Klika není maximální, protože přidáním uzlu u_6 vznikne větší klika, která je znázorněna na obrázku vpravo. Klika vpravo je maximální a skládá se ze čtyř uzlů (u_1, u_2, u_5, u_6).

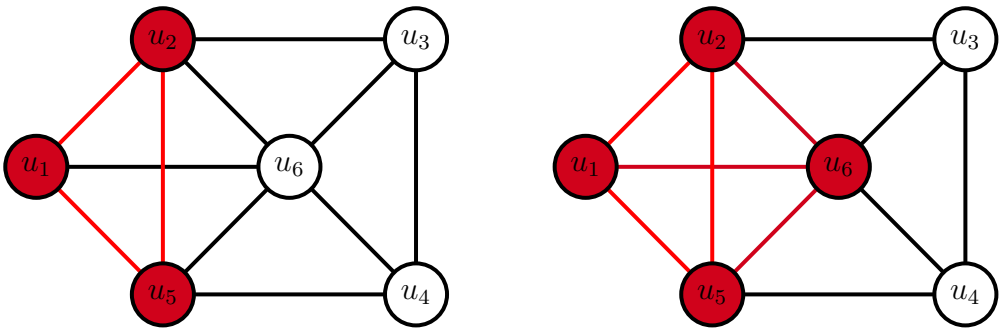
1.1.4 Reprezentace grafu v počítači

Graf můžeme reprezentovat například maticí sousednosti. Na množině V zavedeme pevné uspořádání $v_1, v_2, \ldots, v_{|V|}$. Matice sousednosti bude mít šířku a výšku rovnou počtu vrcholů. Pro jednotlivé prvky matice M_ij platí:

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

Matice sousednosti ke grafu na obrázku 1 by vypadala následovně:

$$M = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$



(a) Klika (b) Maximální klika

Obrázek 4: Kliky

1.2 Rozklad množiny

Nechť M je libovolná neprázdná množina. Pak systém M' neprázdných podmnožin množiny M, splňující podmínky:

1. libovolné dvě různé množiny ze systému M' jsou disjunktní,
2. sjednocení všech množin ze systému M' je rovno celé množině M,

se nazývá *rozklad na množině* M. Prvky systému M' se nazývají *třídy rozkladu* M' [9].

Pro představu si uvedeme příklad na množině $M = \{1, 2, 3, 4\}$, pro kterou můžeme sestavit následující rozklady:

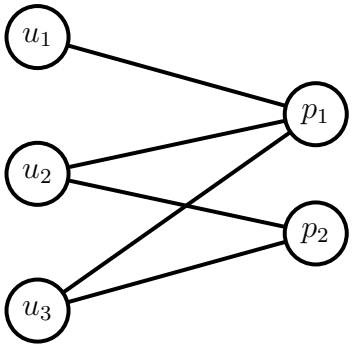
1. $M' = \{\{1, 2\}, \{3, 4\}\}$,
2. $M' = \{\{1, 4\}, \{2, 3\}\}$,
3. $M' = \{\{1\}, \{2\}, \{3, 4\}\}$,
4. $M' = \{\{1\}, \{2\}, \{3\}, \{4\}\}$,

kde v 1. a 2. případě je množina M rozdělena na 2 třídy, ve 3. případě na 3 třídy a v posledním případě na 4 třídy.

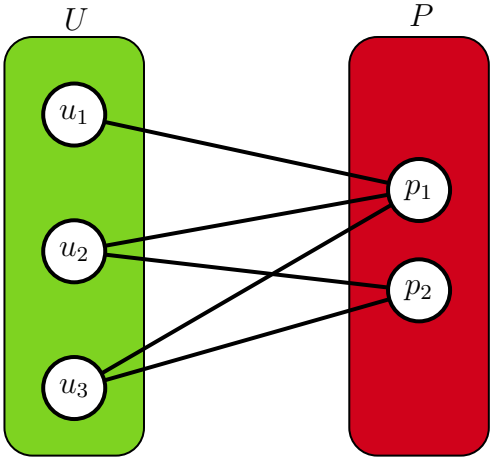
1.3 Bipartitní graf

Pro celé číslo $r \geqslant 2$ se graf $G = (V, E)$ nazývá r-partitní, pokud existuje rozklad V' množiny V na množiny vrcholů, tak že počet tříd je roven r a každá hrana má koncové uzly v jiných třídách (partitách). Uzly ve stejné třídě tedy nemohou být spojeny hranou.

Namísto 2-partitního grafu říkáme bipartitní graf. Bipartitní graf neobsahuje žádný cyklus liché délky. Množina uzlů V je rozdělena na dvě partity (U a P). E je množina hran, což chápeme jako množinu dvojic uzlů. Pro každou hranu $(u, p) \in E$ platí, že $u \in U \wedge p \in P$.



Obrázek 5: Bipartitní graf



Obrázek 6: Bipartitní graf s vyznačenými množinami U a P

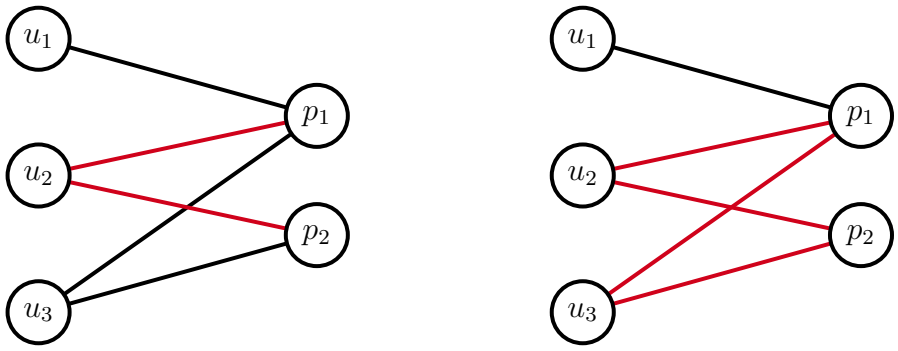
1.3.1 Příklady bipartitního grafu

Bipartitním grafem $G = (V, E)$ můžeme reprezentovat uživatele a jim přiřazená práva. Množina uživatelů je jedna parita, kterou značíme U (*Users*) a množina práv je druhá parita, kterou značíme P (*Permissions*). V grafu je vrchol uživatele $u \in U$ propojen hranou s vrcholem práva $p \in P$, pokud uživateli u je přiděleno právo p.

Dále můžeme mít například množinu zvířat a jim přiřazené vlastnosti. První partita U je tedy množina zvířat a druhá partita P je množina vlastností. V grafu existuje/neexistuje hrana (u, p), pokud zvíře u má/nemá vlastnost p.

1.3.2 Biklika

Biklika (neboli *úplný bipartitní graf*), je graf $G = (V, E)$ s partitami U a P, pro který platí, že každý vrchol z množiny U je propojen hranou s každým vrcholem z množiny P.



(a) Biklika (b) Maximální biklika

Obrázek 7: Bikliky

Velikost bikliky můžeme definovat počtem uzlů. Do *maximální bikliky* nemůžeme přidat uzly, díky kterým by vznikla větší biklika, to znamená, že maximální biklika není součástí větší bikliky.

Na obrázku 7 můžeme vidět dva příklady biklik. Vlevo se biklika skládá ze tří uzlů: u_2, p_1, p_2 a není maximální, protože se do ní dá přidat uzel u_3, takto zvětšená biklika je zvýrazněna vpravo a skládá se tedy ze čtyř uzlů: u_2, u_3, p_1, p_2 a jedná se o maximální bikliku. Uzly u_1, u_2, p_1, p_2 nemohou tvořit bikliku, protože v grafu chybí hrana z u_1 do p_2.

1.3.3 Reprezentace bipartitního grafu v počítači

Jako klasický graf Jeden ze způsobů je reprezentace jako klasického grafu například maticí sousednosti. Nejprve zvolíme pevnou posloupnost vrcholů na množině V: $v_1, v_2, \ldots, v_{|V|}$ (například tak, že naskládáme uzly z první partity následované uzly z druhé partity: $u_1, \ldots, u_{|U|}, p_1, \ldots, p_{|P|}$). Následně sestavíme matici sousednosti, ve které jsou na řádcích i ve sloupcích vrcholy z posloupnosti $v_1, v_2, \ldots, v_{|V|}$. Pokud mezi vrcholy v_i a v_j vede hrana, tak je v matici na řádku i a ve sloupci j hodnota 1. Pokud mezi vrcholy hrana nevede, je tam hodnota 0. Pro úplnost k této matici přidáme informaci o tom, jak je množina vrcholů V rozdělena mezi partity.

Pokud bychom vzali bipartitní graf z obrázku 5 a stanovili by jsme uspořádání množiny V jako u_1, u_2, u_3, p_1, p_2, pak by matice vypadala následovně:

$$M = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Dále přidáme informaci o rozdělení V na partity: $\{u_1, u_2, u_3\}, \{p_1, p_2\}$

Jako bipartitní graf Bipartitní grafy můžeme reprezentovat pomoci jedné matice M, která má počet řádků $m = |U|$ a počet sloupců $n = |P|$. Na řádcích máme uzly z množiny U a ve sloupcích uzly z množiny P. Opět zavedeme pevné uspořádání na množinách U a P. V matici M indexujeme následovně:

$$M_{ij} = \begin{cases} 1 & \text{jestliže jsou uzly } u = U_i \text{ a } p = P_j \text{ spojeny hranou,} \\ 0 & \text{jinak.} \end{cases}$$

Matice bipartitního grafu z obrázku 5 pak bude vypadat následovně:

$$M = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 1 \end{pmatrix}$$

2 Vybrané algoritmické problémy

2.1 Problém rozkladu grafu na kliky

Mějme neorientovaný graf $G = (V, E)$. Rozklad grafu G na kliky odkazuje na problém nalezení množiny klik C takové, že každý vrchol z G je obsažen přesně v jedné z klik v C.[19] V literatuře napsané v angličtině se na problém odkazuje jako na *Clique Partition*. Rozhodovací verze by vypadala následovně:

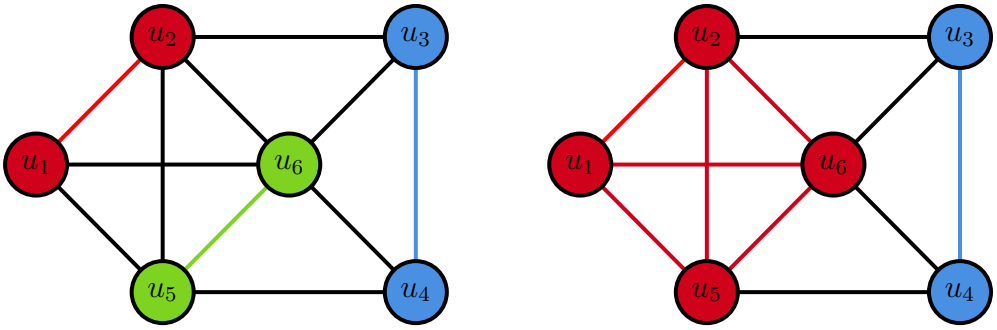
CliquePartition
Instance: Graf G a pozitivní číslo k
Otázka: Existuje množina klik C o velikosti k?

Minimalizační verze problému se jmenuje *Minimal Clique Partition* (dále jen MCP) a jde o nalezení nejmenší možné množiny klik C splňující předchozí podmínku.

MinimalCliquePartition
Instance: Graf G
Výsledek: Množina klik C s minimální kardinalitou.

2.2 Problém barvení grafu

Obarvení uzlů grafu $G = (V, E)$ je zobrazení $c : V \to C$ takové, že pokud jsou uzly v a w spojeny hranou, pak $c(v) \neq c(w)$ (mají odlišnou barvu). Prvky



(a) Clique Partition (b) Minimum Clique Partition

Obrázek 8: Problem Clique Partition

množiny C jsou dostupné barvy a je snaha naleznout nejmenší číslo k takové, že graf G je k-obarvitelný barvením $c : V \to \{1, \ldots, k\}$. Číslo k se nazývá chromatické číslo grafu G, které se značí jako $\chi(G)$. [18]

GraphColoring
Instance: Graf G
Výsledek: Barvení $c : V \to C$, tak že $|C|$ je minimální

Poznačme že k-obarvení grafu není nic jiného než rozdělení množiny V do k disjunktních podmnožin (tříd). Každý netriviální (má více jak 1 vrchol) a nenulový (má alespoň jednu hranu) bipartitní graf je tedy 2-obarvitelný.

Rozhodovací verze problému barvení grafu vypadá následovně:

GraphColoringDecision
Instance: Graf G a pozitivní číslo k
Otázka: Existuje barvení $c : V \to \{1, 2, \ldots, k\}$ pro graf G?

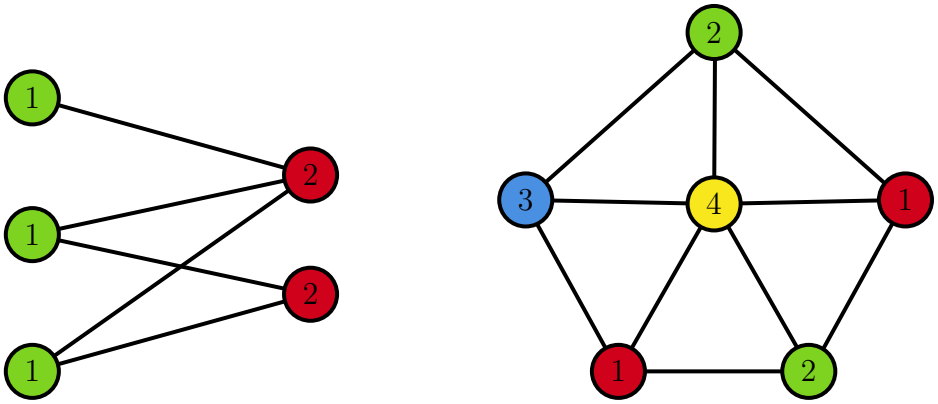
Na obrázku 9 můžeme vidět dva příklady obarvení grafu. Vlevo máme bipartitní graf, ten je obarvitelný dvěma barvami. Vpravo je graf obarvitelný čtyřmi barvami.

2.3 Problém pokrytí bipartitního grafu biklikami

Pro problém pokrytí bipartitního grafu biklikami je dán neorientovaný bipartitní graf $G = (V, E)$ a množina S, což je množina podgrafů grafu G. Množina S je pokrytí grafu G o velikosti $|S|$, pokud pro každou hranu z grafu G platí, že je obsažena alespoň v jednom podgrafu z množiny S. Pokud jsou všechny podgrafy v množině S bikliky, tak mluvíme o S jako o *pokrytí bipartitního grafu biklikami* (dále říkáme *biclique edge cover*, *biclique cover* nebo *bipartite dimension*). [4]

Rozhodovací verze pokrytí bipartitního grafu biklikami:

BicliqueCover
Instance: Graf G a pozitivní číslo k
Otázka: Má graf G pokrytí biklikami o velikosti nanejvýš k biklik?



(a) (b)

Obrázek 9: Barvení grafu

BicliqueCover je NP-úplný problém [5] , zmíněný v knize *A Guide to the Theory of NP-Completeness* [6] pod názvem GT18 (Graph theory, 18. problém). Dále je dokázáno, že je problém těžký k aproximaci (pokud platí $P \neq NP$). [11] Spodní hranice aproximovatelnosti je $|V|^{1/3-\varepsilon}$, pro všechna $\varepsilon > 0$. [21]

Minimalizační verzi problému (tedy hledáme množinu S s nejmenší kardinalitou) nazýváme *Minimal Biclique Cover*, dále jen MBC.

MinimalBicliqueCover
Instance: Graf G
Výsledek: Množina S, která pokrývá graf G a má (ze všech ostatních S' pokrývajících graf G) nejmenší kardinalitu

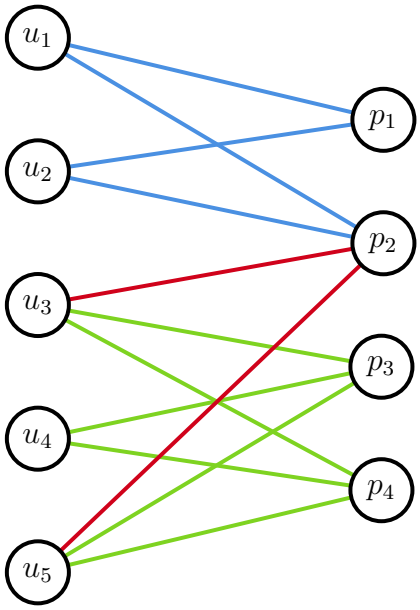
Na obrázku 10 můžeme vidět graf, který je pokrytý třemi biklikami a to konkretně biklikou obarvenou modře s uzly u_1, u_2, p_1, p_2, červeně s uzly u_3, u_5, p_2 a zeleně s uzly u_3, u_4, u_5, p_3, p_4.

2.4 Problém minimalizace rolí

Tento problém je aplikace předchozího problému, anglicky se mu říká *Role Minimization Problem*. Nejprve definujeme pojmy oprávnění uživatelů a oprávnění uživatelů založené na rolích. Následně nadefinujeme problém samotný.

Oprávnění uživatelů je ve výpočetní technice definováno jako delegování práva používat funkce související s bezpečností v počítačovém systému nebo na internetu. Oprávnění umožňuje uživateli provést akci s bezpečnostním dopadem. Jako příklady oprávnění můžeme uvést možnost vytvořit nového uživatele, možnost nainstalovat software nebo změnit funkci jádra. [17]

Oprávnění uživatelů založené na rolích (anglicky *Role-based access control*), dále jen *RBAC*. Aby uživatelé v daném informačních systému neměli přidě-



Obrázek 10: Pokrytí bipartitního grafu biklikami

lena všechna práva a tím pádem i přístup ke všem informacím, tak se každému uživateli přiřadí jen některá práva. Přiřazením práv jednotlivým uživatelům tzv. „na míru“ vzniká nebezpečí, že mu budou přiřazena omylem práva, která mu nenáleží a uživatel tak dostane neoprávněný přístup k citlivým údajům. V RBAC se nepřidělují práva uživateli přímo jedno po druhém, ale místo toho se nadefinují určité role, což jsou množiny práv, které jsou následně přiřazeny uživatelům.

Největší překážkou při zavedení RBAC je definice rolí. [1] Tomuto procesu se anglicky říká *Role Engineering* a jsou dva přístupy pro definici rolí:

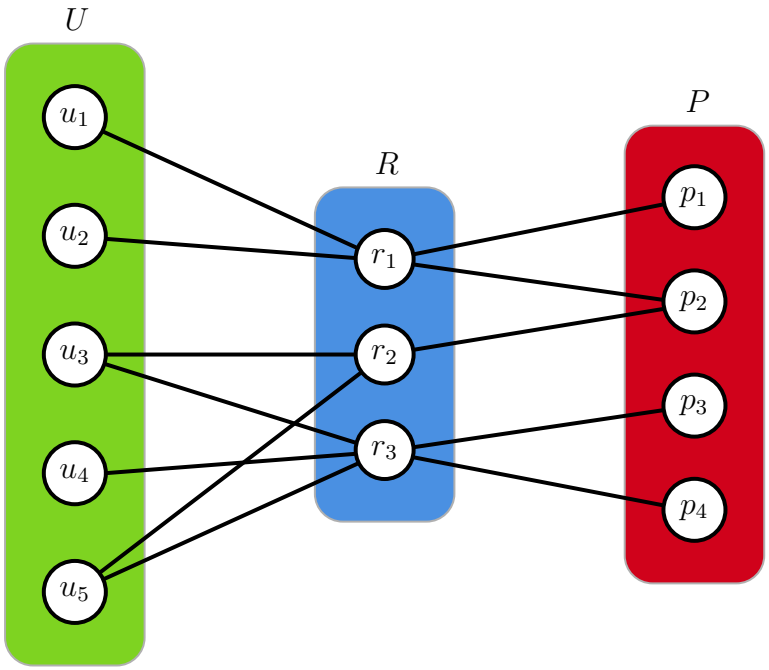
1. ***Top-down* přístup** - expert provede analýzu struktury systému a na základě toho následně nadefinuje role, které se poté přiřadí uživatelům. Tento přístup je časově náročný a tím pádem i drahý.

2. ***Bottom-up* přístup** - provede se analýza již existujících uživatelů a jim přiřazených práv, ze kterých se vyvodí role.

Oba přístupy lze zkombinovat. Výsledek *bottom-up* přístupu může být následně modifikován a vylepšen *top-down* přístupem.

Role Minimization Problem Dále jen RMP je problém, který je převeditelný na MBC a naopak, tato definice pochází z práce Vaidya, Atluri and Guo [3].

Mějme neorientovaný bipartitní graf $G = (V, E)$, s dvěma partitami U a P. Prvky množiny U jsou uživatelé (*Users*) a prvky množiny P jsou práva (*Permissions*). Tento graf reprezentuje RBAC. Cílem je najít nejmenší množinu rolí značenou R spolu s množinou hran spojujících uživatele a role značenou E_{UR} a množinou hran spojujících role a práva značenou E_{RP} tak, že platí:



Obrázek 11: Tripartitní graf G_{RB}

$$E = \{(u, p) \mid \exists r \in R, (u, r) \in E_{UR}, (r, p) \in E_{RP}\} \tag{1}$$

Množinu rolí R si můžeme představit jako (na obrázku 11 prostřední) část tripartitního grafu G_{RB}, která spojuje množiny U a P. Graf G_{RB} je definován následovně:

$$G_{RB} = (U \cup R \cup P, E_{UR} \cup E_{RP}) \tag{2}$$

Náš původní graf G má hranu (u, p) pouze pokud existuje alespoň jedna cesta délky 2 z $u \in U$ do $p \in P$ v tripartitním grafu G_{RB}. Množina C podgrafů, která vznikne tak, že projdeme všechna $r \in R$ a sestrojíme podgrafy

$$\{u \mid (u, r) \in E_{UR}\} \cup \{p \mid (r, p) \in E_{RP}\}, \tag{3}$$

je pokrytí bipartitního grafu G biklikami. Proto je RMP ekvivalentní problému MBC.

3 Algoritmus Via Reduction

Tento algoritmus, který vrací optimální řešení, je popsán v Ene et al. [1]. V tomto algoritmu se postupně převede problém pokrytí bipartitního grafu biklikami na problém minimálního rozkladu grafu klikami s použitím obarvení grafu. Tento algoritmus je v praxi velmi efektivní.

3.1 Redukce na jiný problém

V následující části je popis jak se převede problém MBC (*Minimal Biclique Cover*) na problém MCP (*Minimal Clique Partition*) a barvení grafu.

Mějme bipartitní graf G, ve kterém chceme najít minimální pokrytí bipartitního grafu biklikami. Zhotovíme hranově duální graf G' ke grafu G tak, že hrana z G se stane vrcholem v G' a všechny vrcholy v G' jsou propojeny hranou pouze pokud koncové body odpovídajících hran z G' tvoří bikliku v G. G' tedy definujeme takto:

$$G' = (E, \{(e_1, e_2) \mid e_1 \in E \text{ a } e_2 \in E \text{ tvoří bikliku v } G\}) \tag{4}$$

Mějme (maximální) kliku v G' značenou k. Vrcholy kliky k odpovídají množině hran, jejichž koncové uzly tvoří v grafu G (maximální) bikliku. Hrany pokryté (maximální) biklikou v G tvoří (maximální) kliku v G'. Každé pokrytí bipartitního grafu G biklikami tvoří množinu klik v G', značenou C. Sjednocením všech klik v C dostaneme množinu všech uzlů z G'. Abychom z C vytvořili *Clique Partition*, tak každý vrchol z G' musí být pouze v jedné z klik, proto ho necháme v jedné klice z C a z ostatních klik tento vrchol odebereme. Takhle z MBC dostaneme MCP, proces může být i opačný.

3.2 Pojmy používané v popisu algoritmu

Jak bylo vysvětleno v předchozí kapitole, daný bipartitní graf G a problém MBC se převede na jeho hranově duální graf G' a problém MCP. Nejdříve si nadefinujeme pojmy užívané v popisu:

$\mathbf{N}(v)$ **-** je množina všech sousedů vrcholu v

d **dominuje uzlu** g **-** uzel d dominuje uzlu g pokud platí:

$$(\{g\} \cup N(g)) \subseteq (\{d\} \cup N(d)), \tag{5}$$

tedy oba uzly jsou spojeny hranou a uzel d má všechny sousedy, které má uzel g. Může mít i ty sousedy, se kterými uzel g nesousedí. Dále říkáme, že uzel d je dominátor (uzlu g).

d **je izolovaný -** uzel d nemá žádné sousedy. Platí tedy:

$$N(d) = \emptyset. \tag{6}$$

3.3 Redukční strategie

Redukční strategie pro MCP grafu G pro vrchol d vypadá následovně:

- Pokud vrchol d nemá žádné sousedy, pak se v MCP grafu G vyskytuje klika obsahující jen vrchol d spolu s MPC podgrafu, který vznikne odebráním vrcholu d z G.

- Pokud vrchol d dominuje jinému vrcholu g, pak se MCP grafu G skládá z MCP podgrafu, který vznikne odebráním vrcholu d z G, modifikovaným o přidání vrcholu d do unikátní kliky, která obsahuje vrchol g.

MCP grafu G tedy vznikne rekurzivně: najde se izolovaný vrchol v a přidá se jako nová klika (s jedním vrcholem v) do MCP (nalezeném pomocí rekurze) grafu $G[V \setminus \{v\}]$, pokud se žádný izolovaný vrchol nenajde, tak se najde vrchol d dominující vrcholu g, zhotoví se (rekurzivně) MCP grafu $G[V \setminus \{d\}]$, a vrchol d se přidá do (unikátní) kliky, která obsahuje vrchol g

3.4 Popis algoritmu

Postup vypadá následovně:

1. Z bipartitního grafu G sestroj hranově duální graf G'
2. Najdi MCP grafu G':
   (a) Odstraň uzly a jejich incidentní hrany z grafu G' podle redukční strategie popsané výše (3.3), dokud nelze odstranit další uzel.
   (b) Sestroj komplementární graf grafu redukovaného G', nazvaný I (neredukovatelné jádro, anglicky *irreducible kernel*)
   (c) Obarvi graf I
   (d) Všechny uzly se stejnou barvou tvoří kliku v redukovaném grafu G'. Tyhle kliky tvoří množinu C, což je *clique partition* redukovaného grafu G'.
   (e) Vlož odebrané uzly do odpovídajících klik (vysvětleno v rekurzivní strategii) z C v opačném pořadí v jakém byly odebrány, tím se udělá MCP grafu G'
3. Každá klika v MCP grafu G' je množina hran z grafu G, která tvoří bikliku v grafu G

Časová náročnost redukce je $O(|E|^3|V|\log|V|)$. Je tu nanejvýš $|E|$ iterativních kroků a v každém kroku je v nejhorším případě průchod přes všechny dvojice vrcholů pro nalezení dominátorů. Rekurzivní strategie se v experimentech ukázala velice mocnou, pouhou redukcí se vyřešilo 7 z 9 grafů z datasetu v tabulce 2.

Nejprve se tedy spustí algoritmus 3, ve kterém se sestrojí hranově duální graf G' (krok 1.), následně se zavolá rekurzivní algoritmus 2 provádějící redukci a barvení (krok 2.) a poté se z MCP grafu G' vytvoří bikliky tvořící MBC grafu G (krok 3.). V rekurzivním algoritmu 2 z grafu G' odebírají dominující a izolované podle redukční strategie, následně se zhotoví neredukovatelné jádro, které se obarví. Každá barevná třída (množina uzlů obarvených stejnou barvou) tvoří samostatnou kliku v MCP. Za povšimnutí stojí fakt, že při odebrání izolovaného

Algoritmus 1: GetEdgeDualGraph(bip. graf $G = ((U, P), E)$)

```
G' ← prázdný unipartitní graf s |E| uzly, každý uzel je možno indexovat
     hranou z G
for hrana h1 = (u, v) ∈ E do
    for hrana h2 = (x, y) ∈ E do
        if areNeighbours(G, u, v, x, y) then
            spoj hranou uzly grafu G' reprezentující hrany h1 a h2 (z
             grafu G)
        end
    end
end
return G'
```

Algoritmus 2: MinimalCliquePartitionRec(unipartitní graf G', množina klik MCP)

```
for uzel d = (u, v) ∈ G' do
    for uzel g = (x, y) ∈ G' \ {d} do
        if uzel g dominuje uzlu g then
            MinimalCliquePartitionRec(G' \ {d}, MCP)
            do kliky z MCP obsahující uzel g přidej uzel d
            return MCP
        end
    end
    if uzel d nemá žádné sousedy then
        /* Přidej do MCP kliku obsahující uzel d      */
        MPC ← MCP ∪ {{d}}
        MinimalCliquePartitionRec(G' \ {d}, MCP)
        return MCP
    end
end
/* Nejde provést žádná další redukce                  */
I ← komplementární graf ke grafu G'
c ← GraphColorDSATUR(I)
for K ← množina uzlů se stejnou barvou z barvení c do
    MPC ← MCP ∪ {K}
end
return MCP
```

Algoritmus 3: MinimalBicliqueCoverViaReduction(bip. graf G)

```
G' ← GetEdgeDualGraph (G)
MCP ← ∅
MinimalCliquePartitionRec (G', MCP)
B ← ∅
for klika C ∈ MCP do
    b ← biklika grafu G zkonstruovaná z kliky grafu G'
    B ← B ∪ b
end
return B
```

uzlu z G' se do MCP vloží klika obsahující tento izolovaný uzel a při odebrání dominujícího uzlu g, který dominuje uzlu d, se po dokončení rekurze vloží dominující uzel g do té kliky, ve které se vyskytuje dominovaný uzel d.

3.5 Ukázkový příklad

Algoritmus `MinimalBicliqueCoverViaReduction` není na první pohled jednoduché pochopit. Doufám, že by v tom mohla pomoct názorná ukázka. Mějme bipartitní graf G, jehož znázornění můžeme vidět na obrázku 12(a), který se skládá z 8 uzlů. Množina V je rozdělena na partity $U = \{u_0, \ldots, u_3\}$ a $P = \{p_0 \ldots, p_3\}$.

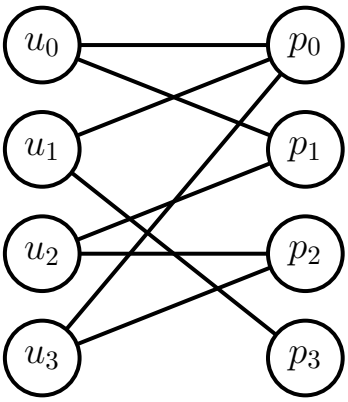
Nejprve převedeme bipartitní graf G na hranově duální graf G', který můžeme vidět na obrázku 12(b). Graf G' obsahuje jako uzly hrany z grafu G. Popisky uzlů grafu G' reprezentují hrany a jsou ve tvaru $[u_i, p_j]$, kde na první pozici je vždy uzel z partity U a na druhé pozici uzel z partity P. Pro zjednodušení jsou vynechány písmena u a p, tedy $[u_i, p_j]$ je zaznačeno jako $[i, j]$. Pro názornou ukázku takového převodu si všimněme, že hrana mezi uzly u_0 a p_0 tvoří s hranou mezi uzly u_0 a p_1 bikliku, proto je v grafu G' uzel $[0, 0]$ spojen hranou s uzlem $[0, 1]$.

Nyní budeme provádět kroky z redukční strategie popsané výše. Nejprve tedy procházíme dvojice uzlů a hledáme tzv. „dominátory“. Z obrázku 12(b) zjistíme, že uzel $[1, 0]$ dominuje uzlu $[1, 3]$. Odebereme proto uzel $[1, 0]$ z grafu G', jak je vidět na obrázku 12(c).

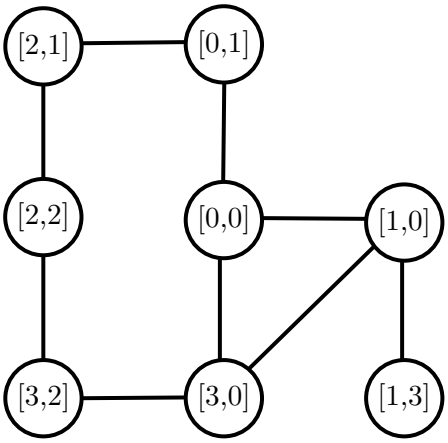
V redukovaném grafu G' se již nenachází žádný uzel, který by jinému uzlu dominoval. Nachází se zde ovšem izolovaný uzel $[1, 3]$, který podle redukční strategie také odstraníme, jak můžeme vidět na obrázku 12(d).

Pro lepší názornost si graf z obrázku 12(d) roztáhneme (obrázek 12(e)), protože dalším krokem je z redukovaného grafu udělat komplementární graf, dále pojmenovaný jako *irreducible kernel*, který můžeme vidět na obrázku 13(a).

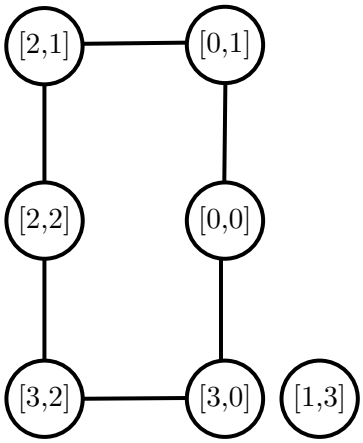
Dalším krokem je obarvení grafu z obrázku 13(a). Takto obarvený graf můžeme vidět na obrázku 13(b). Důležité je, že žádné dva sousedící uzly nemají stejnou barvu.



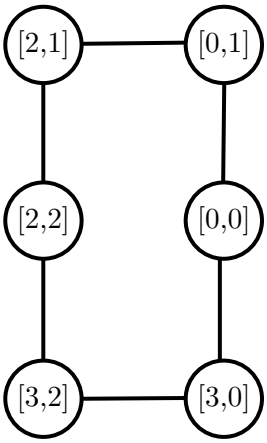
(a) Originální bipartitní graf G



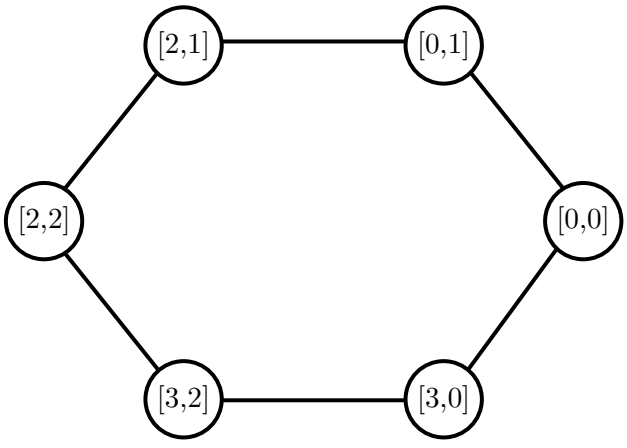
(b) Převod na hranově duální graf G'



(c) Dominátor

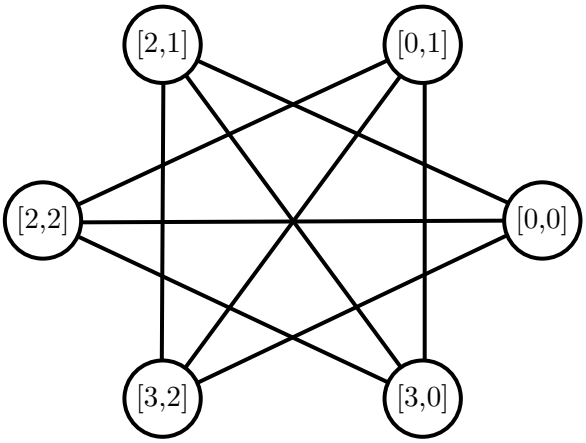


(d) Izolovaný uzel

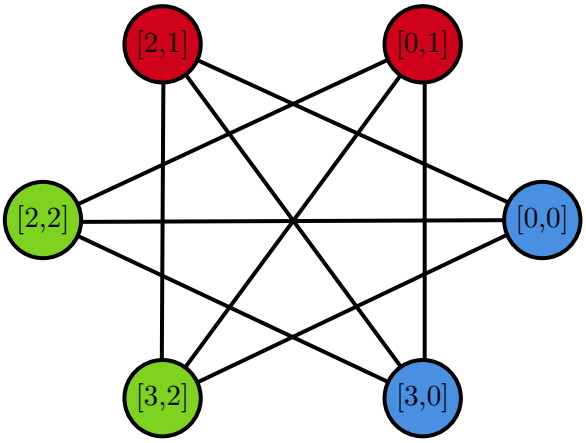


(e) Redukovaný graf

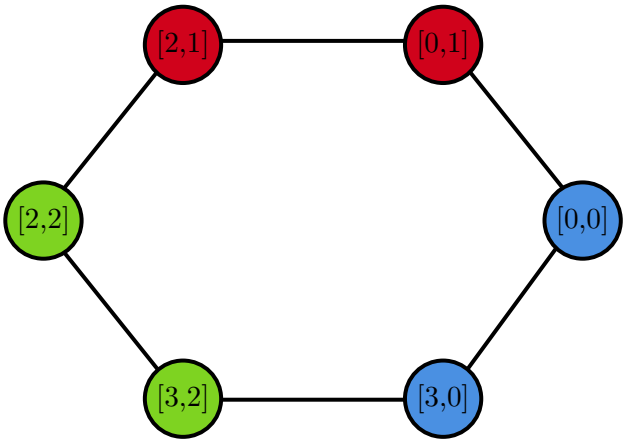
Obrázek 12: Via Reduction ukázkový příklad (1. část)



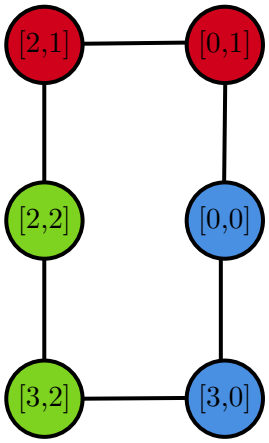
(a) Komplement redukovaného grafu



(b) Obarvení komplementu

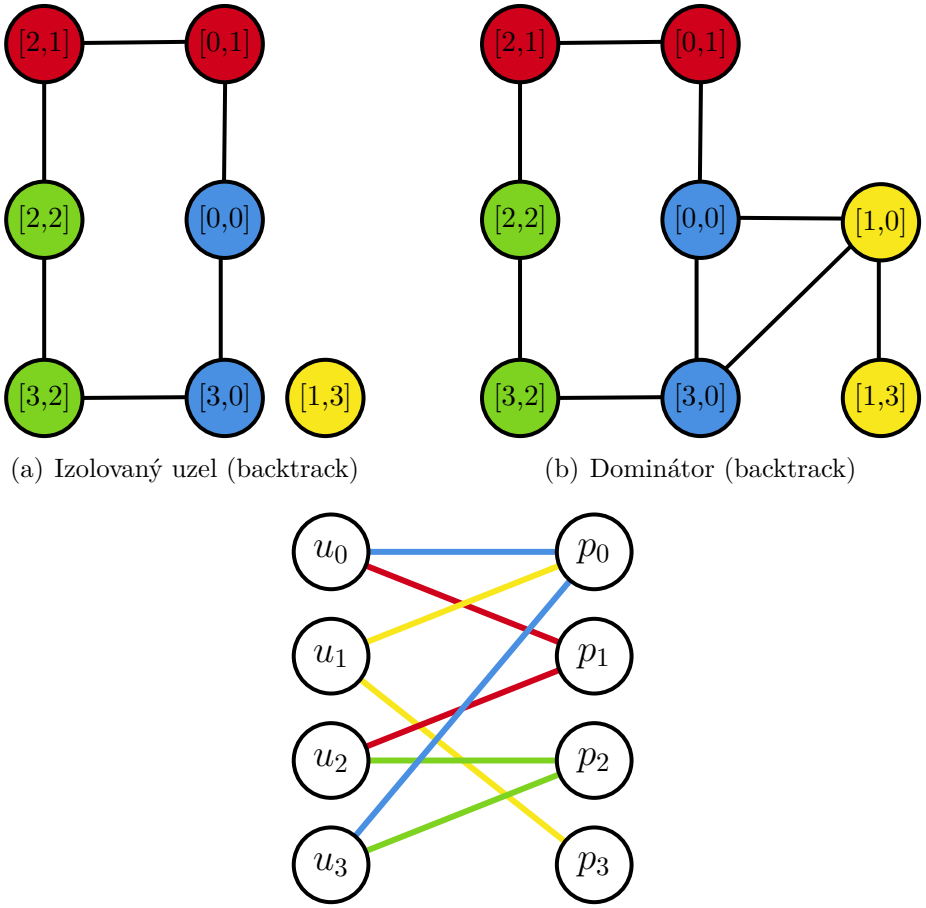


(c) Přenesení obarvení na reduk. graf



(d) Přenesení obarvení na reduk. graf (zúžený)

Obrázek 13: Via Reduction ukázkový příklad (2. část)



(a) Izolovaný uzel (backtrack)

(b) Dominátor (backtrack)

(c) Převod zpět na bip. graf G

Obrázek 14: Via Reduction ukázkový příklad (3. část)

Následně se barvy uzlů propagují zpět od *irreducible kernel* do redukovaného grafu na obrázku 13(c), který se dále zúží do kompaktního tvaru na obrázku 13(d). Zde vidíme, že obarvení uzlů tvoří *clique partition* redukovaného grafu.

Dále obnovíme redukovaný graf v opačném pořadí, v jakém byly uzly odebrány, tedy jako první vložíme do grafu uzel $[1, 3]$, který bude mít svou novou barvu (žlutou), což je znázorněno na obrázku 14(a). Následně do grafu vložíme zpět uzel $[1, 0]$, který bude mít stejnou barvu jako uzel $[1, 3]$, protože mu dominoval. Jsme ve fázi, kdy máme *clique partition* (celého) hranově duálního grafu G', což můžeme vidět na obrázku 14(b). Nyní uděláme zpětnou konverzi z hranově duálního grafu G' na bipartitní graf G se zachováním obarvení, tedy hrany z grafu G budou mít stejné barvy jako jim odpovídající uzly z grafu G'.

Na obrázku 14(c) můžeme vidět výsledný graf G s obarvenými hranami. Každá barva odpovídá jedné biklice, celkem tedy máme 4 bikliky. Jedna z nich je například označena červenou barvou skládající se ze 3 uzlů (u_0, u_2, p_2). Vidíme, že každá hrana grafu G je pokryta právě jednou biklikou.

3.6 Implementační detaily

3.6.1 Hranově duální graf G' grafu G

Pokud graf G' naimplementujeme podle definice jako graf co má $|E|$ vrcholů, dostaneme se do situace, kdy nám na větších vstupech lehce dojde paměť. Například v datasetu *americas-large* je graf, který má něco kolem 185 tisíc hran, což by mohlo představovat matici, která by měla něco kolem 34.2 miliard prvků ($(185000)^2$ prvků). Jen pro představu pokud by jeden prvek byl reprezentován jako 2B číslo, velikost matice v paměti by byla něco kolem 64GB. Je tedy potřeba interpretovat graf G' z reprezentace původního grafu G. Máme tedy dva vrcholy z grafu G', vrchol $v_1 = (u, v)$ a vrchol $v_2 = (x, y)$, což jsou hrany z grafu G, tvořeny koncovými vrcholy z grafu G. Graf G je uchovaný pomocí bipartitní matice popsané na straně 13. Matice obsahuje:

$$M_{i,j} = \begin{cases} 0 & \text{Pokud vrcholy i a j nejsou spojeny hranou} \\ 1 & \text{Pokud vrcholy i a j jsou spojeny hranou} \\ \geqslant 2 & \text{Pokud existovala hrana mezi vrcholy i a j, ale byla vymazána.} \end{cases}$$

Vrcholy $v_1 = (u, v)$ a $v_2 = (x, y)$ jsou sousední pokud platí podmínka *areNeighbours* složená ze tří částí, které musí všechny platit:

1. $M_{u,v} = 1 \wedge M_{x,y} = 1$
2. $M_{u,y} \geq 1$
3. $M_{x,v} \geq 1$

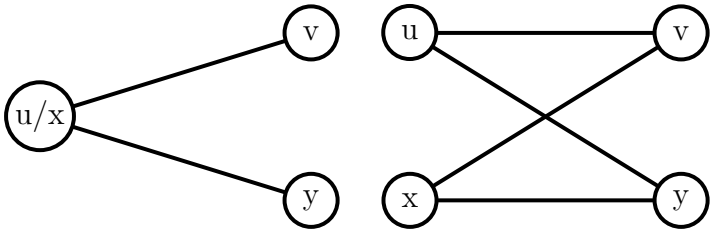
Algoritmus 4: areNeighbours(bip. graf $G = (V, E)$, hrana $h1 = (u, v)$, hrana $h2 = (x, y)$)

return
$(G[u][v] = 1\ AND\ G[x][y] = 1)$
$AND\ (G[u][y] \geqslant 1)$
$AND\ (G[x][v] \geqslant 1)$

Tato složená podmínka zachycuje fakt, že aby byly v grafu G' různé vrcholy sousední, tak musí hrany (které reprezentují v grafu G) tvořit bikliku (o 3 nebo 4 uzlech).

Na obrázcích 15 jde vidět, jak vypadají hrany, pro které podmínka platí a na obrázcích 16 je vidět příklady hran, pro které složená podmínka nebude platit. Zároveň je pod obrázky napsáno, které podmínky platily a které ne. Pseudokód 4 pro tuto složenou podmínku je nastíněn na straně 25, kde každý řádek odpovídá jedné podmínce.

Hodnota ≥ 2 v matici znamená, že byla hrana odebrána z grafu. Zároveň ale tato informace může posloužit jako identifikátor, do které kliky tento uzel patří.

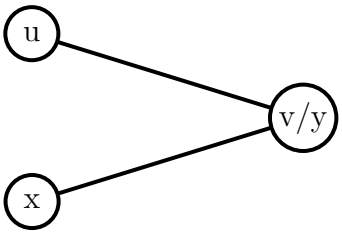


1. Ano
2. Ano
3. Ano

(a) biklika ve tvaru „<“

1. Ano
2. Ano
3. Ano

(b) biklika ve tvaru „⧖“



1. Ano
2. Ano
3. Ano

(c) biklika ve tvaru „>“

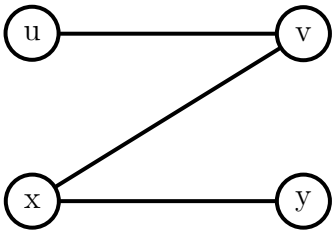
Obrázek 15: Ukázka kdy podmínka *areNeighbours* platí

Graf G' je tedy interpretován z reprezentace grafu G. Po dokončení redukce v kroku 2a na straně 19 se z grafu G opravdu zhotoví matice grafu G', ze které se následně udělá komlementární graf a poté se obarví.

3.6.2 Seznam hran

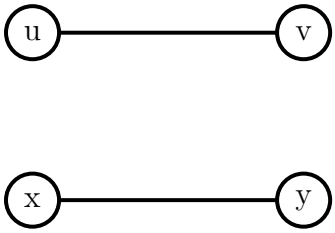
Při redukci v `MinimalCliquePartitionRec` na straně 20 dochází k opakovanému procházení hran grafu G. Abychom se vyhnuli iterování přes matici a neustálému kontrolování, jestli hrana existuje, tak si vytvoříme seznam hran. Jako strukturu jsem ve své implementaci využil cyklicky a obousměrně zřetězený seznam. Tato struktura mi umožňuje jednoduše a rychle oddělat hranu z kolekce. Prvky procházím lineárně, takže není potřeba indexace jako v poli. Pokud chci projít všechny ostatní prvky, tak začnu v jednom prvku a rekurzivně se pohybuji přes pravého potomka, dokud se nedostanu zpět k tomu původnímu prvku. Odpadá jakákoliv nutnost hlídání okrajů seznamu.

V redukci 2a na straně 19 tedy odebíráme vrcholy z G' což jsou hrany z G, které máme uchované v seznamu E, který byl popsán v předchozím odstavci. Odstranění vrcholu z grafu G' tedy probíhá tak, že se oddělá odpovídající hrana v seznamu E a v matici se na odpovídajícím místě změní hodnota z 1 („hrana existuje“) na 2 („hrana byla odebrána“).



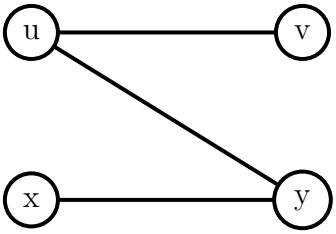
1. Ano
2. Ne
3. Ano

(a) Chybí hrana (u, y)



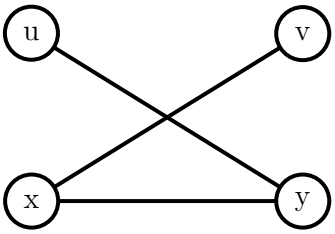
1. Ano
2. Ne
3. Ne

(b) Chybí hrany (u, y) a (x, v)



1. Ano
2. Ano
3. Ne

(c) Chybí hrana (x, v)



1. Ne
2. Ano
3. Ano

(d) Chybí hrana (u, v)

Obrázek 16: Ukázka kdy podmínka *areNeighbours* neplatí

3.6.3 Algoritmus pro barvení grafu

Aby algoritmus *Via Reduction* 3 vracel optimální řešení, je potřeba algoritmus na obarvení grafu, který také vrací optimální řešení. Algoritmus *DSATUR* je poprvé zmíněn v práci *Breláz: New methods to color the vertices of a graph.(1973)* [7]. Od té doby prošel několika změnami, v následujících odstavcích bude popsána varianta z práce Furini and Gabrel [8].

Algoritmus 5: DSATUR(unipartitní graf $G = (V, E)$)

$LB \leftarrow$ `MaxCliqueSize(`G`)`
$UB \leftarrow |V|$
$C' \leftarrow$ pole plné -1 o velikosti $|V|$
$C \leftarrow \emptyset$
$k \leftarrow 0$
`DSATURRec` (G, LB, C', C, k)
return C

Algoritmus DSATUR je pojmenován podle *Deegree of saturation* (dále jen *DSAT*), což je hodnota, kterou má každý uzel a znamená počet různých barev, kterými jsou sousedi daného uzlu obarveni.

Algoritmus pro unipartitní graf $G = (V, E)$ vrátí obarvení C. Nejprve se určí dolní a horní hranice počtu použitých barev. Dolní hranice (anglicky *lower bound*, dále jen *LB*) je velikost největší maximální kliky v grafu G, kterou vypočteme algoritmem `MaxCliqueSize`. Hodnota *LB* se v průběhu algoritmu již dále nemění. Horní hranice (anglicky *upper bound*, dále jen *UB*) je na začátku počet uzlů v grafu G, protože nejhorší případ nastane, když každý uzel obarvíme jinou barvou, máme tedy maximálně $|V|$ počet barev. Když se nalezne obarvení, které má méně barev než *UB*, tak se hodnota *UB* zmenší (*UB* bereme jako globální proměnnou). C' je pole, které reprezentuje zatím nalezené obarvení. Na začátku nemá žádný uzel přiřazenou barvu, to reprezentujeme hodnotou -1. C je pole, ve kterém se bude uchovávat nejlepší zatím nalezené obarvení. k je počet použitých barev v částečném (dosud neúplném) obarvení C', na začátku je 0. Poté se zavolá rekurzivní funkce `DSATURRec` a nakonec se vrátí hodnota nejlepšího obarvení C.

Algoritmus `DSATURRec` je rozdělen na dvě části. Pokud je vše obarveno, tak se zkontroluje, jestli je obarvení C' dosud nejlepší nalezené. Pokud ano tak ho uložíme a změníme hodnotu *UB*. Pokud obarvení C' není kompletní, tak se zvolí uzel v, který bude obarven. Následně zkusíme uzel v obarvit barvami, které již jsou v barvení C' a následně jednou novou barvou. Podmínky, ve kterých se vyskytuje `max`, udržují počet barev barvení C' mezi *LB* a *UB*. To znamená, že pokud jsme nalezli barvení, které má počet barev *LB*, tak algoritmus rychle terminuje a zároveň se neposuneme do větve, ve které bychom měli více barev než je hodnota *UB*. Volání `DSATURRec` s hodnotou $k + 1$ znamená, že jsme do barvení C' přidali novou barvu.

Algoritmus 6: DSATURRec($G = (V, E)$), LB, C', C, k)

```
if vše je obarveno then
    if k ≤ UB then
        C ← C'                    /* zkopírovat data z C' do C */
        UB ← k
    end
else
    v ← neobarvený uzel (hodnota −1 v barvení C') s největší hodnotou
      degree of saturation
    /* Pro barvy, které již jsou v barvení C'                    */
    for i ← barva z barvení C', kterou nemá žádný soused uzlu v do
        if max(k, LB) < UB then
            C'[v] ← i
            DSATURRec(G, LB, C', C, k)
        end
    end
    /* Nová barva                                                */
    if max(k+1, LB) ≤ UB then
        C'[v] ← k
        DSATURRec(G, LB, C', C, k + 1)
    end
    C'[v] ← −1
end
```

Algoritmus 7: MaxCliqueSize(graf $G = (V = \{v_1, \ldots, v_n\}, E)$)

$Max \leftarrow 0$
$Found \leftarrow false$
for $i \leftarrow n$ **downto** 1 **do**
  $Found \leftarrow false$
  $S_i \leftarrow \{v_i, v_{i+1}, \ldots, v_n\}$
  `Clique` $(S_i \cap N(v_i), 1)$
  $c_i \leftarrow Max$
end
return c_1

Algoritmus MaxClique je popsán v *Östergård: A fast algorithm for the maximum clique problem* [10]. Nejprve se nadefinují globální proměnné: pole c (na začátku plné 0), logická hodnota $Found$ a číslo Max. Mějme $S_i = \{v_i, v_{i+1}, \ldots, v_n\}$. Hodnota c_i udává velikost největší kliky v množině uzlů S_i. Hodnota c_1 nám tedy udává velikost největší kliky grafu G. V algoritmu `MaxCliqueSize` tedy začneme s $S_n = \{v_n\}$ a postupně kardinalitu množiny S_i zvětšujeme. Vycházíme z faktu že pro každé $1 \leq i \leq n-1$ platí, že $c_i = c_{i+1}$ nebo $c_i = c_{i+1} + 1$. Neboli máme $c_i = c_{i+1} + 1$ pouze pokud je v S_i klika o velikosti $c_{i+1} + 1$, která obsahuje uzel v_i.

Algoritmus `Clique` má parametry graf U a číslo $size$, což je velikost prozatím nalezené kliky. Algoritmus je rozdělen na dvě části. Pokud je počet uzlů grafu U rovno 0, porovnáme jestli jsme nalezli největší kliku a pokud ano tak uložíme velikost a nastavíme $Found$ na hodnotu $true$, tím `Clique` a jeji rekurzivní volání terminuje až do místa, kde byla funkce `Clique` poprvé volána.

V druhé části algoritmu (to znamená, že v grafu U je ještě nějaký uzel, který by se mohl přidat do skládané kliky) procházíme všechny uzly v grafu U v pořadí podle nejmenší hodnoty počtu sousedů. První podmínka ve *while* cyklu znamená, že i kdybychom přidali do kliky všechny zbývající uzly v grafu U, tak bychom nedostali větší kliku, takže můžeme funkci terminovat. V další podmínce využijeme staré hodnoty c_i. Poté odebereme uzel v_i z grafu U a zavoláme `Clique` s grafem, ve kterém jsou všichni sousedi uzlu v_i, kteří byli zároveň v grafu U a s navýšenou hodnotou velikosti kliky o 1 (protože jsme do kliky přidali uzel v_i). Po rekurzivním zavolání následuje podmínka, která umožňuje rychlou terminaci po nalezení největší kliky v dané iteraci (to znamená, že se našla klika, která obsahuje uzel $v_i \in S_i$ z původního volání funkce `Clique`).

Algoritmus 8: Clique(graf $U = (V', E'), size$)

```
if |V'| = 0 then
    if size > Max then
        Max ← size
        Found ← true
    end
else
    while |V'| > 0 do
        if size + |V'| ≤ Max then
            return
        end
        i ← index uzlu s nejméně sousedy z grafu U
        if size + c_i ≤ Max then
            return
        end
        U ← U \ {v_i}
        Clique(U ∩ N(v_i), size + 1)
        if Found = true then
            return
        end
    end
end
```

4 Backtracking algoritmus

V této kapitole je popsán návrh vlastního algoritmu na výpočet MBC. Problém je popsán pomocí formální konceptuální analýzy, kde formální kontext je analogie grafu a formální koncept je analogie bikliky.

4.1 Pojmy z formální konceptuální analýzy

Formální kontext

Mějme tabulková data reprezentována trojicí $\langle X, Y, I \rangle$, kde X je množina objektů, Y je množina atributů a I je binární relace mezi X a Y. Pokud $x \in X$ a $y \in Y$ jsou v relaci I ($\langle x, y \rangle \in I$), tak to znamená, že objekt x má daný atribut y. Tyto tabulková data se nazývají formální kontext [12] (anglicky *formal context*).

Příklad formálního kontextu můžeme vidět v tabulce 1. Tato tabulka (s vynechanými nulami na prázdných místech pro přehlednost) reprezentuje matici bipartitního grafu na obrázku 10. Máme tedy množinu objektů $X = \{u_1, u_2, u_3, u_4, u_5\}$ a množinu atributů $Y = \{p_1, p_2, p_3, p_4\}$. Binární relace I je naznačena v tabulce, takže například objekt u_3 má atributy p_2, p_3, p_4.

	p_1	p_2	p_3	p_4
u_1	1	1		
u_2	1	1		
u_3		1	1	1
u_4			1	1
u_5		1	1	1

Tabulka 1: Příklad formálního kontextu

Šipkové operátory

Mějme formální kontext $\langle X, Y, I \rangle$. Pro všechny množiny $A \subseteq X$ (množina objektů) a $B \subseteq Y$ (množina atributů), jsou nadefinovány následující šipkové operátory:

1. **Intent operátor (šipka nahoru)**
   - je operátor $^{\uparrow} : 2^X \to 2^Y$ definovaný jako:

$$A^{\uparrow} = \{y \in Y \mid \forall x \in A : \langle x, y \rangle \in I\} \tag{7}$$

   - Množině objektů A je operátorem $^{\uparrow}$ přiřazena množina atributů z Y, které má každý objekt z A.
   - Příklady z tabulky 1:

(a) $A = \{u_1\}$, $A^{\uparrow} = \{p_1, p_2\}$.
(b) $A = \{u_3, u_5\}$, $A^{\uparrow} = \{p_2, p_3, p_4\}$
(c) $A = X = \{u_1, u_2, u_3, u_4, u_5\}$, $A^{\uparrow} = \emptyset$

2. **Extent operátor (šipka dolů)**

- je operátor $^{\downarrow} : 2^Y \to 2^X$ definovaný jako:

$$B^{\downarrow} = \{x \in X \mid \forall y \in B : \langle x, y \rangle \in I\} \tag{8}$$

- Množině atributů B je operátorem $^{\downarrow}$ přiřazena množina objektů z X, které mají všechny atributy z B.
- Příklady z tabulky 1:
  (a) $B = \{p_2\}$, $B^{\downarrow} = \{u_1, u_2, u_3, u_5\}$
  (b) $B = \{p_3, p_4\}$, $B^{\downarrow} = \{u_3, u_4, u_5\}$
  (c) $B = Y = \{p_1, p_2, p_3, p_4\}$, $B^{\downarrow} = \emptyset$

Formální koncept

Mějme formální kontext $\langle X, Y, I \rangle$. Formální koncept (anglicky *formal concept*) je dvojice $\langle A, B \rangle$, kde $A \subseteq X$ je množina objektů a $B \subseteq Y$ je množina atributů a pro které platí, že $A^{\uparrow} = B$ a $B^{\downarrow} = A$. To znamená, že v A jsou jenom ty objekty, které sdílí všechny atributy z B a naopak v B jsou jenom ty atributy, které jsou sdíleny všemi objekty z A. [12]

Příklady formálního konceptu v tabulce 1:

1. $\langle A, B \rangle$, kde $A = \{u_1, u_2\}$ a $B = \{p_1, p_2\}$
2. $\langle A, B \rangle$, kde $A = \{u_3, u_5\}$ a $B = \{p_2, p_3, p_4\}$

Na formální koncept můžeme nahlížet jako na bikliku. [13] Mějme formální kontext $\langle X, Y, I \rangle$, což jsou tabulková data. Tato tabulka představuje matici sousednosti bipartitního grafu $G = (V, E)$, kde množina V jde rozdělit na podmnožiny X a Y. Množiny X a Y jsou tedy v grafu reprezentovány jako uzly. Jestliže $x \in X$, $y \in Y$ a $\langle x, y \rangle \in I$, znamená to, že v grafu G existuje hrana mezi uzly x a y ($(x, y) \in E$). Z formálního konceptu $\langle A, B \rangle$ můžeme sestavit podgraf

$$F = (A \cup B, \{(a, b) \in E : a \in A \wedge b \in B\}), \tag{9}$$

který v grafu G tvoří (maximální) bikliku.

Pokrytí formálního kontextu formálními koncepty

Jelikož v následujícím algoritmu používáme formální koncepty namísto biklik, je třeba definovat analogii k pokrytí bipartitního grafu biklikami. Mějme formální kontext $\mathcal{K} = \langle X, Y, I \rangle$ a množinu formálních konceptů K. Množina K je pokrytí formálního kontextu $\mathcal{K}$ formálními koncepty, pokud pro každou dvojici $\langle x, y \rangle \in I$ platí, že je obsažena alespoň v jednom formálním konceptu z K. Hledáním nejmenší množiny K, splňující předchozí podmínku, získáme minimální pokrytí formálního kontextu formálními koncepty.

4.2 Algoritmus na výpočet všech formálních konceptů

Algoritmus 9: FCBO($\langle A, B\rangle, y, \{N_y \mid y \in Y\}$)

vylistuj formální koncept $\langle A, B\rangle$
if $B = Y$ *OR* $y > n$ **then**
  return
end
for $j \leftarrow y$ **to** n **do**
  $M_j \leftarrow N_j$
  if $j \notin B$ *AND* $N_j \cap Y_j \subseteq B \cap Y_j$ **then**
    $\langle C, D\rangle \leftarrow$ `ComputeClosure`(B, j)
    if $B \cap Y_j = D \cap Y_j$ **then**
      zařaď $\langle\langle C, D\rangle, j\rangle$ do fronty F
    else
      $M_j \leftarrow D$
    end
  end
end
while $\langle\langle C, D\rangle, j\rangle$ *je další prvek z fronty* F **do**
  `FCBO`$(\langle C, D\rangle, j + 1, \{M_y \mid y \in Y\})$
end
return

Algoritmus pochází z práce *Jan Outrata: Computing and Applying Formal Concepts*[14] a jmenuje se `FCBO` (*Fast Close-by-One*). Algoritmus z daného formálního kontextu $\langle X, Y, I\rangle$ vylistuje všechny formální koncepty. *Fast* referuje na fakt, že se jedná o vylepšenou verzi předchozího algoritmu (*Recursive CBO*) v práci a *Close-by-One* odkazuje na algoritmus z práce *Kuznetsov: Interpretation on graphs and complexity characteristics of a search for specific patterns* [15], ze kterého tento algoritmus vychází. Algoritmus *Close-by-One* v každém rekurzivním volání vypočítá jeden formální koncept. Hlavním záměrem `FCBO` je, aby se stejný formální koncept nevypočítával (a nevylistoval) vícekrát. Aby výkon algoritmu `FCBO` nezávisel na pořadí atributů v množině Y, zavádí se vzestupné uspořádání na atributech.

Algoritmus `FCBO` 9 má vstupní parametry formální koncept $\langle A, B\rangle$, číslo y což je index atributu a množiny N_y. `FCBO` se poprvé zavolá s $\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow}\rangle$, $y = 0$ a $\{N_y = \emptyset \mid y \in Y\}$ jako argumenty. Y je množina atributů, n je počet atributů. Algoritmus nejprve vylistuje formální koncept $\langle A, B\rangle$, to znamená, že ho třeba vytiskne na obrazovku nebo přidá do nějaké množiny. Další podmínka značí, že buď B pokrývá celou množinu atributů, nebo jde index y mimo pole atributů Y, v obou případech nelze přidat do B další atribut, a proto se ukončí algoritmus. V případě, kdy neplatí předchozí podmínka, se iteruje přes j od y až po konec pole s atributy Y. Pokud atribut j není obsažen v množině atributů B a zároveň platí podmínka, která má za úkol eliminovat opakovaný výpočet formálního konceptu,

Algoritmus 10: ComputeClosure($\langle A, B\rangle, j$)

$C \leftarrow \emptyset$
$D \leftarrow Y$
foreach $x \in A \cap \{j\}^{\downarrow}$ **do**
  $C \leftarrow C \cup \{x\}$
  $D \leftarrow D \cap \{x\}^{\uparrow}$
end
return $\langle C, D\rangle$

který již byl vypočítán a vylistován, vypočte se algoritmem `ComputeClosure` nový formální koncept $\langle C, D\rangle$. Pokud neplatí následující podmínka, tak se eliminuje rekurzivní volání `FCBO` s formálním konceptem, který již byl vylistován a za M_j se dosadí D. Pokud podmínka platí, tak se do fronty F vloží vypočtený formální koncept spolu s indexem j. Po průchodu všech j se projde fronta a pro každý prvek fronty (formální koncept $\langle C, D\rangle$ a index j) se rekurzivně zavolá `FCBO`.

Algoritmus `ComputeClosure` 10 má vstupní parametry formální koncept $\langle A, B\rangle$ a atribut j. Nejprve nadefinujeme C jako prázdnou množinu a D jako množinu všech atributů Y. Poté procházíme prvky (x) množiny $A \cap \{j\}^{\downarrow}$, což je průnik objektů z A a objektů, které mají atribut j. Následně se každý prvek x z množiny objektů přidá do množiny C a do D se vloží průnik D s množinou atributů, které má objekt x. Po průchodu všech prvků x algoritmus vrátí formální koncept $\langle C, D\rangle$.

4.3 Základní kostra backtracking algoritmu

Algoritmus 11: BTCOMB1(formální kontext $\mathcal{K} = \langle X, Y, I\rangle$

$UB \leftarrow$ `min` $(|X|, |Y|) + 1$
$allC \leftarrow$ všechny formální koncepty (kromě těch obsahujících prázdnou množinu)
$bestConcepts \leftarrow \emptyset$
$C \leftarrow \emptyset$
$index \leftarrow 0$
`BTCOMB1Rec` $(\mathcal{K}, allC, C, bestConcepts, UB, index)$
return $bestConcepts$

V 1. verzi algoritmu 11 se určí hodnota horní hranice počtu konceptů a uloží se do UB (*upper bound*), dále se vypočítají všechny formální koncepty (algoritmem 9 popsaným výše) a dají se do množiny (pole) $allC$ (*All Concepts*). Vyhradí se místa pro nejlepší množinu konceptů (*bestConcepts*) a aktuálně budovanou množinu konceptů (C). Index se nastaví na začátek $allC$ tudíž na 0 a zavolá se rekurzivní algoritmus `BTCOMB1Rec`.

Algoritmus 12: BTCOMB1Rec($\mathcal{K}, allC, C, bestConcepts, UB, index$)

```
if form. kontext K je pokrytý form. koncepty z C then
    if |C| < UB then
        /* mame nejlepsi doposud nalezene pokryti    */
        bestConcepts ← C
        UB ← |C|
    end
else if index < |allC| AND |C| < UB then
    for i ← index to |allC| − 1 do
        if |C| < UB then
            C ← C ∪ allC[i]
            BTCOMB1Rec(G, allC, C, bestConcepts, UB, i + 1)
            C ← C \ allC[i]
        end
    end
end
```

Rekurzivní algoritmus `BTCOMB1Rec` postupně prochází všechny kombinace prvků z $allC$ o maximální velikosti UB. V každém průchodu se buď zkontroluje jestli může být formální kontext $\mathcal{K}$ pokryt koncepty z C a jestli je to nejlepší pokrytí, tak ho uložíme do $bestConcepts$ a aktualizujeme UB. Jestli C není pokrytí formálního kontextu $\mathcal{K}$, tak provedeme nejprve podmínku, ve které se otestuje, jestli nejdeme s indexem mimo pole $allC$ a zároveň jestli přidáním dalšího konceptu nebude velikost C větší jak UB. Poté se ve *for* cyklu prochází přes i od hodnoty $index$ po konec pole $allC$. Následující podmínka značí, že v předchozí iteraci přes i se mohlo najít nejlepší pokrytí, přidáním dalšího konceptu tedy nevznikne pokrytí lepší (ale nejlépe stejně velké). Následně se přidá koncept z $allC$ na indexu i do C, zavolá se rekurzivně `BTCOMB1Rec` a poté se konpcet zase z C odejme. `BTCOMB1Rec` bude pokračovat na následujícím indexu $(i + 1)$, proto nemůže obsahovat C stejné formální koncepty.

4.4 Mandatorní koncepty

Algoritmus můžeme vylepšit tím, že nebudeme začínat s prázdnou množinou konceptů C, ale že do množiny C vložíme koncepty, které v pokrytí určitě nesmějí chybět a to jsou právě mandatorní koncepty (*Mandatory concepts*). [16]

Pro daný formální kontext $\langle X, Y, I\rangle$ mějme množinu objektových konceptů $\mathcal{O}(X, Y, I)$ a množinu atributových konceptů $\mathcal{A}(X, Y, I)$, definované následovně:

$$\mathcal{O}(X, Y, I) = \{\langle\{x\}^{\uparrow\downarrow}, \{x\}^{\uparrow}\rangle \mid x \in X\}, \tag{10}$$

$$\mathcal{A}(X, Y, I) = \{\langle\{y\}^{\downarrow}, \{y\}^{\downarrow\uparrow}\rangle \mid y \in Y\}. \tag{11}$$

Jednotlivé množiny mají následující význam:

- $\{x\}^{\uparrow\downarrow}$ jsou objekty, které mají stejné atributy jako objekt x,
- $\{x\}^{\uparrow}$ jsou atributy, které má objekt x,
- $\{y\}^{\downarrow}$ jsou objekty mající atribut y,
- $\{y\}^{\downarrow\uparrow}$ jsou atributy, které jsou sdíleny všemi objekty, co mají atribut y.

Mandatorní koncepty $\mathcal{M}(X, Y, I)$ jsou definovány jako průnik objektových a atributových konceptů, tedy následovně:

$$\mathcal{M}(X, Y, I) = \mathcal{O}(X, Y, I) \cap \mathcal{A}(X, Y, I) \tag{12}$$

2. verze algoritmu tedy bude vypadat tak, že se v algoritmu 11 do C vloží mandatorní koncepty $\mathcal{M}(X, Y, I)$.

4.5 Třídění podle počtu nově pokrytých hran

V algoritmu 12 se prochází kandidáti z pole $allC$ na přidání do aktuálně budované množiny konceptů $bestConcepts$. Možné vylepšení algoritmu zahrnuje uložení kandidátů $(allC_{index}, \ldots, allC_{|allC|-1})$ do nového pole D. Poté pole D setřídíme sestupně podle toho, kolik by daný kandidát pokryl nových dvojic $\langle x, y\rangle \in I$. Následně procházíme setříděným polem s tím, že pokud daný kandidát nepokryje ani jednu dvojici, tak ho přeskočíme.

5 Experimentální výsledky

5.1 Testování nad známými datasety

Jelikož jsem algoritmus převzal z práce Ene et al. [1], tak jsem přirozeně vybral jako první testovací sadu datasety z této práce. Informace o jednotlivých grafech v daných souborech můžeme vidět v tabulce 2 a výsledky v tabulce 3.

Stojí za povšimnutí, že velikost jádra hraje velkou roli. Velikost jádra značí počet uzlů v grafu *irreducible kernel*, který se následně má obarvit. Z experimentů jsem zjistil, že tato část algoritmu je velmi kritická. V posledním sloupci v tabulce 6 je hodnota redukce v procentech, která značí kolik uzlů se podařilo zredukovat. V tabulce jsou vysoké hodnoty redukce, to v praxi znamenalo, že se například u grafu *americas-small* ze 105205 hran (které tvoří v neredukovatelném jádře uzly) odebralo 99.96% $(105205 * (1 - 0.99958176) = 44$ uzlů v neredukovatelném jádře).

Nabízela se otázka zkusit algoritmus s jinými datasety, které můžeme vidět v tabulce 4. Na některých grafech algoritmus ani nedoběhl, to můžeme poznat z toho, že je na daném řádku ve sloupci s počtem biklik místo číselné hodnoty

Jméno	Uživatelé	Práva	Hrany	Zaplnění(%)
americas_large	3 485	10 127	185 294	0.53
americas_small	3 477	1 587	105 205	1.91
apj	2 044	1 164	6 841	0.29
customer	10 961	277	45 427	1.50
domino	79	231	730	4.00
emea	35	3 046	7 220	6.77
firewall1	365	709	31 951	12.35
firewall2	325	590	36 428	19.00
healthcare	46	46	1 486	70.23

Tabulka 2: Tabulka datasetů z [1]

Název souboru	Uplynulý čas (sec.)	Počet biklik	Velikost jádra	Redukce (%)
americas_large	89.6956	398	97	99.95
americas_small	21.0600	178	44	99.96
apj	0.1324	453	0	100.00
customer	6.6176	276	0	100.00
domino	0.0066	20	0	100.00
emea	0.1966	34	0	100.00
firewall1	1.4154	64	0	100.00
firewall2	1.7304	10	0	100.00
healthcare	0.0102	14	0	100.00

Tabulka 3: Výsledky algoritmu Via Reduction na datasetech z [1]

otazník. V tomto případě jsem nad danými grafy spustil algoritmus pro hledání velikosti jádra a v tabulce zaznamenal čas tohoto výpočtu. Grafy *chess*, *mushroom* a *tic-tac-toe* měly příliš velkou velikost jádra. Limit jsem nastavil na jednu hodinu a po jedné hodině byl u grafu *servo* výpočet ve fázi hledání maximální kliky a u grafu *post* ve fázi hledání nejlepšího obarvení. Graf *nfs* byl ze všech grafů největší, velikost jádra se podařilo najít po cca hodině a půl.

Výsledky běhu backtracking algoritmu na datasetech z [1] jsou vidět v tabulce 6. Limit byl nastavený na 1 hodinu. Na některých grafech algoritmus po jedné hodině nedoběhl, hodnoty s hvězdičkou značí, kolik biklik to nejlépe našlo po 1 hodině. U grafu *customer* to po jedné hodině dokonce nenašlo ani jedno pokrytí. Za zmínku stojí hodnoty posledních dvou sloupců z tabulky 6. Předposlední sloupec (Počet kandidátů) velikost množiny formálních konceptů (kandidátů). K množině mandatorních konceptů postupně přidáváme tyto kandidáty a formulujeme tak pokrytí. V posledním sloupci je počet nalezených mandatorních konceptů. Jak jde vidět z tabulky 6 u grafu *emea* se pokrytí skládá téměř výhradně z mandatorních konceptů, naproti tomu pokrytí grafu *americas-large* tvoří mandatorní

Jméno	Uživatelé	Práva	Hrany	Zaplnění(%)
adult	20	10	100	50.00
advertisement	3279	1557	45139	0.88
chess	3196	76	118252	48.68
dblp	6980	19	17173	12.95
dna	4590	392	26527	1.47
mushroom	8124	119	186852	19.33
nfs	12841	4894	564462	0.90
paleo	501	139	3537	5.08
post	90	25	720	32.00
servo	167	19	668	21.05
shuttle	15	23	105	30.43
tic_tac_toe	958	30	9580	33.33
zoo	101	28	862	30.48

Tabulka 4: Ostatní datasety

Název souboru	Uplynulý čas (sec.)	Počet biklik	Velikost jádra	Redukce (%)
adult	0.007	8	60	40.00
advertisement	9.511	705	6	99.99
chess	247.552	?	71295	39.71
dblp	0.561	19	0	100.00
dna	1.779	366	0	100.00
mushroom	519.004	?	82661	55.76
nfs	5298.240	?	554790	1.71
paleo	0.151	139	0	100.00
post	0.020	?	556	22.78
servo	0.012	?	624	6.59
shuttle	0.007	13	16	84.76
tic_tac_toe	0.926	?	9580	0.00
zoo	0.202	24	86	90.02

Tabulka 5: Výsledky algoritmu Via Reduction na ostatních datasetech

Název souboru	Uplynulý čas (sec.)	Počet biklik	Počet kandidátů	Mand. koncepty
americas_large	?	*470	35981	187
americas_small	?	*189	2612	84
apj	?	*456	256	419
customer	?	*?	28032	126
domino	1.231000	20	35	12
emea	0.470000	34	352	32
firewall1	?	*66	277	24
firewall2	0.057000	10	15	5
healthcare	0.004000	14	14	11

Tabulka 6: Backtracking výsledky na datasetech z [1]

koncepty ze $(187/398) * 100 = 47\%$. V praxi se u grafu *americas-large* začalo s pokrytím o velikosti 187, které obsahovalo všechny mandatorní koncepty. Aby se našlo pokrytí obsahující 469 konceptů (o jedno lepší než doposud nalezené), musí se projít podmnožiny kandidátů velikosti $469-187 = 282$, kterých je $\binom{35981}{282}$ (velmi vysoké číslo). Toto je hlavní slabina tohoto algoritmu - již při poměrně malých číslech (v řádech několika desítek formálních konceptů) se počet různých podmnožin dostane do velkých čísel.

5.2 Generování dat

V této části probereme metody generování náhodného bipartitního grafu. Výsledkem bude matice sousednosti, kterou uložíme do souboru. Následující metody budou pracovat se zadanými argumenty jako jsou výška h a šířka w výsledné matice (neboli počet uzlu v partitě U a počet uzlů v partitě P), hustota d matice (číslo v procentech značící s jakou pravděpodobností bude na daném místě v matici hodnota 1) a pro druhou metodu i počet rolí r.

5.2.1 Metoda Random Graph

Pod tímto názvem si představme klasické generování náhodného grafu, takže vytvoříme matici M dané výšky a šířky. Následně pro každou pozici $M_{i,j}$, kde i je index řádku a j je index sloupce, určíme její hodnotu:

$$M_{i,j} = \begin{cases} 1 & \text{Pokud } b * 100 \leq d, \text{ kde } b \text{ je náhodné číslo z intervalu [0,1]} \\ 0 & \text{jinak} \end{cases}$$

5.2.2 Metoda Random RBAC

Problémem předchozí metody je, že získáme bipartitní graf, ve kterém velká spousta biklik, což znamená, že graf není moc provázaný. Další nevýhoda je

nemožnost dopředu říci, kolik biklik chci v grafu. To řeší následující metoda. Pro hodnoty výška h matice, šířka w matice, hustota d matice (v procentech) a počet rolí r je postup znázorněn v algoritmu 16.

Algoritmus 13: KnuthsArrayShuffle(arr, len)

```
/* Knuth: The art of computer programming [20]    */
for j ← len − 1 downto 1 do
    rand ← náhodné číslo z intervalu [0, 1]
    k ← ⌊j ∗ rand⌋
    SwapInArray(arr, j, k)
end
```

Algoritmus 14: GetRoleToStartIndex(w, r)

```
RTSI ← pole o velikosti w + 1 vyplněné nulami.
for i ← 0 to w do
    u ← P_i  mod r
    RTSI_u ← RTSI_u + 1
end
startIndex ← 0
for i ← 0 to r do
    temp ← RTSI_i
    RTSI_i ← startIndex
    startIndex ← RTSI + temp
end
RTSI_w ← RTSI_{w−1} + 1
return RTSI
```

Algoritmus 15: AssignUserToRole(M, P, w, $user$, $role$, $RTSI$)

```
for i ← RTSI_role to RTSI_{role+1} − 1 do
    perms ← P_i
    M_{user,perms} ← 1
end
```

Algoritmus 16: RandomRBAC(h, w, r)

```
M ← matice s výškou h a šířkou w, která je plná nul
/* pomocná pole (U)sers, (P)ermissions a RTSI      */
U ← pole velikosti h, vyplněné hodnotami 0, 1, ..., h − 1
P ← pole velikosti w, vyplněné hodnotami 0, 1, ..., w − 1
KnuthsArrayShuffle(P, w)
KnuthsArrayShuffle(U, h)
RTSI ← GetRoleToStartIndex(w, r)
/* přiřaď každé roli alespoň jednoho uživatele    */
for i ← 0 to r do
    AssignUserToRole(M, P, w, U_i, i, RTSI)
end
/* Každému uživateli přiřaď alespoň 1 roli        */
for i ← 0 to h do
    roles ← 0
    for role ← 0 to r do
        b ← náhodné číslo z intervalu [0, 1]
        if b * 100 ≤ d then
            AssignUserToRole(M, P, w, U_i, role, RTSI)
            roles ← roles + 1
        end
    end
    if roles = 0 AND i ≥ r then
        role ← náhodné číslo z intervalu [0, r − 1]
        AssignUserToRole(M, P, w, U_i, role, RTSI)
    end
end
return M
```

5.3 Testování nad generovanými daty

5.3.1 Tabulkové výsledky

V tabulkách 7, 8 a 9 jsou vypsané zjištěné údaje běhu algoritmů nad vygenerovanými daty, Vždy se vygenerovalo 20-50 náhodných grafů. Aby se tabulky vlezly šířkou na jednu stránku, byly názvy sloupců výrazně zkráceny. Sloupce V, $\check{S}$ a H odkazují na výšku, šířku a hustotu vygenerovaných matic.

Vysvětlení názvů sloupců v tabulkách 7 a 8:

V - výška vygenerovaných matic,

Š - šířka vygenerovaných matic,

H - hustota vygenerovaných matic (v procentech),

Čas hotových - průměrný čas výpočtu biklik, ale pouze u těch grafů, nad kterými stihl algoritmus doběhnout v daném časovém limitu,

Bikl. hotov. - průměrný počet biklik u grafů, nad kterými stihl algoritmus doběhnout,

Jádro hotov. - průměrná velikost jádra, nad kterými stihl algoritmus doběhnout,

Počet hotov. - počet grafů (v procentech), nad kterými algoritmus úspěšně doběhl v časovém limitu (5 minut),

Čas (sec.) výp. jádra všech - průměrný čas v sekundách výpočtu velikosti jádra nad všemi grafy stejných rozměrů,

Red. (%) všech - průměrný počet uzlů (v procentech), které se podařilo zredukovat, aby vzniklo neredukovatelné jádro,

Jádro všech - průměrná velikost jádra všech všech grafů stejných rozměrů.

V tabulce 7 byl použit při generování grafů algoritmus RandomGraph. V tabulce 8 byl naopak použit algoritmus RandomRBAC, ve kterém byl nastaven parametr počet rolí na náhodné číslo z intervalu $\left[\frac{1}{4}s, \frac{1}{2}s\right]$, kde s je výška nebo šířka matice (zde to není důležité, protože hodnoty jsou stejné).

V tabulce 9 se nachází srovnání algoritmu Backtracking se tříděním (v tabulce jako BT1) a bez třídění (v tabulce BT2). Všechny hodnoty jsou zprůměrovány pouze z těch grafů, nad kterými stihl algoritmus doběhnout v časovém limitu 2 minut. Vysvětlení názvů sloupců v tabulce 9 (kde za x můžeme dostadit 1 nebo 2):

V - výška vygenerovaných matic,

Š - šířka vygenerovaných matic,

H - hustota vygenerovaných matic (v procentech),

BTx čas (sec.) - průměrný čas běhu algoritmu BTx,

BTx bikl. - průměrný počet biklik algoritmu BTx

BTx kand. - průměrný počet formálních konceptů, ze kterých se generovaly podmnožiny u algoritmu BTx,

BTx hotov. - počet grafů (v procentech) nad kterými algoritmus BTx stihl doběhnout v daném časovém limitu

V	Š	H	Čas hotových	Bikl. hotov.	Jádro hotov.	Počet hotov.	Čas (sec.) výp. jádra všech	Red. (%) všech	Jádro všech
5	5	30	0.00046	3.64	0.12	100	0.00108	98.67	0.12
5	5	50	0.00070	3.96	0.84	100	0.00090	94.12	0.84
5	5	70	0.00102	3.40	0.90	100	0.00116	94.82	0.90
7	7	30	0.00072	5.58	0.68	100	0.00082	95.95	0.68
7	7	50	0.00104	5.72	3.48	100	0.00120	86.24	3.48
7	7	70	0.00148	4.88	7.28	100	0.00174	78.15	7.28
10	10	30	0.00112	8.92	7.34	100	0.00134	78.20	7.34
10	10	50	0.00312	8.72	26.16	100	0.00162	47.51	26.16
10	10	70	0.00754	7.02	31.60	100	0.00184	54.35	31.60
12	12	30	0.00288	11.20	19.30	100	0.00144	59.04	19.30
12	12	50	0.03484	10.56	47.20	100	0.00164	34.12	47.20
12	12	70	0.12096	8.24	54.02	100	0.00252	45.83	54.02
15	15	30	0.03470	14.62	41.22	100	0.00176	39.79	41.22
15	15	50	12.47060	13.62	90.08	100	0.00190	19.51	90.08
17	17	30	0.22204	16.76	67.24	100	0.00156	22.21	67.24
17	17	50	90.82197	15.74	118.55	62	0.00212	13.63	124.72
20	20	30	11.74898	20.00	104.66	100	0.00164	13.03	104.66
20	20	35	30.30206	20.00	121.25	80	0.00165	9.15	125.00
22	22	15	0.03545	20.30	29.40	100	0.00185	59.50	29.40
22	22	25	2.29805	21.80	103.00	100	0.00170	14.30	103.00
25	25	15	0.04145	23.70	47.25	100	0.00220	50.12	47.25
25	25	25	20.81435	25.00	140.00	100	0.00180	9.86	140.00
30	30	15	1.64430	29.65	101.25	100	0.00200	25.55	101.25
30	30	20	34.24088	30.00	160.12	85	0.00225	9.38	165.90
40	40	10	13.26380	38.70	110.50	100	0.00270	30.89	110.50
40	40	15	71.93243	39.79	219.71	70	0.00200	6.93	222.05
50	50	7	9.49345	47.50	85.85	100	0.00330	51.10	85.85
50	50	10	62.62823	49.62	213.85	65	0.00235	12.45	218.30
60	60	5	2.29595	54.80	52.70	100	0.00360	71.71	52.70
60	60	7	51.15494	58.67	189.78	90	0.00240	24.48	190.55
70	70	5	13.74182	66.35	136.76	85	0.00285	43.55	143.70
70	70	7	34.16900	68.75	303.38	40	0.00315	10.84	304.90
80	80	3	0.00920	68.05	14.80	100	0.00430	92.56	14.80
80	80	5	77.15500	78.00	227.73	55	0.00330	26.47	236.00
90	90	3	0.54085	79.50	33.20	100	0.00345	86.55	33.20
90	90	4	114.29129	86.43	184.14	35	0.00340	38.47	200.30

Tabulka 7: RandomGraph a algoritmus Via Reduction

V	Š	H	Čas hotových	Bikl. hotov.	Jádro hotov.	Počet. hotov.	Čas (sec.) výp. jádra všech	Red. (%) všech	Jádro všech
5	5	30	0.00158	1.00	0.00	100	0.00114		0.00
5	5	50	0.00144	1.00	0.00	100	0.00106	100.00	0.00
5	5	70	0.00132	1.00	0.00	100	0.00114	100.00	0.00
7	7	30	0.00142	2.00	0.00	100	0.00124	100.00	0.00
7	7	50	0.00170	2.00	0.00	100	0.00138	100.00	0.00
7	7	70	0.00198	2.80	0.72	100	0.00150	98.11	0.72
10	10	30	0.00286	2.00	0.00	100	0.00220	100.00	0.00
10	10	50	0.00288	3.00	0.00	100	0.00228	100.00	0.00
10	10	70	0.00370	3.80	2.48	100	0.00294	96.81	2.48
12	12	30	0.00264	4.00	0.00	100	0.00236	100.00	0.00
12	12	50	0.00364	3.00	0.00	100	0.00336	100.00	0.00
12	12	70	0.00450	3.96	2.18	100	0.00416	97.98	2.18
15	15	30	0.00378	4.00	0.00	100	0.00344	100.00	0.00
15	15	50	0.00530	6.98	12.30	100	0.00472	90.05	12.30
17	17	30	0.00456	6.00	0.00	100	0.00380	100.00	0.00
17	17	50	0.00678	5.00	0.86	100	0.00610	99.47	0.86
20	20	30	0.00586	6.00	0.00	100	0.00530	100.00	0.00
20	20	35	0.00702	5.00	0.00	100	0.00628	100.00	0.00
22	22	15	0.00436	10.00	0.00	100	0.00390	100.00	0.00
22	22	25	0.00648	8.00	0.00	100	0.00588	100.00	0.00
25	25	15	0.00520	11.00	0.00	100	0.00468	100.00	0.00
25	25	25	0.00684	10.00	0.94	100	0.00632	99.52	0.94
30	30	15	0.00750	8.00	0.00	100	0.00666	100.00	0.00
30	30	25	0.00858	14.00	38.06	100	0.00474	85.38	38.06
40	40	10	0.00644	19.00	0.00	100	0.00556	100.00	0.00
40	40	15	0.00672	16.00	0.00	100	0.00584	100.00	0.00
50	50	10	0.00598	24.00	0.00	100	0.00526	100.00	0.00
50	50	15	0.00534	15.00	0.00	100	0.00490	100.00	0.00
60	60	10	0.00610	28.00	0.00	100	0.00510	100.00	0.00
60	60	7	0.00582	19.00	0.00	100	0.00478	100.00	0.00
70	70	10	0.02300	33.00	34.60	100	0.00582	94.08	34.60
70	70	7	0.00566	19.00	0.00	100	0.00494	100.00	0.00
80	80	3	0.00668	32.98	0.00	100	0.00452	100.00	0.00
80	80	5	0.00568	23.00	0.00	100	0.00484	100.00	0.00
80	80	7	0.00620	24.00	0.00	100	0.00498	100.00	0.00
90	90	3	0.00606	39.00	0.00	100	0.00484	100.00	0.00
90	90	5	0.00576	38.00	0.00	100	0.00492	100.00	0.00
90	90	7	0.00694	42.00	0.00	100	0.00610	100.00	0.00

Tabulka 8: RandomRBAC a algoritmus Via Reduction

V	Š	H	BT1 čas (sec.)	BT1 bikl.	BT1 kand.	BT1 hotov.	BT2 čas (sec.)	BT2 bikl.	BT2 kand.	BT2 hotov.
5	5	30	0.00120	1.00	0.00	100	0.00086	1.00	0.00	100
5	5	50	0.00098	1.00	0.00	100	0.00088	1.00	0.00	100
5	5	70	0.00094	1.00	0.00	100	0.00094	1.00	0.00	100
7	7	30	0.00096	2.00	0.00	100	0.00100	2.00	0.00	100
7	7	50	0.00098	2.00	0.00	100	0.00106	2.00	0.00	100
7	7	70	0.00104	2.80	2.26	100	0.00106	2.80	2.26	100
10	10	30	0.00092	2.00	0.00	100	0.00096	2.00	0.00	100
10	10	50	0.00102	3.00	0.82	100	0.00102	3.00	0.82	100
10	10	70	0.00144	3.80	9.48	100	0.00168	3.80	9.48	100
12	12	30	0.00102	4.00	1.62	100	0.00102	4.00	1.62	100
12	12	50	0.00090	3.00	1.10	100	0.00102	3.00	1.10	100
12	12	70	0.00160	3.96	10.00	100	0.00170	4.00	10.00	100
15	15	30	0.00116	4.00	1.32	100	0.00120	4.00	1.32	100
15	15	50	13.10192	6.98	43.13	94	16.21494	7.00	42.94	92
17	17	30	0.00180	6.00	10.16	100	0.00166	6.00	10.16	100
17	17	50	0.00432	5.00	17.56	100	0.00580	5.00	17.56	100
20	20	30	0.00138	6.00	8.76	100	0.00150	6.00	8.76	100
20	20	35	0.00118	5.00	6.12	100	0.00122	5.00	6.12	100
22	22	15	0.02770	10.00	16.72	100	0.02862	10.00	16.72	100
22	22	25	0.08640	8.00	23.70	100	0.08352	8.00	23.70	100
25	25	15	0.26516	11.00	21.74	100	0.26582	11.00	21.74	100
25	25	25	11.91766	10.00	40.68	82	12.42963	10.00	40.68	82
30	30	15	0.00128	8.00	4.42	100	0.00140	8.00	4.42	100

Tabulka 9: RandomRBAC a algoritmy Bactracking se tříděním (BT1) a bez třídění (BT2)

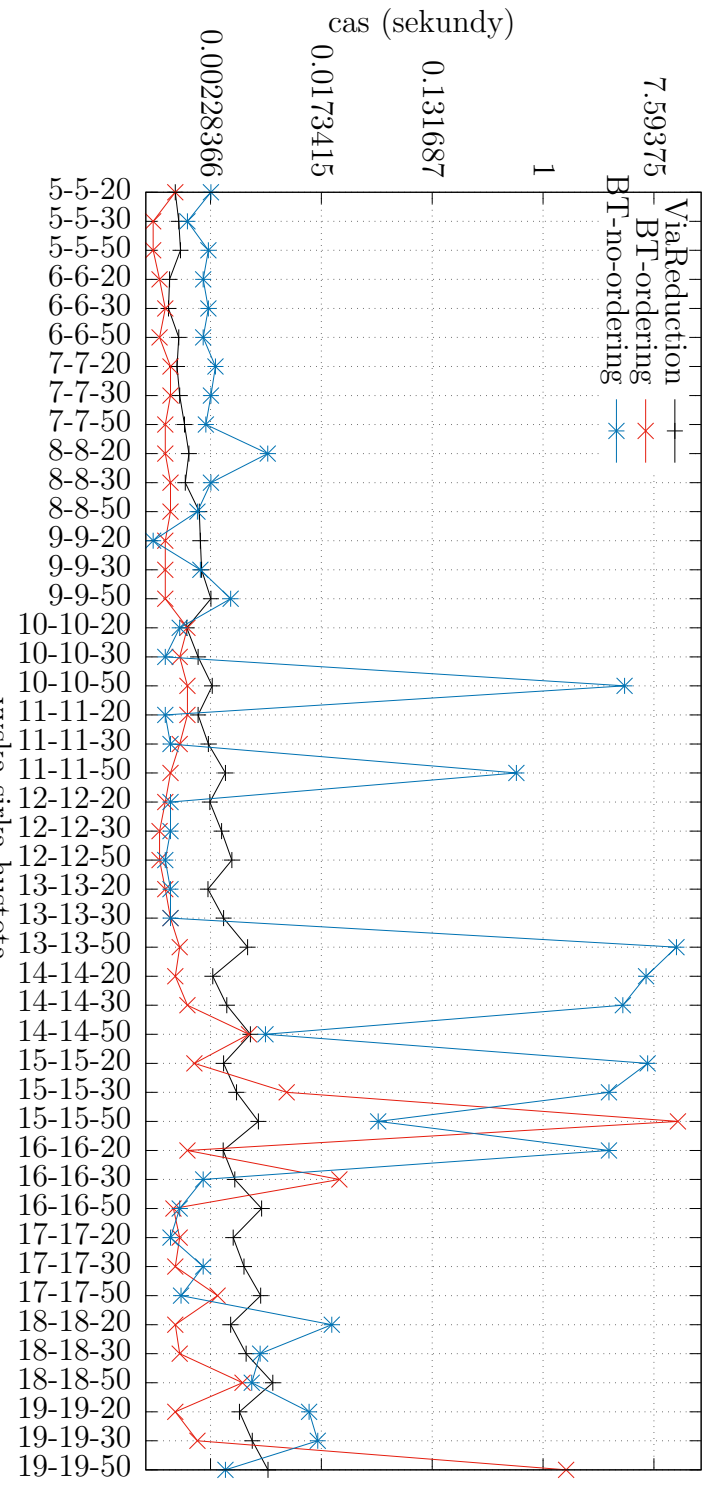
5.3.2 Výsledky zobrazené v grafech

V grafu 17 a 18 jsem se pokusil porovnat rychlosti algoritmů nad stejnými vygenerovanými daty. Porovnával jsem algoritmus Via Reduction s Backtracking algoritmem se tříděním (BT-ordering) a bez třídění (BT-no-ordering). Na horizontální ose (pozor graf je zrotovaný, aby se vešel na stránku a zároveň byly čitelné hodnoty na ose) jsou názvy velikostí grafu, které značí výšku, šířku a hustotu vygenerovaných matic. Na vertikální ose je čas v sekundách (škála není lineární ale logaritmická, protože mely hodnoty vysoký rozptyl a na lineární ose by nebyly malé hodnoty zřetelné). Vždy se vygenerovalo 30 grafů stejných rozměrů algoritmem RandomGraph.

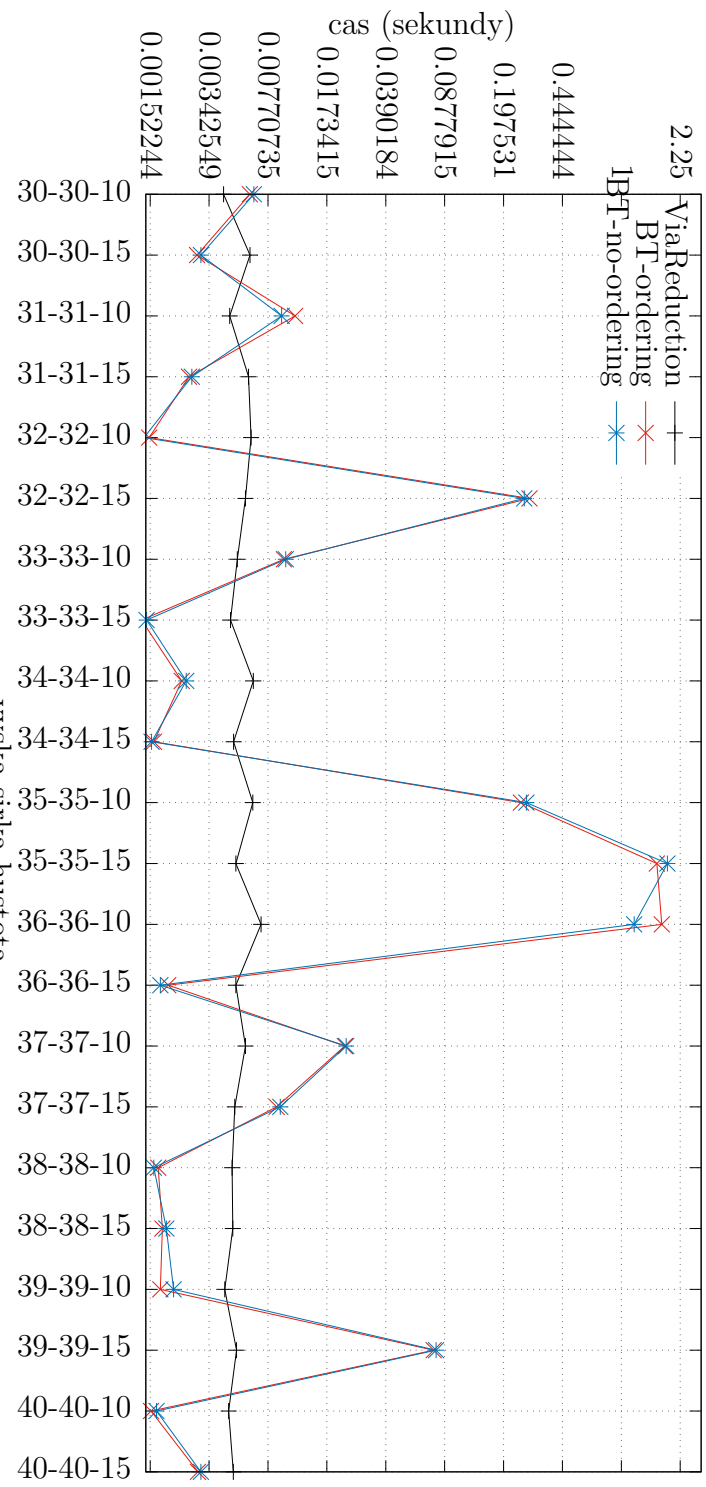
Z grafu 17 jde vidět, že se obecně strategie třídění u Backtracking algoritmu vyplatila. Nutno dodat, že v některých případech verze se tříděním doběhne, zatímco verze bez třídění ne. U takto malých dat se v algoritmu Via Reduction vyskytovalo velmi malé jádro, se kterým si algoritmus na barvení grafu rychle poradil, proto jde z grafů vyčíst, že jsou rychlosti konzistentní (neobjevují se tu přílišné výkyvy v časech). To samé nelze říci o BT algoritmu. Zde platilo, že pokud se počet nalezených formálních konceptů dostal přes určitou hranici a zároveň bylo málo mandatorních konceptů, výpočet trval déle.

V grafu 18 jsou vygenerovány větší matice, které se generovaly stejným způsobem jako v předchozím grafu. U grafů s takovou velikostí se již projevovaly nedostatky BT algoritmu. Některé vygenerované grafy měly větší velikost množiny kandidátů, což se následně promítlo do času běhu.

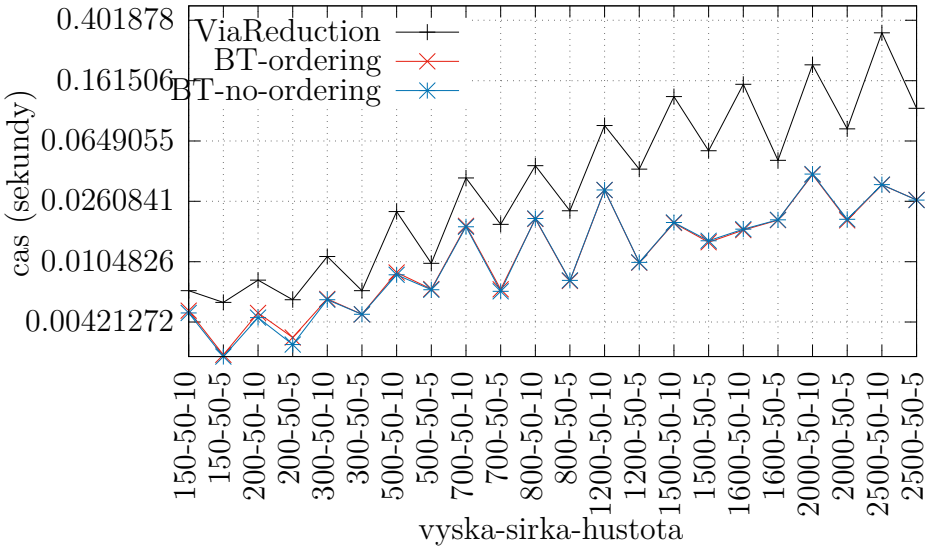
Na grafu 19 jsem generoval postupně matice se zafixovanou šířkou, ale rostoucí výškou. Grafy jsem generoval algoritmem RandomRBAC, který bere jako parametr počet rolí, který jsem nastavil na náhodné číslo z intervalu $[12, 25]$. U algoritmu Via Reduction s většími daty rostl pouze čas potřebný pro redukci, jelikož se podařilo redukovat všechny grafy tak, aby nezbylo žádné neredukovatelné jádro. Co se BT algoritmu týče, pokrytí se skládala výhradně z mandatorních konceptů, takže výpočet byl rychlý (bez generování podmnožin). Z takto nastavených parametrů by se dalo vyvodit, že výpočet všech formálních konceptů a následně všech mandatorních konceptů (u BT algoritmu) byl rychlejší než výpočet redukce (u Via Reduction).



Obrázek 17: Porovnání algoritmů 1



Obrázek 18: Porovnání algoritmů 2



Obrázek 19: Porovnání algoritmů 3

Závěr

Cílem práce bylo prostudovat a implementovat algoritmus pro nalezení optimálního řešení problému pokrytí bipartitního grafu biklikami z práce [1].

Na začátku práce bylo potřeba definovat určité pojmy a algoritmické problémy, které souvisely s danou problematikou, aby měl čtenář dobrý znalostní základ při následném popisu algoritmu Via Reduction, ve kterém bylo hlavním cílem podrobně popsat, jak tento algoritmus funguje. Toho bylo docíleno mimo jiné i popisem v pseudokódech a názorným příkladem. Dále byla snaha o návrh vlastního algoritmu vracejícího optimální řešení, který by měl sice větší časovou složitost, ale sloužil by k ověření, že algoritmus Via Reduction opravdu funguje. Zároveň bylo zajímavé podívat se na problém z jiného úhlu pohledu a to sice z pohledu formální konceptuální analýzy.

Zde je nutné podotknout, že výběr algoritmu na obarvení grafu nebyl nejlepší. Na datasetech z práce [1] si sice vedl dobře, ale v poslední fázi diplomové práce (při testování) se ukázalo, že na jiné datasety nebo na náhodně vygenerované grafy (určité velikosti a hustoty) tento algoritmus nemusí být dostačující. Jako vylepšení bych tedy navrhoval zvolit lepší algoritmus na obarvení grafu.

Přínosem práce je dle mého názoru zreplikování algoritmu formou implementace v programovacím jazyce C a následné potvrzení na experimentech, že algoritmus opravdu funguje.

Conclusions

The aim of the work was to study and implement the algorithm for finding an optimal solution for biclique cover problem from article [1].

At the beginning of the work it was necessary to define certain terms and algorithmic problems that were related to this topic. Then I described algorithm Via Reduction in which the main goal was to describe in detail how this algorithm works. This was achieved, among other things, by a description in pseudocodes and an illustrative example. There was also an effort to design my own algorithm returning an optimal solution, which would have more time complexity, but would serve to verify that the Via Reduction algorithm really works. At the same time, it was interesting to look at the problem from a different point of view, namely from the point of view of formal conceptual analysis.

It should be noted here that the choice of the algorithm for coloring the graph was not the best. It performed well on datasets from the work [1], but in the last phase of the master thesis (during testing) it turned out that this algorithm may not be sufficient for other datasets or randomly generated graphs (certain sizes and densities). As an improvement, I would suggest choosing a better algorithm for coloring the graph.

The benefit of the work is, in my opinion, the replication of the algorithm in the form of implementation in the C programming language and subsequent confirmation in experiments that the algorithm really works.

A Dokumentace souboru MinimalBicliqueCover.h

Veškerý kód se nachází v souboru `MinimalBicliqueCover.h` a je napsán v programovacím jazyce C s využitím MSVC (*Microsoft Visual C++*). Pro použití stačí nakopírovat do určitého adresáře se zdrojovými soubory a vložit do kódu následující: `#include "MinimalBicliqueCover.h"`. Soubor je rozdělen do skupin, které dříve tvořily vlastní moduly. Každá skupina je nadepsaná vlastním komentářem.

Výčet skupin:

Random.h obsahuje pomocné funkce pro generování pseudonáhodných čísel.

Utils.h obsahuje funkce, které se vyskytují napříč celým kódem, například velikost pole, makro pro kontrolu, zda se alokovala paměť a další.

BitSet.h obsahuje implementaci číselných množin. Reprezentace a množinové operace jsou implementované na bitové úrovni. Uložení množin je sice náročnější na paměť (pokud bychom chtěli uchovávat čísla z velkého intervalu), ale operace nad nimi jsou velice efektivní.

Matrix.h obsahuje definici matice a pomocné funkce pro práci s maticemi.

EdgesList.h cyklický dvojitě propojený seznam, který obsahuje hrany (2 číselné hodnoty). Používá se v algoritmu Via Reduction.

Queue.h fronta a funkce pro práci s frontou - enqueue a dequeue.

RedirectOutput.h pro přesměrování výstupu z obrazovky do souboru a zpět.

Set.h implementace množin pomocí hash tabulek.

SetInt.h implementace číselných množin pomocí hash tabulek.

Stack.h zásobník a funkce pro práci se zásobníkem - push a pop.

StackInt.h celočíselný zásobník.

Stopwatch.h funkce pro měření dobu běhu.

TupleInt.h struktura pro uchování dvojice čísel.

GraphColoring (Furini, Gabrel) implementace obarvení grafu, realizované funkcí `GraphColoringDSATUR`.

Biclique struktura pro reprezentaci bikliky.

Bicliques funkce pro práci s biklikami.

Minimal Biclique cover (Exact cover Ene) implementace algoritmu Via Reduction, funkce `MinimalBicliqueCoverViaReduction`. Dále obsahuje funkce `MBCViaReductionSizeOfKernel` pouze pro výpočet velikosti neredukovatelného jádra.

Minimal Biclique cover (Backtracking) implementace backtracking algoritmu. Funkce `MinimalBicliqueCoverBacktracking`.

B Ukázková konzolová aplikace

Pro příklad použití předchozího souboru jsem vytvořil ukázkovou aplikaci. Tato aplikace byla zároveň využita v kapitole 5.

Režimy a vstupní parametry aplikace se ovládají pomocí přepínačů. Aplikace má 4 módy:

1. GEN (`--gen`),
2. TEST (`--test`),
3. RES (`--res`),
4. HELP (`--help`),

ke kterým se přistupuje pomocí přepínačů uvedených v závorkách.

B.1 Módy

B.1.1 GEN (`--gen`)

Tento mód slouží pro generování testovacích případů.

Přepínače:

-alg <a> pro nastavení algoritmu, kterým se budou testovací případy generovat. Za <a> lze dosadit čísla:

0 - generování pomocí algoritmu Random Graph, kapitola 5.2.1,

1 - generování pomocí algoritmu Random RBAC, kapitola 5.2.2.

Defaultní hodnota je 0.

-out <path> pro nastavení cesty složky, do které se uloží vygenerované testovací případy. Pokud cesta obsahuje mezeru, je nutné mít cestu uvedenou v uvozovkách.

-count <c> pro nastavení počtu vygenerovaných testovacích případů. Defaultní hodnota je 1.

-size <n> pro nastavení počtu řádků (<n>) a sloupců (<n>) ve vygenerované matici. Defaultní velikost matice je 10x10.

-size <n> <m> pro nastavení počtu řádků (<n>) a sloupců (<m>) ve vygenerované matici. Defaultní velikost matice je 10x10.

-density <d> pro nastavení hustoty matice. <d> je v jednotkách procent (celé číslo v intervalu $[0, 100]$). Například při nastavení hodnoty 100 bude matice celá vyplněna jedničkami. S hodnotou 50 je 50% šance, že na dané pozici v matici bude 1. Defaultní hodnota je 50.

-roles <r> pro nastavení předpokládaného počtu rolí ve vygenerovaných testovacích případech (platí pouze pro algoritmus Random RBAC). Jestliže není tento přepínač použit, je zvoleno náhodné celé číslo z intervalu $[1, min(h, w)]$, kde h je výška matice a w je šířka matice.

B.1.2 TEST (`--test`)

Testovací mód, který vygeneruje informace (jako například rychlost v sekundách) o běhu algoritmu nad danými vstupy.

Přepínače:

-alg <a> pro nastavení algoritmu. Za <a> lze dosadit čísla:

- **0** - algoritmus Via Reduction, kapitola 3
- **1** - algoritmus Backtracking kapitola 4,
- **2** - algoritmus Backtracking kapitola 4 bez třídění,
- **3** - algoritmus KernelSize (jako Via Reduction, ale výsledkem je pouze velikost neredukovatelného jádra).

Defaultní hodnota je 0.

-in <path> pro nastavení vstupní cesty. Tato cesta může mít různé významy (složka, soubor), které se specifikují čtyřmi následujícími přepínači. Defaultní je přepínač `--file`

--file vstupní cesta odkazuje na jeden soubor s koncovkou .txt.

--files vstupní cesta odkazuje na složku obsahující soubory s koncovkou .txt. Ve výstupním souboru (nebo závěrečném souhrnu na obrazovce) bude mít každý soubor v této složce vlastní výstup. (To znamená že ve výstupním souboru bude mít každý soubor svůj vlastní řádek.)

--dir vstupní cesta odkazuje na složku obsahující soubory s koncovkou .txt, ale narozdíl od předchozího přepínače se všechny výsledky zprůměrují a zapíší na jeden řádek ve výstupním souboru nebo na obrazovku.

--dirs vstupní cesta odkazuje na složku, která obsahuje podsložky. Jednotlivé podsložky budou mít ve výstupním souboru svůj řádek, ve kterém bude zprůměrovaný výsledek.

-out <path> pro nastavení výstupní cesty odkazující na soubor (s koncovkou .csv), do kterého se budou zapisovat výsledky. Pokud nebyl použit tento přepínač, výsledky se po skončení testování pouze vytisknou na obrazovku.

-count <c> algoritmus se spustí <c>-krát pro zpřesnění naměřeného času. Defaultní hodnota je 1.

-limit <l> je čas v sekundách, který určije maximální dobu běhu algoritmu. Po uplynutí této doby je algoritmu dán pokyn pro ukončení. Toto ukončení nemusí nastat hned. Defaultní hodnota je 120 (2 minuty). Za <l> se dá dosadit `inf`, což znamená, že algoritmus nebude mít časový limit.

Zde je třeba upozornit na to, že pokud vstupní nebo výstupní cesta odkazuje na složku a zároveň obsahuje mezeru (takže je uvedena v uvozovkách), pak nesmí cesta končit symbolem zpětného lomítka (\).

B.1.3 RES (`--res`)

Tento mód slouží k vylistování výsledných biklik nebo neredukovatelného jádra ať už do souboru nebo jen na obrazovku.

Přepínače:

-alg <a> pro nastavení algoritmu. Za <a> lze dosadit čísla:

- **0** - algoritmus Via Reduction, kapitola 3
- **1** - algoritmus Backtracking kapitola 4,
- **2** - algoritmus Backtracking kapitola 4 bez třídění,
- **3** - algoritmus pro hledání neredukovatelného jádra (připraveného k obarvení).

Defaultní hodnota je 0.

-in <path> pro nastavení vstupní cesty, která odkazuje na soubor (s příponou .txt).

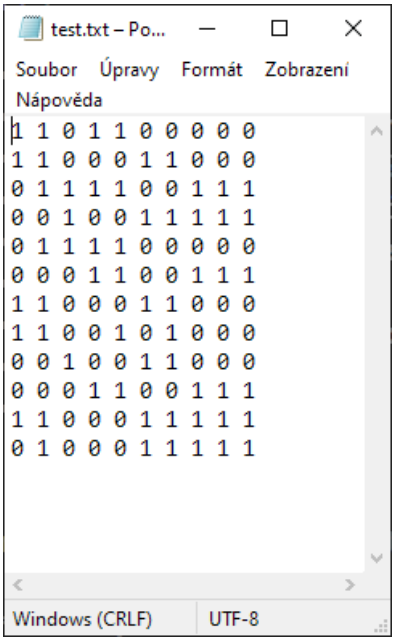
-out <path> pro nastavení výstupní cesty, která odkazuje na soubor (s příponou .txt).

-limit <l> je čas v sekundách, který určije maximální dobu běhu algoritmu. Po uplynutí této doby je algoritmu dán pokyn pro ukončení. Toto ukončení nemusí nastat hned. Defaultní hodnota je 120 (2 minuty). Za <l> se dá dosadit `inf`, což znamená, že algoritmus nebude mít časový limit.

B.1.4 HELP (`--help`)

Mód pro zobrazení nápovědy. Nemá žádné přepínače.

B.2 Vstupní data a výstupní data



Obrázek 20: Ukázka vstupního souboru

Vstupní data jsou soubory s příponou .txt, které reprezentují matici grafu. Na každém řádku v souboru je jeden řádek matice, prvky matice na řádku jsou odděleny mezerou. Na obrázku 20 vidíme vstupní matici, která má 12 řádků a 10 sloupců.

Výstupní data pro mód `--test` jsou ve formátu .csv. První řádek je hlavička, která obsahuje:

path cesta vstupního souboru, kterému náleží daný řádek.

alg číselná hodnota použitého algoritmu (daná přepínačem `-alg`).

timeout s hodnotou true (vypršel časový limit) nebo false (nevypršel časový limit).

elapsedTime uplynulý čas v sekundách.

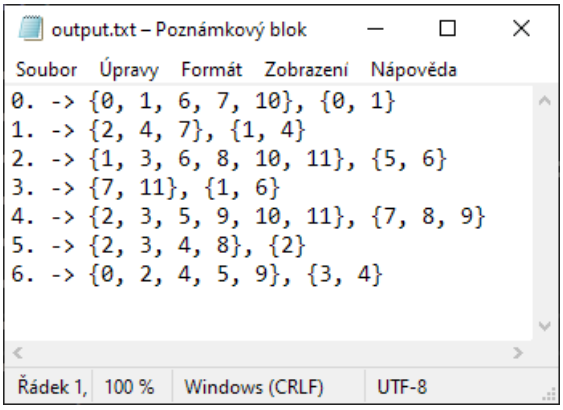
bicliques-reduction při použití algoritmu Kernel Size je to hodnota počtu uzlů (v procentech), které byly odebrány při redukci. Pro ostatní algoritmy je to počet biklik.

kernel-concepts při použití algoritmu Via Reduction nebo Kernel Size hodnota značí velikost neredukovatelného jádra, jinak značí počet konceptů (kandidátů), ze kterých se generují podmnožiny (nejsou v tom započítány mandatorní koncepty).

completed počet dokončených testů.

count celkový počet testů.

Výstupní data pro mód `--res` jsou sobory s příponou .txt. Při použití přepínačů `-alg 0`, `-alg 1` nebo `-alg 2` je výstupem soubor, ve kterém je na každém řádku jedna biklika.



Obrázek 21: Ukázka výstupního souboru pro `--res` znázorňující bikliky

Řádek 0. -> {0, 1, 6, 7, 10}, {0, 1} z obrázku 21 tedy značí, že se jedná o 0. bikliku, která obsahuje 7 uzlů: 5 uzlů z první partity a 2 uzly z druhé partity.

Při použití přepínače `-alg 3` je výstupem soubor s maticí, která reprezentuje neredukovatelné jádro, které je připravené k obarvení.

C Obsah přiloženého CD/DVD

bin/
: Ukázková aplikace *TestingApp.exe*

data/
: Datasety a testovací data z tabulek 7, 8 a 9 spolu s výslednými (naformátovanými) soubory .csv.

doc/
: Text práce ve formátu PDF a všechny soubory potřebné pro vygenerování PDF dokumentu.

src/
: Kompletní zdrojové texty ukázkového programu včetně hlavičkového souboru *MinimalBicliqueCover.h*.

literature/
: Obsahuje literaturu, na kterou se odkazovalo v práci, ale i na jinou související s tématem.

readme.txt
: Podrobný popis adresáře.

Literatura

[1] ENE, Alina, et al. Fast exact and heuristic methods for role minimization problems. Proceedings of the 13th ACM symposium on Access control models and technologies. 2008.

[2] Demel, Jiří. Grafy a jejich aplikace. Vyd. 2., (Vlastním nákladem 1.). Libčice nad Vltavou: J. Demel, 2015. ISBN 978-80-260-7684-1.

[3] J. Vaidya, V. Atluri, and Q. Guo. The role mining problem: Finding a minimal descriptive set of roles. In SACMAT '07, pages 175-184. ACM Press, 2007.

[4] Herbert Fleischner, Egbert Mujuni, Daniël Paulusma, Stefan Szeider. Covering graphs with few complete bipartite subgraphs. Theoretical Computer Science, Volume 410, Issues 21–23, 2009, Pages 2045-2053. ISSN 0304-3975

[5] James Orlin. Contentment in graph theory: Covering graphs with cliques. Indagationes Mathematicae (Proceedings), Volume 80, Issue 5, 1977, Pages 406-424. ISSN 1385-7258

[6] Michael R. Garey and David S. Johnson. 1990. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., USA.

[7] Daniel Brélaz. 1979. New methods to color the vertices of a graph. Commun. ACM 22, 4 (April 1979), 251–256.

[8] Furini, Fabio & Gabrel, Virginie & Ternier, Ian-Christopher. (2016). An Improved DSATUR-Based Branch-and-Bound Algorithm for the Vertex Coloring Problem. Networks. 10.1002/net.21716.

[9] Definice 1.3.1 (Rozklad množiny), http://www.math.muni.cz/ xberanj/VS/-kombo.pdf

[10] Patric R.J. Östergård. A fast algorithm for the maximum clique problem. Discrete Applied Mathematics, Volume 120, Issues 1–3, 2002, Pages 197-207. ISSN 0166-218X

[11] Hans Ulrich Simon. On Approximate Solutions for Combinatorial Optimization Problems. SIAM Journal on Discrete Mathematics 1990 3:2, 294-310.

[12] Radim Belohlavek. Introduction to formal concept analysis. Volume 47 (2008), Palacky University, Department of Computer Science, Olomouc.

[13] Gaume, Bruno and Navarro, Emmanuel and Prade, Henri. Clustering bipartite graphs in terms of approximate formal concepts and sub-contexts. (2013) International Journal of Computational Intelligence Systems, 6 (6). 1125-1142. ISSN 1875-6891

[14] Outrata, Jan. Computing and Applying Formal Concepts. Informatika, Masarykova univerzita, 2017.

[15] Sergei O. Kuznetsov. Interpretation on graphs and complexity characteristics of a search for specific patterns. (1989) Automatic Documentation and Mathematical Linguistics. 23.

[16] Radim Belohlávek, Vilém Vychodil. Discovery of optimal factors in binary data via a novel method of matrix decomposition. Journal of Computer and System Sciences Volume 76, Issue 1, February 2010, Pages 3-20.

[17] Oprávnění (informatika), wikipedie.cz, odkaz: https://cs.wikipedia.org/wiki/Opr%C3%A1vn%C4%9Bn%C3%AD_(informatika)

[18] Diestel, Reinhard. Graph Theory. Springer-Verlag Heidelberg, New York 1997, 2000, 2005.

[19] Bhasker, J., & Samad, T. (1991). The clique-partitioning problem. Computers and Mathematics with Applications, 22(6), 1-11.

[20] Donald E. Knuth. 1997. The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms. Addison-Wesley Longman Publishing Co., Inc., USA.

[21] H. Gruber and M. Holzer. Inapproximability of nondeterministic state and transition complexity assuming P $\neq$ NP. In T. Harju, J. Karhumki, and A. Lepist, editors, Developments in Language Theory, volume 4588 of LNCS, pages 205-216. Springer, Heidelberg, 2007.