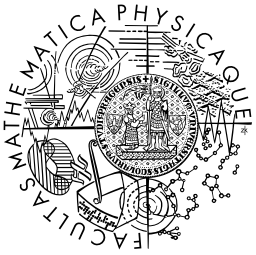


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Michal Berg

Aproximace nezávislosti rovinných grafů

Informatický ústav Univerzity Karlovy

Vedoucí bakalářské práce:	doc. Mgr. Zdeněk Dvořák, Ph.D.
Studijní program:	Informatika
Studijní obor:	Obecná informatika

Praha 2017

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

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

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

Název práce: Aproximace nezávislosti rovinných grafů

Autor: Michal Berg

Katedra: Informatický ústav Univerzity Karlovy

Vedoucí bakalářské práce: doc. Mgr. Zdeněk Dvořák, Ph.D., Informatický ústav Univerzity Karlovy

Abstrakt: Problém nezávislé množiny je dobře známý NP-úplný problém, který je NP-úplný i pro rovinné grafy. Ale na rozdíl od obecných grafů, pro rovinné grafy existuje polynomiální aproximační schéma. Popíšeme přesný algoritmus pro hledání největší nezávislé množiny v rovinných grafech založený na dynamickém programování. Tento přesný algoritmus lze jednoduše upravit na polynomiální aproximační schéma. Obě jeho verze jsme implementovali a otestovali. Při tom jsme používali několik generátorů náhodných rovinných grafů. Přesný algoritmus jsme experimentálně srovnávali s dalšími dvěma algoritmy. Aproximační algoritmus jsme srovnávali s jeho přesnou verzí a měřili skutečný aproximační poměr a také jeho časovou náročnost v porovnání s přesnou verzí. Zjistili jsme, že přesný algoritmus na zvolených grafech většinou dokončí výpočet rychleji než ostatní dva algoritmy. Také jsme zjistili, že aproximační verze má vzhledem k teoretickému minimu většinou lepší aproximační poměr s dobrou časovou složitostí.

Klíčová slova: rovinné grafy, nezávislá množina, aproximace

Title: Approximation of independence number of planar graphs

Author: Michal Berg

Department: Computer Science Institute of Charles University

Supervisor: doc. Mgr. Zdeněk Dvořák, Ph.D., Computer Science Institute of Charles University

Abstract: The independent set problem is a well-known NP-complete problem, which is NP-complete even for planar graphs. But unlike general graphs, there exists an polynomial-time approximation scheme for planar graphs. We are going to describe an exact algorithm for maximum independent set problem in planar graphs based on dynamic programming. This algorithm can be easily modified to an polynomial-time approximation scheme. We implemented both versions of this algorithm and tested them. We used a few random planar graph generators for that. We compared the exact algorithm with another two algorithms. We compared the approximation algorithm with its exact version and measured its real approximation ratio and also its time complexity in comparison with the exact version. We discovered that the exact algorithm usually finishes the computation faster than the other two algorithms. We also discovered that the approximation version has a better approximation ratio compared to the theoretical minimum with good time complexity.

Keywords: planar graphs, independent set, approximation

Rád bych poděkoval panu doc. Mgr. Zdeňkovi Dvořákovi, Ph.D. za odborné vedení, za poskytnuté rady a pomoc s vypracováním této práce.

Obsah

Úvod

V této práci se budeme zabývat největší nezávislou množinou v rovinných grafech. Nezávislá množina grafu $G = (V,E)$ je množina vrcholů $S \subseteq V$ taková, že žádná dvojice vrcholů z množiny S není spojena hranou. Hledání největší nezávislé množiny je známý problém z teorie grafů, pro daný graf $G = (V,E)$ chceme najít jeho nezávislou množinu S největší velikosti. Jedná se o optimalizační problém, ale existuje i rozhodovací varianta problému nezávislé množiny taková, že se pro daný graf $G = (V,E)$ a celé číslo k ptáme, jestli existuje nezávislá množina velikosti alespoň k. O této rozhodovací variantě problému je dobře známo, že se jedná o NP-úplný problém, důkaz lze nalézt třeba v knize (Garey a Johnson, 1979) kapitole 3.1.3.

Pokud $P \neq NP$, tak nemůže existovat polynomiální algoritmus pro hledání největší nezávislé množiny v obecných grafech. Nejtriviálnější algoritmus pracuje v čase $O(2^n n^2)$ a to tak, že pro každou podmnožinu množiny vrcholů otestujeme, zda se jedná o nezávislou množinu a vybereme největší nalezenou. Lepšími algoritmy jsou například algoritmy založené na metodě rozděl a panuj, první popsaný v článku (Fomin a kol., 2006) s časovou složitostí $O^*(1{,}2210^n)$ a druhý v článku (Fomin a kol., 2009) s časovou složitostí $O^*(1{,}2201^n)$. Článek (Bourgeois a kol., 2012) se zabývá algoritmy pro grafy s nízkým průměrným stupněm vrcholu. Prezentuje algoritmy s časovými složitostmi $O^*(1{,}0854^n)$, $O^*(1{,}1571^n)$, $O^*(1{,}1918^n)$ a $O^*(1{,}2070^n)$ pro grafy s průměrným stupněm vrcholu maximálně 3, 4, 5 a 6 a také algoritmus pro obecné grafy s časovou složitostí $O^*(1{,}2125^n)$.

Pokud tedy nedokážeme najít největší nezávislou množinu v polynomiálním čase, můžeme se ptát, jestli bychom byli schopni v polynomiálním čase najít alespoň nějakou její dobrou aproximaci. Dá se ukázat, že neexistuje aproximační algoritmus s konstantním aproximačním poměrem, který by pracoval v polynomiálním čase. Což tedy znamená, že pro žádnou konstantu $0 < c < 1$ neexistuje polynomiální algoritmus takový, že pro každý graf G je schopný najít nezávislou množinu velikosti alespoň $c|S|$, kde S je největší nezávislá množina grafu G. Dokonce v článku (Håstad, 1999) je dokázáno, že neexistuje žádný polynomiální aproximační algoritmus s aproximačním poměrem $1/n^{1-\varepsilon}$ pro jakékoliv $\varepsilon > 0$, kde n značí počet vrcholů daného grafu.

Existují ale třídy grafů, ve kterých se dají největší nezávislé množiny hledat v polynomiálním čase. Jednou takovou třídou jsou bipartitní grafy, kde tento problém souvisí s maximálním párováním. Problém maximálního párování pro graf $G = (V,E)$ spočívá v tom, že hledáme množinu hran $M \subseteq E$ maximální velikosti takovou, že každý vrchol $v \in V$ je součástí maximálně jedné hrany z M. Maximální párování pro bipartitní grafy, stejně jako pro obecné grafy, lze nalézt v polynomiálním čase. Existuje Kőnigova věta, která dává do souvislosti párování a vrcholové pokrytí, publikovaná v knize (Bondy, 1976) jako věta 5.3. Vrcholové pokrytí grafu $G = (V,E)$ je množina vrcholů $K \subseteq V$ taková, že pro každou hranu $e \in E$ je alespoň jeden z jejích vrcholů v množině K. Kőnigova věta říká, že velikost maximálního párování v bipartitním grafu G je rovna velikosti minimálního vrcholového pokrytí a dokonce jsme schopni v polynomiálním čase minimální vrcholové pokrytí z maximálního párování sestrojit. Protože množina K je vrcholovým pokrytím právě tehdy, když $V \setminus K$ je nezávislou množinou, tak

jsme v polynomiálním čase schopni nalézt i největší nezávislou množinu.

Rovinné grafy ale nejsou jednou z tříd, kde by se největší nezávislá množina dala hledat v polynomiálním čase. V článku (Garey a kol., 1974) je dokázáno, že vrcholové pokrytí v rovinných grafech je NP-úplný problém. Protože vrcholové pokrytí a nezávislá množina jsou na sebe polynomiálně převoditelné problémy tak, že zachovávají rovinnost, tudíž i problém nezávislé množiny v rovinných grafech je NP-úplný problém. Stejně jako pro obecné grafy, nemůžeme předpokládat, že by pro největší nezávislou množinu v rovinném grafu existoval polynomiální algoritmus. V článku (Lipton a Tarjan, 1979) je publikován algoritmus, který pracuje v čase $2^{O(\sqrt{n})}$.

Stejně jako u obecných grafů se můžeme ptát, jestli existuje nějaký dobrý aproximační algoritmus. Ale na rozdíl od nich, pro rovinné grafy existuje polynomiální aproximační schéma. Tedy pro každé $0 < c < 1$ existuje polynomiální algoritmus takový, že pro každý rovinný graf G najde nezávislou množinu velikosti alespoň $c|S|$, kde S je největší nezávislá množina daného grafu. Takové polynomiální aproximační schéma je popsáno v článku (Baker, 1994). Tento článek obsahuje přesný algoritmus pro hledání největší nezávislé množiny, který sice pracuje v čase lineárním s počtem vrcholů, ale exponenciálním s počtem úrovní grafu. Pokud máme rovinný graf $G = (V,E)$ včetně jeho nakreslení, tak úroveň vrcholu $v \in V$ je o jedna větší než nejkratší cesta (měřená počtem hran) z v do libovolného vrcholu na vnější stěně grafu G. Graf má pak stejně úrovní, jako je maximální úroveň nějakého z jeho vrcholů. Tento přesný algoritmus se pak dá jednoduše upravit na polynomiální aproximační schéma.

V této práci se budeme zabývat algoritmy popsanými v článku (Baker, 1994) zmíněném v předchozím odstavci, jak přesným tak i aproximačním. Tyto algoritmy implementujeme a experimentálně srovnáme jejich časovou složitost v praxi s implementacemi jiných algoritmů. V kapitole 1 popíšeme tyto algoritmy včetně popisu naší implementace, v kapitole 2 popíšeme další algoritmy, se kterými budeme srovnávat a také způsob generování testovacích grafů. Nakonec v kapitole 3 prezentujeme naměřené výsledky.

1. Popis algoritmu a jeho implementace

Hlavním algoritmem, kterým se budeme v této práci zabývat, je algoritmus publikovaný v článku (Baker, 1994). V tomto článku je prezentován přesný algoritmus založený na dynamickém programování. Tento algoritmus pracuje tak, že dělí graf na části nazývané plátky. Algoritmus hledá největší nezávislou množinu v daném plátku a pak plátky spojuje do větších plátků, až získáme plátek reprezentující celý graf a tím i nezávislou množinu v tomto grafu. V části 1.1 uvedeme potřebné definice k popisu algoritmu, v části 1.2 popíšeme algoritmus, v části 1.3 se podíváme na naši implementaci, v části 1.4 popíšeme úpravu na aproximační verzi a nakonec v části 1.5 uvedeme uživatelskou dokumentaci naší implementace.

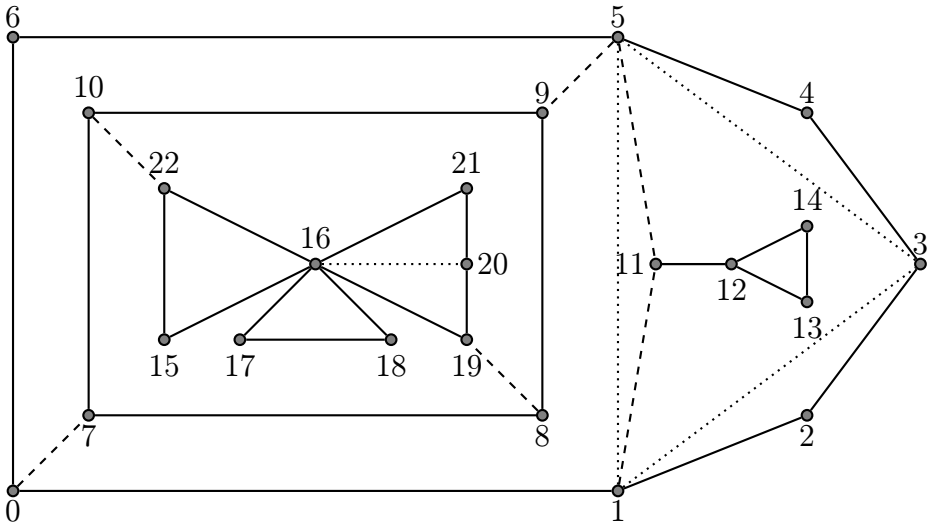
1.1 Definice pojmů

V této části definujeme potřebné pojmy pro popis algoritmu. Hlavním pojmem bude plátek, na kterém je celý algoritmus založen. Plátek bude tvořen nějakým podgrafem vstupního grafu. Plátek bude mít také hranici tvořenou nějakými svými vrcholy. Jak už jsme zmínili, algoritmus pracuje tak, že hledá největší nezávislou množinu v daném plátku. A nejen to, také pro každou pevně zvolenou podmnožinu hranice hledá největší nezávislou množinu, která obsahuje přesně tyto vrcholy hranice a žádné jiné. To se hodí k tomu, že budeme chtít postupně plátky spojovat a konstruovat největší nezávislou množinu spojeného plátku pomocí největších nezávislých množin plátků, které jsme spojili. Jak uvidíme později, plátky budeme definovat pomocí jiných plátků, a to nám určí, jak budeme plátky spojovat. Teď všechny potřebné pojmy nadefinujeme.

Mějme graf $G = (V,E)$, pro který chceme najít velikost největší nezávislé množiny. Předpokládejme, že graf G je souvislý. Pokud není, tak algoritmus použijeme na jeho jednotlivé komponenty souvislosti. Nejprve definujeme pojmy i-komponenta a i-stěna, které jsou velmi důležité pro popis algoritmu. Tyto definice jsou závislé na rovinném nakreslení grafu G, proto odteď předpokládejme, že ke grafu G máme i nějaké jeho rovinné nakreslení.

Definice 1. *Podgraf indukovaný vrcholy na vnější stěně grafu G nazveme komponentou úrovně 1, neboli 1-komponentou. Cyklus f grafu G je stěna úrovně i, neboli i-stěna, pokud je vnitřní stěnou komponenty úrovně i. Nechť f je stěna nějaké úrovně, pak G_f je podgraf indukovaný vrcholy uvnitř stěny f. Nechť f je stěna úrovně $i-1$, pak podgraf indukovaný vrcholy na vnější stěně grafu G_f nazveme komponentou úrovně i, neboli i-komponentou. Vrchol $v \in V$ je úrovně i, neboli i-vrchol, pokud leží v nějaké komponentě úrovně i.*

Protože předpokládáme, že všechny grafy, se kterými pracujeme, jsou souvislé, tak slovem komponenta vždy myslíme i-komponentu, kde i bude zřejmé z kontextu. V předchozí definici může nastat situace, že i-komponenta není souvislá. Této situaci se chceme vyhnout, protože v algoritmu vyžaduje speciální ošetření. Prozatím pro jednoduchost předpokládejme, že tato situace nenastala, tedy že pro každou i-stěnu f je graf indukovaný vrcholy uvnitř této i-stěny f souvislý.



Obrázek 1.1: Příklad grafu G

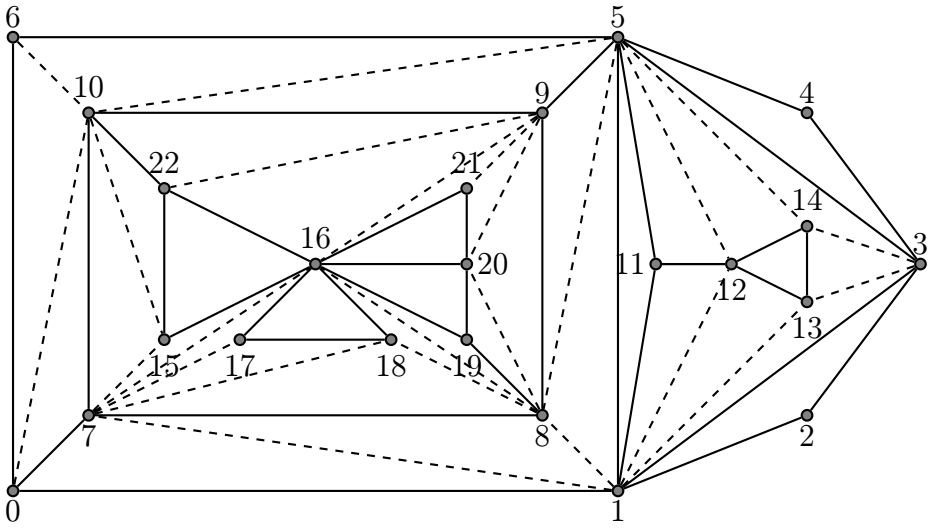
Jak vyřešit situaci, kde tento předpoklad neplatí, ukážeme později. Jestliže nějaká i-komponenta obsahuje most, nahradíme ho dvěma paralelními hranami tak, aby se jednalo o stěnu a nemuseli jsme tento případ speciálně rozebírat. Rozdělením vrcholů do úrovní získáme několik typů hran.

Definice 2. *Hrana $e = uv \in E$ je vnější, pokud vrcholy u a v leží v jedné i-komponentě a hrana e leží na vnější stěně této i-komponenty. Hrana $e = uv \in E$ je vnitřní, pokud vrcholy u a v leží v jedné i-komponentě a hrana e neleží na vnější stěně této i-komponenty. Hrana $e = uv \in E$ je meziúrovňová, pokud vrcholy u a v nejsou stejné úrovně.*

Pro lepší názornost uvedeme příklad, uvažujme graf G na obrázku 1.1. Vrcholy 0 až 6 jsou úrovně 1, vrcholy 7 až 14 jsou úrovně 2 a vrcholy 15 až 22 jsou úrovně 3. Graf indukovaný vrcholy 0 až 6 je komponenta úrovně 1. Grafy indukované vrcholy 7 až 10 a 11 až 14 tvoří komponenty úrovně 2 a graf indukovaný vrcholy 15 až 22 tvoří komponentu úrovně 3. Dále uveďme nějaké příklady i-stěn, například cyklus 0-1-5-6 tvoří stěnu úrovně 1 a cyklus 15-16-22 tvoří stěnu úrovně 3. Hrany nakreslené plnou čarou jsou vnější, tečkované hrany jsou vnitřní a čárkované hrany jsou meziúrovňové.

Pro další popis algoritmu budeme potřebovat triangulace oblastí mezi každou $(i-1)$-stěnou a její vnořenou i-komponentou, které získáme přidáním dalších hran. Tyto nově přidané hrany nazveme triangulační a takto upravený graf označíme G'. Pro funkčnost algoritmu nezáleží na volbě triangulace, můžeme ji tedy zvolit libovolně. Může se stát, že přidáním triangulačních hran vzniknou násobné hrany, což ale v algoritmu nebude představovat problém. Příklad triangulace grafu G z obrázku 1.1 je zobrazen na obrázku 1.2, kde čárkovaně jsou znázorněny přidané triangulační hrany. Dále už budeme předpokládat, že pracujeme vždy s grafem G'.

Zaveďme následující konvenci. Pokaždé, když mluvíme o nějaké vnější hraně $e = uv$, tak u je před v při obcházení vrcholů na vnější stěně dané i-komponenty proti směru hodinových ručiček. Algoritmus potřebuje nějakou počáteční hranu na vnější stěně, ze které bude začínat. Pro definování ostatních pojmů je potřeba,



Obrázek 1.2: Příklad grafu G', triangulace grafu G z obrázku 1.1

aby tato počáteční hrana byla pevně zvolena. Nechť tedy $s = ab$ je pevně zvolená hrana, která leží na vnější stěně grafu G.

Dále budeme chtít pro každou i-stěnu definovat identifikátor, který bude tvořit buď vnitřní hrana nebo vrchol. Než ale budeme moci identifikátor i-stěny f definovat, potřebujeme znát počáteční hranu a vrchol i-komponenty, které je f součástí. Pro 1-komponentu je to jednoduché, jako počáteční hranu vezmeme pevně zvolenou hranu s a jako počáteční vrchol vezmeme a. Pro i-komponentu C, $i > 1$, je to složitější. K tomu, abychom mohli definovat počáteční hranu a vrchol pro komponentu C, musíme znát identifikátor $(i-1)$-stěny, ve které je komponenta C vnořena. Pro potřeby následující definice tedy předpokládejme, že pro každou i-komponentu je počáteční hrana pevně zvolená vnější hrana této komponenty a počáteční vrchol je levý vrchol zvolené hrany. Jak tyto hrany a vrcholy zvolíme, definujeme později. Pokud řekneme, že vrchol v není obsažen v identifikátoru d, myslíme tím, že buď d je vrchol různý od v nebo d je hrana a vrchol v je různý od obou vrcholů hrany d. Pokud řekneme, že hrana e není shodná s identifikátorem d, myslíme tím, že buď d není hrana nebo d je hrana různá od e.

Definice 3. *Nechť C je i-komponenta s počátečním vrcholem u a počáteční hranou $e = uv$. Pokud je i-stěna f součástí komponenty C a zároveň obsahuje hranu e, pak identifikátor stěny f je vrchol u. Pokud je vrchol u artikulací komponenty C, tedy existuje nějaká vnější hrana $e' = uw$ komponenty C taková, že $v \neq w$, tak i-stěna f obsahující hranu e' je identifikovaná vrcholem u.*

Nechť f je i-stěna komponenty C s identifikátorem d a nechť e' je nějaká její hrana, která není shodná s identifikátorem d a zároveň je vnitřní hranou komponenty C. Pak i-stěna f' různá od i-stěny f obsahující také hranu e' je identifikovaná hranou e'.

Nechť f je i-stěna komponenty C s identifikátorem d a w je nějaký její vrchol, který není obsažen v identifikátoru d a zároveň je artikulací komponenty C a nechť $e' = wx$ je poslední vnější hrana komponenty C při průchodu proti směru hodinových ručiček od počáteční hrany e obsahující vrchol w jako levý. Pak pro každou vnější hranu wy různou od e' je i-stěna f' obsahující tuto hranu identifikovaná vrcholem w.

Teď, když máme definované identifikátory i-stěn, můžeme postoupit k definici počátečních vrcholů a hran. Než to ale uděláme, zavedeme ještě konvenci, v jakém směru se díváme na vnitřní hrany. Mějme nějakou i-komponentu C s počáteční hranou $e = uv$ a počátečním vrcholem u. Když budeme mluvit o nějaké vnitřní hraně $e' = u'v'$ i-komponenty C, tak předpokládáme, že u' je před v' při průchodu vrcholů na vnější stěně komponenty C proti směru hodinových ručiček, kde vrchol v bereme jako první vrchol.

Definice 4. *Máme-li $(i-1)$-stěnu f identifikovanou vrcholem u, která má vnořenou i-komponentu C, pak za počáteční vrchol C zvolíme libovolný vrchol v z i-komponenty C, který je hranou spojený s vrcholem u. Pokud komponenta C není jednovrcholová, tak za její počáteční hranu zvolíme $e = vw$, která je první vnější hranou i-komponenty C při průchodu hran proti směru hodinových ručiček kolem vrcholu v od hrany vu.*

Máme-li $(i-1)$-stěnu f identifikovanou hranou $e = uv$, která má vnořenou i-komponentu C, pak za počáteční vrchol C zvolíme vrchol w z i-komponenty C, který je hranami spojený s oběma vrcholy u i v. Pokud komponenta C není jednovrcholová, tak za její počáteční hranu zvolíme hranu $e' = wx$, která je první vnější hranou i-komponenty C při průchodu hran proti směru hodinových ručiček kolem vrcholu w od hrany uw.

Identifikátory nemusí být jednoznačné, každá vnitřní hrana identifikuje právě jednu i-stěnu, ale vrchol může identifikovat více i-stěn. Vrchol u je identifikátorem i-stěny pouze, pokud je počátečním vrcholem nebo se jedná o artikulaci. V následující definici definuje pořadí i-stěn identifikovaných stejným vrcholem.

Definice 5. *Nechť C je i-komponenta s počáteční hranou e a nechť v je vrchol komponenty C, který je identifikátor nějaké i-stěny. Nechť $f_1, \ldots, f_k$ jsou všechny stěny identifikované vrcholem v. Pak pořadí těchto i-stěn je určeno podle pořadí hran $v^N_{f_j}$ na vnější hranici komponenty C při obcházení vnějších hran proti směru hodinových ručiček od hrany e.*

Pro lepší pochopení uvedeme příklad, uvažujme stále graf G' z obrázku 1.2. Jako počáteční vrchol 1-komponenty zvolíme vrchol 0 a jako počáteční hranu vezmeme hranu (0,1). Pak 1-stěna 0-1-5-6 je identifikována vrcholem 0, 1-stěna 1-3-5 je identifikována hranou (1,5), 1-stěna 1-2-3 je identifikována hranou (1,3) a 1-stěna 3-4-5 je identifikována hranou (3,5). Za počáteční vrchol 2-komponenty 7-8-9-10 vezmeme vrchol 7, ale mohli bychom zvolit i vrchol 10, protože je také hranou spojen s vrcholem 0. Počáteční hranou této 2-komponenty je hrana (7,8). U druhé 2-komponenty 11-12-13-14 už je volba počátečního vrcholu jednoznačná, protože stěna 1-3-5, ve které je tato komponenta vnořená, má za identifikátor hranu (1,5), takže za počáteční vrchol musíme zvolit vrchol, který je hranami spojený jak s vrcholem 1, tak i s vrcholem 5, což je jedině vrchol 11. Počáteční hrana této 2-komponenty pak bude hrana (11,12). Dále určíme identifikátory 2-stěn této komponenty. Stěna 11-12, což je 2-stěna, protože každý most nahrazujeme dvojicí paralelních hran, bude mít za identifikátor vrchol 11. A 2-stěna 12-13-14 bude mít identifikátor vrchol 12. Teď se dostáváme k jediné 3-komponentě grafu G'. U této komponenty máme zase na výběr, jak zvolit počáteční vrchol, protože 2-stěna 7-8-9-10, ve které je komponenta vnořena, má za identifikátor vrchol 7, takže jako počáteční vrchol můžeme zvolit libovolný vrchol komponenty spojený s vrcholem

7, tedy 15, 16, 17 a 18. Pro tento příklad zvolíme vrchol 15 a jako počáteční hranu tedy vezmeme hranu (15,16). Pak 3-stěna 15-16-22 bude mít identifikátor vrchol 15, 3-stěna 16-17-18 bude mít identifikátor vrchol 16, 3-stěna 16-19-20 bude mít identifikátor také vrchol 16 a 3-stěna 16-20-21 bude mít identifikátor hranu (20,16).

Dále pro zjednodušení zavedeme značení pro následující a předchozí hrany.

Definice 6. *Je-li e vnější hrana komponenty C, pak e_N je následující hrana a e_P je předchozí hrana při průchodu vnějších hran komponenty C proti směru hodinových ručiček od hrany e. Je-li e hrana i-stěny f, pak e_f^N je následující a e_f^P je předchozí hrana při průchodu hran i-stěny f proti směru hodinových ručiček od hrany e. Je-li v vrchol i-stěny f, pak v_f^N je následující a v_f^P je předchozí hrana při průchodu hran i-stěny f proti směru hodinových ručiček od vrcholu v.*

Důvodem, proč v předcházející definici předchozí a následující hrany stěny používáme i označení i-stěny, ale u komponenty ho nepoužíváme, je, že každá hrana může ležet pouze v jedné komponentě, ale zároveň může ležet ve více i-stěnách. Dále musíme ještě pro každou vnější hranu úrovně 2 a vyšších definovat dělící hrany, levou a pravou, které budou tvořeny meziúrovňovými nebo triangulačními hranami a také definovat následující a předchozí stěnové hrany meziúrovňových a triangulačních hran. Tyto hrany budeme potřebovat při definici hranice plátku.

Definice 7. *Nechť C je i-komponenta vnořená v $(i-1)$-stěně f a nechť $e = uv$ je počáteční hrana komponenty C a u je její počáteční vrchol. Pro každou vnější hranu $e' = u'v'$ komponenty C různou od e_P definujeme pravou dělící hranu, značenou ${e'}_R$, jako první hranu proti směru hodinových ručiček při průchodu incidentních hran s vrcholem v' od hrany e'. Dále pro každou vnější hranu $e' = u'v'$ komponenty C různou od e definujeme levou dělící hranu, značenou ${e'}_L$, jako pravou dělicí hranu hrany přecházející hranu e', tedy ${e'}_L = ({e'}_P)_R$. Pokud má stěna f jako identifikátor vrchol w, tak definujeme levou dělící hranu hrany e a pravou dělící hranu hrany e_P jako hranu wu. Pokud má stěna f jako identifikátor hranu xy, tak definujeme levou dělící hranu e jako wx a pravou dělící hranu e_P jako hranu wy.*

Definice 8. *Nechť $e = uv$ je meziúrovňová nebo triangulační hrana taková, že vrchol u leží v i-komponentě C a vrchol v leží na $(i-1)$-stěně f ve které je komponenta C vnořena. Pak následující stěnová hrana e_{FN} je první hrana stěny f, která je po směru hodinových ručiček od hrany e kolem vrcholu v a předchozí stěnová hrana e_{FP} je první hrana stěny f, která je proti směru hodinových ručiček od hrany e kolem vrcholu v.*

Dále musíme zavést levé a pravé okolí počátečního vrcholu i-komponenty. Ty budeme potřebovat při definici hranice plátku, protože budeme muset hranici pro plátky hran z okolí počátečního vrcholu definovat jinak.

Definice 9. *Nechť C je i-komponenta s počátečním vrcholem u a počáteční hranou $e = uv$ vnořená v $(i-1)$-stěně f. Pak levé okolí počátečního vrcholu u obsahuje hranu e, pokud hrany e, e_L a e_R tvoří trojúhelník, a všechny hrany e' takové, že hrany e', e'_L a e'_R tvoří trojúhelník a zároveň hrana e'_P je součástí levého okolí vrcholu u. Pravé okolí počátečního vrcholu u obsahuje hranu e_P, pokud*

hrany e_P, $(e_P)_L$ a $(e_P)_R$ tvoří trojúhelník, a všechny hrany e' takové, že hrany e', e'_L a e'_R tvoří trojúhelník a zároveň hrana e'_N je součástí pravého okolí vrcholu u.

Konečně se dostáváme k definici plátku. Plátek bude tvořit podgraf grafu G', takže než podáme přesnou definici, můžeme si plátky představovat jako nějaké podgrafy. Plátky budeme definovat pro vnější hrany a i-stěny, kde plátek i-stěny budeme přiřazovat identifikátoru této stěny, tedy vnitřní hraně, artikulaci nebo počátečnímu vrcholu. Když řekneme, že plátek je úrovně i, myslíme tím, že se jedná buď o plátek pro vnější hranu i-komponenty, nebo plátek pro i-stěnu. Když budeme mluvit o plátku vnitřní hrany nějaké i-komponenty, budeme tím myslet plátek pro i-stěnu identifikovanou touto hranou. U plátku pro i-stěnu identifikovanou vrcholem to bude složitější z důvodu nejednoznačnosti identifikátorů. Když budeme mluvit o plátku i-stěny identifikované vrcholem, myslíme tím plátek této stěny. Pokud ale mluvíme o plátku vrcholu, myslíme tím sjednocení plátků i-stěn, které jsou tímto vrcholem identifikované.

Každý plátek bude mít hranici skládající se ze dvou částí, levé a pravé, které budeme nazývat levou a pravou hranicí. Toto je hlavní myšlenka algoritmu, protože plátky definujeme tak, aby měly krátké hranice, konkrétně délky $2i$ pro plátek úrovně i. Jak uvidíme později, časová složitost algoritmu závisí exponenciálně na délce hranice plátku, ale pouze lineárně na počtu vrcholů grafu. Proto se algoritmus hodí pro grafy s malým počtem úrovní a jak popíšeme později i ke konstrukci aproximačního algoritmu, kde budeme uměle omezovat počet úrovní. Nejprve definujeme hranici plátku.

Definice 10.

- *Je-li $e = uv$ vnější hrana 1-komponenty, pak plátek pro hranu e má levou hranici u a pravou hranici v.*
- *Je-li f i-stěna identifikovaná hranou e, pak plátek pro stěnu f má levou hranici shodnou s levou hranicí plátku pro hranu e_f^N a pravou hranici shodnou s pravou hranicí plátku pro hranu e_f^P. Je-li f i-stěna identifikovaná vrcholem v, pak plátek pro stěnu f má levou hranici shodnou s levou hranicí plátku pro hranu v_f^N a pravou hranici shodnou s pravou hranicí plátku pro hranu v_f^P.*
- *Nechť v je vrchol takový, že existuje i-stěna identifikovaná vrcholem v, pak plátek pro vrchol v má levou hranici shodnou s levou hranicí plátku pro první i-stěnu identifikovanou vrcholem v a pravou hranici shodnou s pravou hranicí plátku pro poslední i-stěnu identifikovanou vrcholem v.*
- *Nechť f je i-stěna s identifikátorem d a nechť e je nějaká hrana i-stěny f, která není shodná s identifikátorem d. Pak rozšířený plátek pro hranu e má levou hranici shodnou s levou hranicí plátku pro e a pravá hranice rozšířeného plátku pro e je buď shodná s pravou hranicí plátku pro vrchol v, pokud plátek pro vrchol v existuje a zároveň vrchol v není obsažen v identifikátoru d, jinak je shodná s pravou hranicí plátku pro hranu e.*
- *Nechť C je i-komponenta pro $i > 1$ s počátečním vrcholem x vnořená v $(i-1)$-stěně f a nechť $e = uv$ je vnější hrana i-komponenty C. Pak*

levá hranice plátku pro e začíná vrcholem u a pravá hranice plátku pro e začíná vrcholem v. Pokud e je součástí levého okolí vrcholu x, tak obě hranice plátku pro e pokračují levou hranicí rozšířeného plátku pro hranu $(e_L)_{FN}$. Pokud e je součástí pravého okolí vrcholu x, tak obě hranice plátku pro e pokračují pravou hranicí rozšířeného plátku pro hranu $(e_R)_{FP}$. Pokud e není součástí ani levého ani pravého okolí vrcholu x, tak levá hranice plátku pro e pokračuje levou hranicí rozšířeného plátku pro hranu $(e_L)_{FN}$ a pravá hranice plátku pro e pokračuje pravou hranicí rozšířeného plátku pro $(e_R)_{FP}$.

Když budeme mluvit o pořadí vrcholů na hranici, tak předpokládáme, že vrchol nejvyšší úrovně bereme jako první a vrchol nejnižší úrovně jako poslední. V předchozí definici jsme použili pojem rozšířený plátek, který formálně definujeme společně s plátkem. Bude se jednat o sjednocení plátku pro hranu a pro vrchol. Důvodem pro zavedení tohoto pojmu je, že chceme plátek pro artikulaci v spojit s právě jedním plátkem pro hranu, která má v jako pravý vrchol. Ještě než vyslovíme formální definici plátku, musíme zavést následující značení.

Definice 11. *Nechť f je i-stěna a $e = uv$ a $e' = u'v'$ jsou její dvě hrany. Pak označme $e_1 = e, e_2, \ldots, e_{n-1}, e_n = e'$ hrany při průchodu hran i-stěny f proti směru hodinových ručiček a označme $v_1 = v, v_2, \ldots, v_{n-2}, v_{n-1} = u', v_n = v'$ vrcholy při průchodu vrcholů i-stěny f proti směru hodinových ručiček. Pak $V(e \to e')$ označuje množinu vrcholů $\{u, v_1, \ldots, v_n\}$. Pro každé $i = 1, \ldots, n$ nechť $P(e_i)$ označuje rozšířený plátek hrany e_i. Pak*

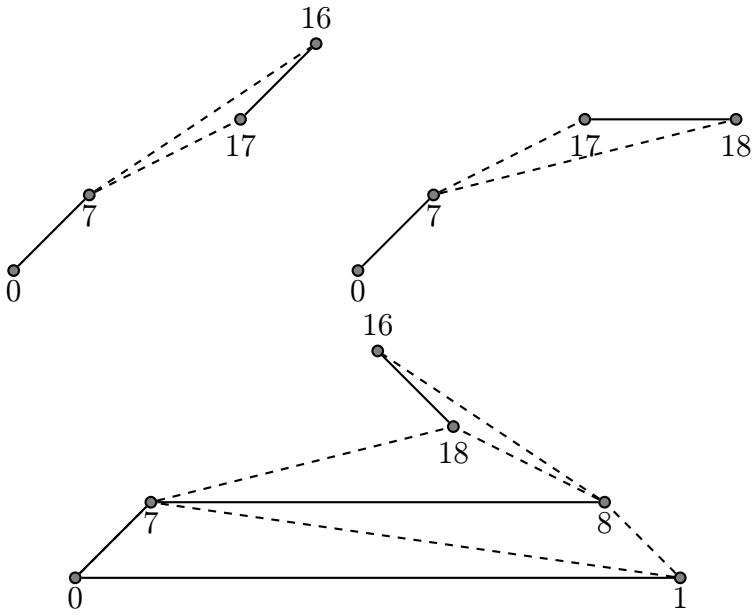
$$\bigcup(e \to e') = P(e_1) \cup P(e_2) \cup \cdots \cup P(e_n)$$

značí sjednocení rozšířených plátků pro hrany.

Následuje definice plátku.

Definice 12.

- *Je-li e vnější hrana 1-komponenty, pak plátek pro hranu e obsahuje pouze hranu e.*
- *Je-li f i-stěna identifikovaná hranou e, která nemá vnořenou žádnou komponentu, pak sjednocení rozšířených plátků $\bigcup(e_f^N \to e_f^P)$ plus hrana e je plátkem pro stěnu f. Je-li f i-stěna identifikovaná vrcholem v, která nemá vnořenou žádnou komponentu, pak plátek pro stěnu f je definujeme jako sjednocení rozšířených plátků $\bigcup(v_f^N \to v_f^P)$.*
- *Je-li f i-stěna identifikovaná hranou e, která má vnořenou komponentu C s počátečním vrcholem u, pak plátek pro hranu e získáme z plátku pro vrchol u zkrácením hranic o vrchol u a přidáním hrany e. Je-li f i-stěna identifikovaná vrcholem v, která má vnořenou komponentu C s počátečním vrcholem u, pak plátek pro stěnu f získáme z plátku pro vrchol u zkrácením hranic o vrchol u.*
- *Je-li v vrchol pro který existuje i-stěna, která je jím identifikovaná, pak plátek pro vrchol v je sjednocením plátků všech i-stěn, které jsou identifikovány vrcholem v, v pořadí definovaném v definici 5.*



Obrázek 1.3: Příklad plátků pro hrany (16,17), (17,18) a (18,16) grafu G' z obrázku 1.2

- *Nechť f je i-stěna s identifikátorem d a nechť $e = uv$ je nějaká hrana i-stěny f, která není shodná s identifikátorem d, pak rozšířený plátek pro hranu e je buď sjednocením plátku pro hranu e a plátku pro vrchol v, pokud plátek pro vrchol v existuje a zároveň vrchol v není obsažen v identifikátoru d, jinak je rozšířený plátek pro hranu e shodný s plátkem pro hranu e.*

- *Nechť C je i-komponenta pro $i > 1$ s počátečním vrcholem x vnořená v $(i-1)$-stěně f a nechť $e = uv$ je vnější hrana i-komponenty C. Pokud je hrana e součástí levého nebo pravého okolí počátečního vrcholu x nebo hrany e, $(e_L)_{FN}$ a $(e_P)_{FP}$ tvoří trojúhelník, tak plátek pro hranu obsahuje hranu e a vrcholy své hranice včetně hran mezi nimi. Jinak je plátek pro hranu e tvořen sjednocením rozšířených plátků $\bigcup((e_L)_{FN} \to (e_R)_{FP})$, hrany e a hran z vrcholů u a v do vrcholů z množiny $V((e_L)_{FN} \to (e_R)_{FP})$.*

Tímto máme definován plátek a jeho hranici. Teď uveďme nějaké příklady plátků grafu G' z obrázku 1.2 pro volby počátečních vrcholů shodně s předchozím příkladem. Na obrázku 1.3 jsou zobrazeny plátky pro hrany (16,17), (17,18) a (18,16). Teď se podíváme, jak jsme určili hranice těchto plátků. Začněme hranou $e = (16,17)$. Levá hranice začíná vrcholem 16 a pravá hranice začíná vrcholem 17. Levá dělící hrana pro hranu e je z definice rovna pravé dělící hraně hrany $e_P = (15,16)$, což je první hrana proti směru hodinových ručiček od hrany (15,16) kolem vrcholu 16, tedy hrana (16,7). Pravá dělící hrana pro hranu e je hrana (17,7). Protože hrany e, e_L a e_R tvoří trojúhelník a hrana $e_P = (15,16)$ je součástí levého okolí vrcholu 15, tak je jeho součástí i hrana e. Proto obě hranice plátku pro e pokračují levou hranicí plátku pro hranu $(e_L)_{FN} = (7,8)$. Levá hranice plátku pro hranu $e' = (7,8)$ začíná vrcholem 7. Protože hrana e' je počáteční hrana dané 2-komponenty a 1-stěna 0-1-5-6, ve které je tato 2-komponenta vnořena, má identifikátor vrchol 0, tak levá dělící hrana pro hranu e' je hrana (7,0). Proto tedy levá hranice plátku pro hranu e' pokračuje levou hranicí plátku

pro hranu $(e'_L)_{FN}$, tedy vrcholem 0. Tímto jsme určili, že levá hranice plátku pro hranu (16,17) je tvořena vrcholy 16, 7 a 0 a pravá hranice tohoto plátku je tvořena vrcholy 17, 7 a 0.

Určení hranice plátku pro hranu (17,18) je velmi podobné určení hranice plátku předchozí hrany, proto ho zde uvádět nebudeme. Podíváme se ale na hranice plátku pro hranu (18,16), kterou teď budeme označovat e. Nejprve určíme levou hranici, která začíná vrcholem 18. Levá dělící hrana pro hranu e je z definice shodná s pravou dělící hranou pro hranu e_P = (17,18), což je hrana (18,7). Tedy levá hranice plátku pro hranu e pokračuje levou hranicí plátku pro hranu $(18,7)_{FN}$ = (7,8), kterou jsme ale už určili v předchozím odstavci. Proto je levá hranice plátku pro hranu e tvořena vrcholy 18, 7 a 0. Teď určíme pravou hranici tohoto plátku, která začíná vrcholem 16. Pravá dělící hrana pro hranu e je hrana (16,8) a proto pravá hranice plátku pro hranu e pokračuje pravou hranicí rozšířeného plátku pro hranu $(e_R)_{FP}$ = (7,8), kterou označme e'. Protože neexistuje plátek pro vrchol 8, tak je pravá hranice rozšířeného plátku pro hranu e' shodná s pravou hranicí plátku pro e'. Pravá hranice plátku pro hranu e' začíná vrcholem 8 a pokračuje pravou hranicí rozšířeného plátku pro hranu $(e'_R)_{FP}$ = (0,1), tedy vrcholem 1. Tím jsme určili pravou hranici plátku pro hranu (18,16), která je tvořena vrcholy 16, 8 a 1. Pokud sloučíme plátky z obrázku 1.3 získáme tím plátek pro 3-stěnu 16-17-18. Z definice je levá hranice tohoto plátku shodná s levou hranicí plátku pro hranu (16,17) a jeho pravá hranice je shodná s pravou hranicí plátku pro hranu (18,16).

V algoritmu budeme pro každý plátek tvořit tabulku a tyto tabulky budeme spojovat. K tomu, abychom tabulky pro plátky mohli spojit, tak potřebujeme, aby plátky měli společnou hranici, tedy aby se pravá hranice prvního shodovala s levou hranicí druhého. Ve zbytku této části budeme chtít dokázat, že sousední plátky mají společnou hranici. Nejprve v následující definici ke každé hraně i-komponenty definujeme levou a pravou vnější hranu.

Definice 13. *Nechť $e = uv$ je hrana i-komponenty C. Pokud e je vnitřní hrana, pak e^{CW}_{OUT} označuje první vnější hranu při průchodu hran incidentních s vrcholem u od hrany e po směru hodinových ručiček a e^{CCW}_{OUT} označuje první vnější hranu při průchodu hran incidentních s vrcholem v od hrany e proti směru hodinových ručiček. Dále $d^{CW}_{OUT}(e)$ označuje počet vnitřních hran i-komponenty C při průchodu po směru hodinových ručiček kolem vrcholu u od hrany e k hraně e^{CW}_{OUT} a podobně $d^{CCW}_{OUT}(e)$ označuje počet vnitřních hran i-komponenty C při průchodu proti směru hodinových ručiček kolem vrcholu v od hrany e k hraně e^{CCW}_{OUT}. Pokud e je vnější hrana i-komponenty C, pak $e^{CW}_{OUT} = e^{CCW}_{OUT} = e$.*

Teď uvedeme několik pomocných tvrzení, které se nám budou hodit při důkazu hlavního tvrzení o shodnosti hranic. Z definice víme, že každý plátek pro i-stěnu f identifikovanou hranou e má levou hranici shodnou s levou hranicí plátku hrany následující e a pravou hranici plátku má shodnou s hranicí plátku pro hranu předcházející e. V následujícím tvrzení ukážeme, že existuje vnější hrana, která má levou hranici plátku shodnou s levou hranicí plátku pro f a jiná vnější hrana taková, že pravá hranice plátku pro tuto hranu je shodná s pravou hranicí plátku pro f.

Tvrzení 1. *Nechť f je i-stěna identifikovaná hranou $e = uv$, pak levá hranice plátku pro e je shodná s levou hranicí plátku pro vnější hranu e^{CW}_{OUT} a pravá hranice*

plátku pro e je shodná s pravou hranicí plátku pro vnější hranu e^{CCW}_{OUT}.

Důkaz. Nejprve dokážeme část tvrzení pro levou hranici. Důkaz provedeme indukcí podle $d = d^{CW}_{OUT}(e)$. Pro $d = 0$ tvrzení plyne z definice 10, protože $e^{CW}_{OUT} = e^N_f$. Dále předpokládejme, že tvrzení platí pro $d-1$ a dokážeme ho pro d. Nechť $e' = e^N_f$. Protože platí, že e' je následující vnitřní hrana po směru hodinových ručiček kolem u od hrany e, tak $e'^{CW}_{OUT} = e^{CW}_{OUT}$ a $d' = d^{CW}_{OUT}(e') = d-1$. Použitím indukčního předpokladu pro $d-1$ získáme, že levá hranice pro e' je shodná s levou hranicí pro e^{CW}_{OUT} a z definice 10 plyne, že levá hranice pro e je shodná s levou hranicí pro e'. Tím dostáváme, že levá hranice pro plátek hrany e je shodná s levou hranicí pro plátek vnější hrany e^{CW}_{OUT}.

Důkaz pro pravou hranici je velmi podobný, tentokrát je to důkaz indukcí podle $d = d^{CCW}_{OUT}(e)$. Pro $d = 0$ tvrzení zase plyne z definice 10, protože $e^{CCW}_{OUT} = e^P_f$. Dále předpokládejme, že tvrzení platí pro $d-1$ a dokážeme ho pro d. Nechť $e' = e^P_f$. Protože platí, že e' je následující vnitřní hrana proti směru hodinových ručiček kolem v od hrany e, tak $e'^{CCW}_{OUT} = e^{CCW}_{OUT}$ a $d' = d^{CCW}_{OUT}(e') = d-1$. Použitím indukčního předpokladu pro $d-1$ získáme, že pravá hranice pro e' je shodná s pravou hranicí pro e^{CW}_{OUT} a z definice 10 plyne, že pravá hranice pro e je shodná s pravou hranicí pro e'. Tím dostáváme, že pravá hranice pro plátek hrany e je shodná s pravou hranicí pro plátek vnější hrany e^{CCW}_{OUT}.

□

Důsledek. Nechť C je i-komponenta a e její hrana, pak levá hranice plátku pro e je shodná s levou hranicí plátku pro e^{CW}_{OUT} a pravá hranice plátku pro e je shodná s pravou hranicí plátku pro e^{CCW}_{OUT}.

Důkaz. Pokud je e vnější hrana, tak tvrzení platí z definice. Pokud je e vnitřní hrana, tak tvrzení plyne z tvrzení 1.

□

Teď dokážeme tvrzení podobné předchozímu pro i-stěnu identifikovanou vrcholem.

Tvrzení 2. *Nechť f je i-stěna identifikovaná vrcholem v, pak levá hranice plátku pro f je shodná s levou hranicí plátku pro vnější hranu v^N_f a pravá hranice plátku pro f je shodná s pravou hranicí plátku pro vnější hranu $(v^P_f)^{CCW}_{OUT}$.*

Důkaz. Důkaz pro levou hranici je zřejmý. Z definice 3 víme, že pro i-stěnu f identifikovanou vrcholem je hrana v^N_f vnější a z definice 10 víme, že levá hranice plátku pro f je shodná s levou hranicí plátku pro hranu v^N_f. Pro pravou hranici je to trochu složitější. Z definice je pravá hranice plátku pro f shodná s pravou hranicí plátku pro hranu $e = v^P_f$. Pokud je hrana e vnější hranou, tak podle definice je pravá hranice pro f shodná s pravou hranicí pro e a také $e^{CCW}_{OUT} = e$, takže tvrzení pro pravou hranici platí. Pokud e je vnitřní hranou, tak e je identifikátor nějaké i-stěny f'. Z předchozího tvrzení víme, že pravá hranice plátku pro e je shodná s pravou hranicí plátku pro vnější hranu e^{CCW}_{OUT} a z definice víme, že pravá hranice plátku pro f je shodná s pravou hranicí pro e.

□

Pokud budeme mluvit o levém vrcholu identifikátoru d, myslíme tím buď d, pokud d je vrchol, nebo u, pokud d je hrana uv. Podobně pravým vrcholem identifikátoru d myslíme buď d, pokud je d vrchol, nebo v, pokud je d hrana uv. Následuje ještě jedno pomocné tvrzení.

Tvrzení 3. *Nechť C je i-komponenta s počátečním vrcholem u a počáteční hranou $e = uv$ vnořená v $(i-1)$-stěně f s identifikátorem d. Pak pro každou vnější hranu e' různou od e_P, která není součástí levého okolí vrcholu u a zároveň hrana e'_N není součástí pravého okolí vrcholu u platí, že hrana e'_R nevede do vrcholu, který je součástí identifikátoru d.*

Důkaz. Nechť w je levý vrchol identifikátoru d a w' je pravý vrchol identifikátoru d. Pro potřeby tohoto důkazu předpokládejme, že vrcholy w a w' různé. Pokud tento předpoklad neplatí, tak si dále představujme, že když říkáme, že hrana e' vede do w, tak myslím, že vede do w a zároveň leží mezi hranami e_L a $(e_L)_{FN}$ při průchodu proti směru hodinových ručiček a podobně pokud říkáme, že hrana e' vede do w', myslíme tím, že vede do vrcholu w' a zároveň hrana e' leží mezi hranami $((e_P)_R)_{FP}$ a $(e_P)_R$ při průchodu proti směru hodinových ručiček.

Nejprve si všimneme, že pokud hrana e' je součástí levého okolí vrcholu u, tak hrana e'_P vede do vrcholu w. To nahlédneme indukcí podle vzdálenosti od počáteční hrany e. Z definice platí, že e_L vede do w. Pokud e je součástí levého okolí, tak z definice okolí platí, že e_L a e_R vedou do stejného vrcholu. Pokud e' je hrana levého okolí, taková, že $(e'_P)_R$ vede do w, tak i hrana $e'_L = (e'_P)_R$ vede do w a z definice okolí i e'_R vede do w. Ze stejného důvodu pro hranu e', která je součástí pravého okolí u platí, že e'_L vede do w'.

Pořadí vrcholů stěny f bereme v pořadí proti směru hodinových ručiček tak, že vrchol w je první a vrchol w' je poslední. Pořadí vnějších hran i-komponenty C bereme v pořadí při průchodu hran proti směru hodinových ručiček, kde e je první a e_P poslední.

Dále si všimneme, že pro každou hranu e' komponenty C platí, že hrana e'_R vede do vrcholu, který je buď shodný nebo je dále v pořadí, než vrchol do kterého vede hrana e'_L. Kdyby toto neplatilo, tak by se hrany e'_L a e'_P musely křížit. A tedy i pro sousední hrany g a g' komponenty C, $g_N = g'$, platí, že vrchol, do kterého vede hrana g'_R, je shodný nebo dále v pořadí než vrchol, do kterého vede hrana g_R a také vrchol, do kterého vede hrana g'_L, je shodný nebo dále v pořadí než vrchol do kterého vede hrana g_L.

Nechť e' je hrana, která následuje po poslední hraně levého okolí vrcholu u (nebo $e' = e$ pokud je levé okolí prázdné). Potom platí, že hrana e'_R nevede do vrcholu w (jinak by hrana e' také patřila do okolí) a tedy z předchozího pozorování ani pro žádnou hranu g, která je v pořadí po e' nemůže g_R vést do w. Stejně tak pro každou hranu g, která neleží v pravém okolí vrcholu u platí, že hrana g_L nevede do vrcholu w'.

Teď už dokončíme důkaz. Mějme hranu e', která není součástí levého okolí vrcholu u a zároveň e'_N není součástí pravého okolí u. Pak jsme ukázali, že hrana e'_R nevede do vrcholu w a zároveň hrana $(e'_N)_L = e'_R$ nevede do vrcholu w'.

□

Dále pro artikulaci v i-komponenty definujeme v-podkomponenty. Bude se jednat o podgrafy i-komponenty, které vzniknou roztržením komponenty v arti-

kulaci v. Pro každou část komponenty, které jsou spojeny vrcholem v, bude jedna v-podkomponenta.

Definice 14. *Nechť C je i-komponenta a v je vrchol, který je její artikulací. Mějme vnější hranu $e = vu$ komponenty C a hranu $e' = wv$, která první vnější hranou komponenty C mající za pravý vrchol v při průchodu vnějších hran proti směru hodinových ručiček od hrany e. Pak podgraf indukovaný vrcholy cyklu, který získáme průchodem vnějších hran i-komponenty C od hrany e k hraně e' proti směru hodinových ručiček, nazveme podkomponentou vrcholu v, neboli v-podkomponentou. Hranu e bereme jako první hranu dané v-podkomponenty a hranu e' jako poslední. Podkomponentu vrcholu v obsahující počáteční hranu komponenty C nazveme hlavní v-podkomponentou.*

Pořadí v-podkomponent je určeno pořadím i-stěn identifikovaných vrcholem v, který daná v-podkomponenta obsahuje, kde hlavní v-podkomponentu do pořadí nezařazujeme, pokud v není počáteční vrchol i-komponenty C.

Všimněme si, že každá v-podkomponenta obsahuje pouze jednu vnější hranu mající v za levý vrchol a pouze jednu vnější hranu mající v za pravý vrchol, tedy vrchol v netvoří artikulaci ve v-podkomponentě.

Teď dokažme hlavní tvrzení o shodnosti hranic.

Tvrzení 4.

1. *Nechť C je i-komponenta a $e = uv$, $e' = vw$ jsou její dvě sousední vnější hrany takové, že e' není počáteční hranou komponenty C. Pak pravá hranice plátku pro hranu e je shodná s levou hranicí plátku pro hranu e'.*

2. *Nechť C je i-komponenta a v její vrchol takový, že existují alespoň dvě i-stěny identifikované tímto vrcholem. Nechť f a f' jsou dvě po sobě jdoucí i-stěny identifikované vrcholem v. Pak pravá hranice plátku pro f je shodná s levou hranicí plátku pro f'.*

3. *Nechť f je i-stěna s identifikátorem d a nechť $e = uv$ je nějaká hrana f taková, že vrchol v není obsažen v identifikátoru d a zároveň existuje plátek pro vrchol v, pak pravá hranice plátku pro e je shodná s levou hranicí plátku pro v.*

4. *Nechť f je i-stěna s identifikátorem d a nechť $e = uv$, $e' = vw$ jsou dvě sousední hrany i-stěny f takové, že vrchol v není obsažen v identifikátoru d. Pak pravá hranice rozšířeného plátku pro hranu e je shodná s levou hranicí rozšířeného plátku pro e'.*

Důkaz. Důkaz provedeme indukcí následujícím způsobem. Nejprve dokážeme část 1 pro $i = 1$. Dále budeme předpokládat, že část 1 platí pro i a dokážeme pro i i zbývající části tvrzení. A nakonec budeme předpokládat, že platí část 4 pro $i-1$ a dokážeme část 1 pro i.

Důkaz části 1 pro $i = 1$ je jednoduchý, plyne přímo z definice hranice plátku. Teď předpokládejme, že část 1 platí pro i a dokážeme, že pro i platí i část 2. Mějme tedy vrchol v a dvě po sobě jdoucí i-stěny f a f' identifikované vrcholem v a chceme ukázat, že pravá hranice plátku pro f je shodná s levou hranicí plátku pro f'. Z tvrzení 2 plyne, že pravá hranice plátku pro f je shodná s pravou hranicí

plátku pro $e = (v_f^P)_{OUT}^{CCW} = uv$ a levá hranice plátku pro f' je shodná s levou hranicí plátku pro $e' = v_{f'}^N = vw$. Hrana e je poslední vnější hranou v-podkomponenty C_f obsahující i-stěnu f a hrana e' je první vnější hranou v-podkomponenty $C_{f'}$, obsahující i-stěnu f' a tyto dvě v-podkomponenty jsou sousední. Tedy hrany e a e' jsou sousední vnější hrany i-komponenty C. Pokud vrchol v není počáteční vrchol i-komponenty C, tak hrana e' nemůže být počáteční hranou C. Pokud v je počáteční vrchol i-komponenty C, tak počáteční hrana C je součástí první i-stěny s identifikátorem v a protože i-stěna f' není první i-stěnou s identifikátorem v, tak e' není počáteční hrana C. Takže e a e' jsou dvě sousední vnější hrany a hrana e' není počáteční hranou i-komponenty C, tedy díky části 1 pro i část 2 pro i platí.

Dále předpokládejme, že část 1 platí pro i a chceme dokázat část 3 pro i. Mějme tedy i-stěnu f s identifikátorem d obsaženou v i-komponentě C. Nechť $e = uv$ je hrana i-stěny f taková, že vrchol v není obsažen v identifikátoru d a zároveň existuje plátek pro vrchol v. Chceme ukázat, že pravá hranice plátku pro hranu e je shodná s levou hranicí plátku pro vrchol v. Mějme hranu $e' = e_{OUT}^{CCW} = u'v$, i-stěnu f', která je první i-stěna identifikovaná vrcholem v, a hranu $g = v_{f'}^N = vw$. Z tvrzení 1 víme, že pravá hranice plátku pro e je shodná s pravou hranicí plátku pro e' a z definice je levá hranice plátku pro g shodná s levou hranicí plátku pro artikulaci v. Protože hrana e' je vnější hranou hlavní v-podkomponenty mající vrchol v za pravý vrchol a hrana g je první vnější hrana první v-podkomponenty, tak hrany e' a g jsou sousední vnější hrany komponenty C. Vrchol v není počáteční vrchol i-komponenty C, protože pokud by stěna f počáteční vrchol obsahovala, tak by byl součástí identifikátoru d, my ale předpokládáme, že v součástí identifikátoru d není. Proto tedy e' není počáteční hranou i-komponenty C a použitím části 1 pro i dostaneme, že část 3 pro i platí.

Teď se podívejme na část 4. Předpokládejme, že část 1 platí pro i a ukažme, že část 4 platí pro i. Mějme i-stěnu f s identifikátorem d obsaženou v i-komponentě C. Nechť $e = uv$ a $e' = vw$ jsou dvě sousední hrany i-stěny f. Chceme ukázat, že pravá hranice rozšířeného plátku pro e je shodná s levou hranicí rozšířeného plátku pro e'. Nejprve předpokládejme, že vrchol v není artikulací. Protože v není artikulací, tak z definice je pravá hranice rozšířeného plátku pro e shodná s pravou hranicí plátku pro e. Nechť $g = e_{OUT}^{CCW} = u'v$ a $g' = e'^{CW}_{OUT} = vw'$. Z tvrzení 1 plyne, že pravá hranice plátku pro e je shodná s pravou hranicí plátku pro g a levá hranice plátku pro e' je shodná s levou hranicí plátku pro g'. Protože hrany g a g' jsou vnější a protože vrchol v není artikulací, tak existuje pouze jedna vnější hrana mající pravý vrchol v a pouze jedna vnější hrana mající levý vrchol v, tak hrany g a g' musejí být sousední vnější hrany. Vrchol v není počáteční vrchol i-komponenty C, protože pokud by stěna f počáteční vrchol obsahovala, tak by byl součástí identifikátoru d, my ale předpokládáme, že v součástí identifikátoru d není. Proto tedy e' není počáteční hrana i-komponenty C a použitím části 1 pro i dostaneme, že část 4 platí. Teď předpokládejme, že vrchol v je artikulací. Nechť f' je poslední i-stěna identifikovaná vrcholem v a nechť $g = (v_{f'}^P)_{OUT}^{CCW} = u'v$ a $g' = e'^{CW}_{OUT} = vw'$. Hrana g je poslední vnější hranou poslední v-podkomponenty a hrana g' je vnější hrana hlavní v-podkomponenty mající levý vrchol v. Proto g a g' jsou sousední vnější hrany i-komponenty C a hrana e' není počáteční hranou i-komponenty C ze stejného důvodu, jako když jsme předpokládali, že v není artikulací, tedy část 4 platí pro i.

Nakonec předpokládejme, že část 4 platí pro $i-1$ a dokážeme, že část 1 platí pro i. Mějme i-komponentu C s počátečním vrcholem q, která je vnořena v $(i-1)$-stěně f. Nechť $e = uv$ a $e' = vw$ jsou dvě sousední vnější hrany i-komponenty C takové, že e' není počáteční hrana C. Chceme ukázat, že pravá hranice plátku pro e je shodná s levou hranicí plátku pro e'. Z definice hranice plyne, že první vrchol pravé hranice plátku pro e je shodný s prvním vrcholem levé hranice plátku pro e'. Zbývá dokázat, že i zbytek hranice je shodný. Rozlišíme několik situací. Nejprve předpokládejme, že hrana e je součástí levého okolí vrcholu q. Protože hrany e, e_L a e_R tvoří trojúhelník, tak hrany e_L a e_R vedou do stejného vrcholu a tedy hrany $(e_L)_{FN}$ a $(e_R)_{FN}$ jsou shodné. Z definice plyne, že $e_R = e'_L$, protože e' není počáteční hrana. Z toho plyne, že hrany $g = (e_L)_{FN}$ a $g' = (e'_L)_{FN}$ jsou shodné a z definice platí, že pravá hranice plátku pro e pokračuje levou hranicí rozšířeného plátku pro g a levá hranice plátku pro e' pokračuje levou hranicí rozšířeného plátku pro g' (nezávisle na tom, jestli je hrana e' také součástí levého okolí q, nebo ne), tak část 1 v tomto případě platí. Dále předpokládejme, že hrana e' je součástí pravého okolí vrcholu q. Protože hrany e', e'_L a e'_R tvoří trojúhelník, tak hrany e'_L a e'_R vedou do stejného vrcholu a tedy hrany $(e'_L)_{FP}$ a $(e'_R)_{FP}$ jsou shodné. Z definice plyne, že $e_R = e'_L$, protože e' není počáteční hrana. Z toho plyne, že hrany $g = (e_R)_{FP}$ a $g' = (e'_R)_{FP}$ jsou shodné a z definice platí, že pravá hranice plátku pro e pokračuje pravou hranicí rozšířeného plátku pro g (nezávisle na tom, jestli je hrana e také součástí pravého okolí q, nebo ne) a levá hranice plátku pro e' pokračuje pravou hranicí rozšířeného plátku pro g', tak část 1 v tomto případě také platí. Nakonec předpokládejme, že nenastala ani jedna z předchozích možností, tedy e není součástí levého okolí a ani e' není součástí pravého okolí. Protože z definice plyne, že $e_R = e'_L$, tak hrany $g = (e_R)_{FP} = xw$ a $g' = (e_L)_{FN} = wy$ jsou sousedními hranami $(i-1)$-stěny f. Z části 4 pro $i-1$ plyne, že pravá hranice rozšířeného plátku pro g je shodná s levou hranicí rozšířeného plátku pro g', pokud vrchol w není součástí identifikátoru d. Protože hrana e není součástí levého okolí vrcholu q a hrana $e' = e_N$ není součástí pravého okolí vrcholu q, tak z tvrzení 3 plyne, že hrana e_R nevede do vrcholu, který je součástí identifikátoru d a tedy vrchol w není obsažen v identifikátoru d.

□

Teď se podívejme, co jsme vlastně dokázali. Část 2 nám říká, že dva sousední plátky identifikované stejným vrcholem v mají společnou hranici, tedy můžeme zkonstruovat tabulku plátku pro vrchol v. Část 3 říká, že můžeme vytvořit tabulku pro rozšířený plátek hrany $e = uv$ spojením tabulek pro hranu e a vrchol v. Nakonec část 4 říká, že můžeme spojovat tabulky pro rozšířené plátky sousedních hran i-stěny, což se nám bude hodit při vytváření plátku pro tuto i-stěnu. Podrobnosti o tvoření tabulek uvedeme v následující části této kapitoly.

Ještě nám zbývá ukázat, jak vyřešit situaci, kdy i-komponenta není souvislá. Mějme tedy $(i-1)$-stěnu f, která má vnořenou i-komponentu C, která není souvislá. To vyřešíme tak, že stěnu f rozdělíme hranami do více stěn, tak aby každá komponenta souvislosti i-komponenty C padla do jiné nové $(i-1)$-stěny. Tyto nově přidané hrany nazveme nepravé vnitřní. Může nastat i situace, kde tímto přidáváním hran můžeme přidat i smyčku, což ale algoritmu nevadí a dokáže si s tím poradit.

Tímto máme definované všechny potřebné pojmy a můžeme se pustit do po-

pisu algoritmu. Ale ještě než to uděláme, dokážeme následující tvrzení, které říká, že pokud má hranice plátku na dané úrovni levý a pravý vrchol rozdílný a na následující nižší úrovni už jsou vrcholy shodné, tak už jsou shodné pro všechny nižší úrovně. Toto tvrzení se nám bude hodit v části o implementaci.

Tvrzení 5. *Nechť S je plátek, který má pro nějakou úroveň levý a pravý vrchol hranice rozdílný a pro nějakou nižší úroveň je levý vrchol hranice shodný s pravým. Označme k maximální číslo takové, že levý vrchol hranice úrovně k a pravý vrchol hranice úrovně k plátku S jsou různé. Nechť $j < k$ je maximální číslo takové, že levý vrchol hranice úrovně k a pravý vrchol hranice úrovně k plátku S jsou shodné. Pak pro všechny úrovně $\ell < j$, je levý vrchol hranice úrovně ℓ shodný s pravým vrcholem hranice úrovně ℓ plátku S.*

Důkaz. Nechť u je levý vrchol úrovně $j+1$, v je pravý vrchol úrovně $j+1$ a w je společný vrchol úrovně j hranice plátku S. Nechť e je první vnější hranou úrovně $j+1$, jejíž plátek je součástí plátku S a nechť e' je poslední vnější hranou úrovně $j+1$, jejíž plátek je součástí plátku S. Označme C $(j+1)$-komponentu, která hrany e a e' obsahuje, a nechť q je počáteční vrchol této komponenty. Pokud hrana e' je součástí levého okolí vrcholu q, tak je jeho součástí i hrana e a z definice levá hranice plátku pro e pokračuje levou hranicí rozšířeného plátku pro $(e_L)_{FN}$ a pravá hranice plátku pro e' je shodná s levou hranicí rozšířeného plátku pro $(e'_L)_{FN}$, a protože hrany $(e_L)_{FN}$ a $(e'_L)_{FN}$ jsou shodné, tak i hranice je shodná. Podobně pokud e je součástí pravého okolí. Pokud e není součástí pravého okolí a zároveň e' není součástí levého okolí, pak levá hranice plátku pro e pokračuje levou hranicí rozšířeného plátku pro $(e_L)_{FN} = wx$ a pravá hranice plátku pro e' pokračuje pravou hranicí rozšířeného plátku pro $(e'_R)_{FP} = yw$. Tedy hrany yw a wx jsou sousední hrany $(i-1)$-stěny, kde w není součástí jejího identifikátoru (jinak by hrany e a e' ležely v nějakém okolí vrcholu q) a tedy z části 4 tvrzení 4 platí, že pravá hranice rozšířeného plátku pro yw je shodná s levou hranicí rozšířeného plátku pro wx a tím je tvrzení dokázáno.

□

1.2 Detailní popis algoritmu

V předchozí části jsme definovali potřebné pojmy, z nichž nejdůležitější je plátek, a v této části popíšeme přesný algoritmus. Pro každý plátek budeme tvořit tabulku, která nám bude říkat velikost největší nezávislé množiny v daném plátku, v závislosti na tom, jaké vrcholy z hranice se nachází v nezávislé množině. Pro každou možnost výskytu vrcholů na hranici budeme mít v tabulce jednu položku. Mějme nějaký plátek S úrovně i a nechť $\alpha, \beta \in \{0,1\}^i$ jsou nulajedničkové vektory, pak $S[\alpha, \beta]$ označuje položku tabulky pro plátek S, kde obsažené vrcholy na levé hranici jsou shodné s vektorem α a na pravé hranici shodné s vektorem β. Když říkáme, že obsažené vrcholy hranice jsou shodné s vektorem, myslíme tím, že když je na dané pozici vektoru 0, tak daný vrchol hranice není v nezávislé množině a naopak 1 znamená, že vrchol do nezávislé množiny patří a vrcholy hranice bereme v pořadí od nejvyšší úrovně, tedy první prvek vektoru určuje výskyt vrcholu nejvyšší úrovně. Když v této části řekneme, že jsou dva vrcholy

spojeny hranou, myslíme tím, že jsou spojeny hranou v grafu G, tedy v grafu bez přidaných triangulačních hran.

Hodnoty tabulky budou tvořit přirozená čísla (včetně nuly) a speciální nedefinovaná hodnota označená $\perp$, která bude označovat, že daná kombinace není validní, tedy danou kombinací bychom do nezávislé množiny zařadili oba vrcholy nějaké hrany nebo obě hranice sdílí stejný vrchol a v jedné v nich vrchol patří do nezávislé množiny a v druhé ne. Protože budeme s hodnotami tabulek provádět aritmetické operace a porovnávání, musíme dodefinovat, jak se budou tyto operace chovat vzhledem k hodnotě $\perp$. Výsledkem všech aritmetických operací, kde alespoň jedna z hodnot je $\perp$, bude $\perp$ a při porovnávání platí, že $\perp$ je menší než jakékoliv přirozené číslo.

Algoritmus funguje tak, že z tabulek pro menší plátky bude postupně tvořit tabulky pro větší plátky, stejným způsobem, jako když jsme v definici 12 definovali plátek pomocí jiných plátků. Začneme tabulkami vnějších hran úrovně jedna, protože je můžeme sestrojit přímo. Tabulka plátku pro vnější hranu e úrovně jedna, který obsahuje pouze hranu e, bude obsahovat čtyři hodnoty podle toho, které hraniční vrcholy jsou obsažené v nezávislé množině. Pro položky, kde je obsažen pouze jeden z vrcholů hrany, bude tabulka obsahovat hodnotu 1, pro položku, kde ani jeden z vrcholů není součástí nezávislé množiny, bude obsahovat hodnotu 0 a pro položku, kde jsou oba vrcholy součástí nezávislé množiny, bude obsahovat hodnotu $\perp$. Tuto operaci, která bude vytvářet tabulku pro vnější hranu e, budeme označovat $create(e)$. Jak bude fungovat pro vnější hrany vyšších úrovní, popíšeme později.

Algoritmus se bude skládat z procedur $process\_edge(e)$, $process\_vertex(v,e)$ a $process\_cutpoint(v)$. První dvě procedury budou tvořit tabulky plátků pro i-stěny identifikované hranou nebo vrcholem a poslední z nich bude tvořit tabulku pro plátek vrcholu v. Před spuštěním algoritmu nejprve zvolíme počáteční vrchol v a hranu e vstupního grafu G. Spustíme algoritmus na graf G a začneme tak, že projdeme graf, najdeme jednotlivé i-komponenty, přidáme triangulační hrany, přidáme případné falešné vnitřní hrany k zajištění souvislosti i-komponent a najdeme dělící hrany. Toto vše zvládneme průchodem grafu v lineárním čase. V dalším průchodu budeme konstruovat tabulky pro jednotlivé plátky a to tak, že zavoláme proceduru $process\_cutpoint(v)$ na počáteční vrchol. Ta nám vrátí tabulku T se dvěma hodnotami, jednu pro případ, že se vrchol v v nezávislé množině vyskytuje a druhou pro případ, kdy se v ní vrchol v nevyskytuje. Protože plátek počátečního vrcholu obsahuje celý graf G, tak maximální hodnota v tabulce T nám udává velikost největší nezávislé množiny grafu G.

Teď popíšeme jednotlivé procedury a začneme od $process\_cutpoint(v)$, protože je v algoritmu použita jako první. Tato procedura vytvoří tabulku plátku pro každou i-stěnu f identifikovanou vrcholem v voláním procedury $process\_vertex(v,e)$, kde e je první vnější hrana i-stěny f, tedy ta obsahující v jako levý vrchol. Tímto získáme tabulky pro všechny i-stěny identifikované vrcholem v. Pokud jich je více, tak použijeme operaci $merge$, která spojuje tabulky sousedních plátků, a všechny tyto získané tabulky plátků spojíme v pořadí definovaném v definici 5. Důvod, proč voláme tuto proceduru na počáteční vrchol grafu G, i když se nemusí jednat o artikulaci, je takový, že toto nemusíme v algoritmu ověřovat a zavoláním této procedury získáme správný výsledek, protože pokud počáteční vrchol není artikulací, tak tato procedura pouze zavolá $process\_vertex(v,e)$ na jedinou 1-

stěnu identifikovanou tímto počátečním vrcholem a vrátí výslednou tabulku. Ze stejného důvodu budeme tuto proceduru volat i na počáteční vrcholy vnořených i-komponent, jak uvidíme později.

Dále popišme proceduru *process_vertex*(v,e), která bude zpracovávat i-stěnu f identifikovanou vrcholem v, která obsahuje vnější hranu e, a vytvoří pro ni tabulku. Důvod, proč při volání procedury předáváme i hranu e, je nejednoznačnost identifikátorů a tedy pro jednoznačnou identifikaci i-stěny potřebujeme znát i nějakou její vnější hranu. Procedura nejprve projde všechny hrany $e' = u'v'$ od v_f^N do v_f^P a vytvoří tabulky pro rozšířené plátky těchto hran. To provede následovně: Pokud e' je vnější hrana, tak na ni zavolá operaci $create(e')$, která vytvoří tabulku pro hranu e'. Pokud se jedná o vnitřní nebo nepravou vnitřní hranu, tak na ni zavolá proceduru *process_edge*(e'), která nám také vytvoří tabulku pro tuto hranu. Pokud vrchol v' není shodný s vrcholem v a zároveň je vrchol v' identifikátorem nějaké stěny, tak na něj zavolá proceduru *process_cutpoint*(v') a výslednou tabulku spojí operací *merge* s tabulkou pro hranu e'. Tím získá tabulku rozšířeného plátku pro e'. Pokud stěna f neobsahuje vnořenou komponentu, tak spojí všechny vytvořené tabulky rozšířených plátků hran v pořadí od v_f^N do v_f^P proti směru hodinových ručiček a výslednou tabulku označí T. Nakonec na T zavolá operaci $adjust(T)$, která nám dokončí tabulku pro stěnu f a tu také procedura *process_vertex*(v,e) vrátí. Pokud stěna f obsahuje vnořenou komponentu, tak nechť w je počáteční vrchol této komponenty. Pak zavolá proceduru *process_cutpoint*(w) na počáteční vrchol a získá tabulku T. Tabulka získaná operacemi $adjust(contract(T))$ je tabulka pro stěnu f, kterou procedura *process_vertex*(v,e) vrátí.

Zbývá nám procedura *process_edge*(e), která pro vnitřní nebo nepravou vnitřní hranu e zpracuje i-stěnu f identifikovanou hranou e a vytvoří pro ni tabulku. Bude pracovat velmi podobně jako předchozí procedura *process_vertex*(v,e). Nejprve projde všechny hrany $e' = u'v'$ od e_f^N do e_f^P a vytvoří tabulky rozšířených plátků pro tyto hrany stejným způsobem jako v předchozí proceduře. Pokud stěna f neobsahuje vnořenou komponentu, tak spojí všechny vytvořené tabulky rozšířených plátků hran v pořadí od e_f^N do e_f^P proti směru hodinových ručiček a výslednou tabulku označí T. Nakonec na T zavolá operaci $adjust(T)$, která nám dokončí tabulku pro stěnu f a tu také procedura *process_edge*(e) vrátí. Pokud stěna f obsahuje vnořenou komponentu, tak nechť w je počáteční vrchol této komponenty. Pak zavolá proceduru *process_cutpoint*(w) na počáteční vrchol a získáme tabulku T. Tabulka získaná operacemi $adjust(contract(T))$ je tabulka pro stěnu f, kterou procedura *process_edge*(e) vrátí.

Tímto máme algoritmus popsaný, ale ještě nám zbývá popsat použité operace. Začneme s operací *merge*(L,R), která spojí tabulky dvou plátků, které sdílí společnou hranici. Mějme plátky L a R společně s tabulkami takové, že pravá hranice plátku L je shodná s levou hranicí plátku P a budeme chtít vytvořit tabulku pro plátek M, který vznikne sloučením plátků L a R. Hodnoty nové tabulky spočítáme vzorcem

$$M[\alpha,\beta] = \max_{\gamma} \left(L[\alpha,\gamma] + R[\gamma,\beta] - w(\gamma) \right),$$

kde $w(\gamma)$ značí Hammingovu váhu vektoru γ, tedy počet jedniček v daném vektoru. Výpočet tedy probíhá tak, že projdeme všechny možnosti, jak můžeme vybrat vrcholy na společné hranici a vezmeme tu možnost, která nám dá maximální

nezávislou množinu. Důvod, proč odečítáme ještě Hammingovu váhu vektoru, je takový, že kdybychom to neudělali, tak součtem hodnot v tabulkách pro L a R započítáme vrcholy na společné hranici dvakrát, musíme je tedy jednou odečíst.

Dále popíšeme operaci $create(e)$, která konstruuje tabulky pro vnější hrany. Pro vnější hrany úrovně jedna jsme už operaci popsali, proto se teď podíváme, jak tato operace pracuje pro vnější hranu vyšší úrovně. K jejímu popisu budeme ještě potřebovat operaci $extend(v,S)$, která vezme tabulku pro plátek S úrovně $i-1$ a rozšíří ji na tabulku pro plátek S' úrovně i, který vznikne tak, že k oběma hranicím přidáme vrchol v. Plátek S' není plátek pro hranu, stěnu nebo vrchol, jak bylo definováno dříve, ale má stejné vlastnosti jako plátek a budeme na něj používat stejné operace, proto pro jednoduchost pro něj budeme také používat pojem plátek. Pokud vrchol v nebude v nezávislé množině, pak hodnoty tabulek pro plátky S a S' jsou shodné, tedy $S'[0\alpha,0\beta] = S[\alpha,\beta]$, $\alpha,\beta \in \{0,1\}^{i-1}$. Nechť l je vrcholem nejvyšší úrovně levé hranice S a r vrcholem nejvyšší úrovně pravé hranice S. Pokud vrchol v bude v nezávislé množině, tak záleží na tom, jestli je vrchol v spojen hranou s l nebo r. Pokud není spojen ani s jedním z vrcholů, tak platí $S'[1\alpha,1\beta] = S[\alpha,\beta] + 1$, $\alpha,\beta \in \{0,1\}^{i-1}$. Pokud je spojený s vrcholem l, tak platí $S'[11\alpha,1\beta] = \perp$, $S'[10\alpha,1\beta] = S[0\alpha,\beta] + 1$, $\alpha \in \{0,1\}^{i-2}$, $\beta \in \{0,1\}^{i-1}$, obdobně pro v spojený pouze s r. Pokud je spojený s oběma vrcholy l i r, tak platí $S'[11\alpha,11\beta] = \perp$, $S'[10\alpha,10\beta] = S[\alpha,\beta] + 1$, $\alpha,\beta \in \{0,1\}^{i-2}$.

Mějme i-komponentu C vnořenou v $(i-1)$-stěně f s identifikátorem d a konstruujeme tabulku pro vnější hranu $e = uv$ komponenty C. Nejprve vezmeme vrchol m stěny f, který je hranou spojený s oběma vrcholy u i v v grafu s přidanými triangulačními hranami. Plátek pro hranu e rozdělíme na tři části, které budeme nazývat levá, střední a pravá. Nejdříve zpracujeme levou část. Pokud hrana e_L vede do vrcholu m, tak je levá část prázdná, jinak pro každou hranu $e' = u'v'$ od $(e_L)_{FN}$ do $(um)_{FP}$ proti směru hodinových ručiček kolem i-stěny f provedeme následující: Nechť T je tabulka rozšířeného plátku pro hranu e', použijeme na ní operaci $extend(u,T)$ a všechny takto vzniklé tabulky spojíme a označíme L. Podobně vytvoříme pravou část. Pokud hrana e_R vede do vrcholu m, tak je pravá část prázdná, jinak pro každou hranu $e' = u'v'$ od $(vm)_{FN}$ do $(e_R)_{FP}$ proti směru hodinových ručiček kolem i-stěny f uděláme následující: Nechť T je tabulka rozšířeného plátku pro hranu e', použijeme na ni operaci $extend(v,T)$ a všechny takto vzniklé tabulky spojíme a označíme P. Zbývá nám prostřední část, kterou vytvoříme tak, že vezmeme plátek M tvořený hranou e, vrcholem m a zbytkem hranice shodným s levou hranicí plátku pro hranu $(um)_{FN}$, pokud hrana e není součástí pravého okolí počátečního vrcholu i-komponenty C, jinak bude zbytek hranice tvořen pravou hranicí plátku pro hranu $(um)_{FP}$. Vytvoříme tabulku pro tento plátek M. Jak tuto tabulku efektivně zkonstruovat (v konstantním čase) ukážeme v další části věnované implementaci, protože je závislá na zvolené implementaci tabulek. Teď už nám stačí spojit tabulky L, M a R pomocí operace $merge$ a máme tabulku pro hranu e.

Operaci $contract(S)$ budeme používat na zkrácení hranice, která bude definována pro situaci, kdy je první vrchol levé a pravé hranice shodný. Tento případ nastane u plátku pro počáteční vrchol i-komponenty, což je situace, ve které tuto operaci použijeme. Nechť C je i-komponenta s počátečním vrcholem v vnořená v $(i-1)$-stěně f a nechť S je plátek pro počáteční vrchol komponenty C. Chceme z tabulky úrovně i plátku S vytvořit tabulku T úrovně $i-1$

tak, že zkrátíme hranici S o vrchol v. To provedeme jednoduše tak, že vezmeme maximum z velikostí nezávislých množin, kde se v vyskytuje a kde ne, tedy $T[\alpha,\beta] = \max(S[0\alpha,0\beta], S[1\alpha,1\beta])$.

Operaci $adjust(T)$ budeme používat jako poslední krok pro dokončení tabulky pro i-stěnu f. Nechť T je tabulka vzniklá buď sloučením tabulek pro rozšířené plátky hran f (pokud neobsahuje vnořenou komponentu) nebo tabulka vzniklá operací *contract* na tabulku počátečního vrcholu vnořené komponenty. Operace pracuje rozdílně podle toho, čím je stěna identifikovaná. Pro stěnu f identifikovanou vnitřní hranou uv musíme v tabulce zakázat situaci, kdy jsou oba vrcholy zároveň v nezávislé množině. Protože u tvoří první vrchol levé hranice a v první vrchol pravé hranice, tak nastavíme $T[1\alpha,1\beta] := \perp$, $\alpha,\beta \in \{0,1\}^{i-1}$. Pokud je ale i-stěna f identifikovaná nepravou vnitřní hranou, tak operace nic nedělá. Pokud je stěna f identifikována vrcholem v, tak první vrchol levé i pravé hranice tvoří v, musíme tedy zakázat situaci, kdy nezávislá množina obsahuje první vrchol levé hranice a neobsahuje první vrchol pravé hranice a naopak. Nastavíme tedy $T[1\alpha,0\beta] := \perp$, $T[0\alpha,1\beta] := \perp$ $\alpha,\beta \in \{0,1\}^{i-1}$. Pak tabulka T tvoří tabulku pro i-stěnu f.

Tímto máme algoritmus popsaný. Podívejme se ještě na jeho časovou složitost. Nechť n je počet vrcholů grafu G a k je maximální úroveň vrcholu v grafu G. První fázi algoritmu zvládneme v lineárním čase v n. Protože graf G je rovinný, tak počet hran tohoto grafu je $O(n)$. Nejprve odhadněme počet plátků, se kterými algoritmus pracuje. Máme plátek pro každou vnější hranu i-komponenty a také plátek pro každou vnitřní hranu i-komponenty, který reprezentuje i-stěnu. Těchto plátků je tedy maximálně $O(n)$. Zbývají ještě plátky pro i-stěny identifikované vrcholem. Každá taková stěna ale obsahuje alespoň jednu vnější hranu, tedy jich může být také maximálně $O(n)$. Celkový počet plátků je proto omezen $O(n)$. Každé volání operace *merge* potřebuje čas $O(2^{3k}) = O(8^k)$ a každé volání operací *extend*, *contract* a *adjust* potřebuje čas $O(2^{2k}) = O(4^k)$. Zbývá operace *create*. Časová složitost *create*, když do ní nezapočítáváme operace *extend* a *merge* (to započítáme hranám předchozí úrovně, které rozšiřujeme), je konstantní pro naši zvolenou implementaci tabulek, jak uvidíme v následující části. Každou z jmenovaných operací použijeme na každý plátek maximálně jednou, tedy časová složitost na jeden plátek je $O(8^k)$. Protože máme $O(n)$ plátků, tak celková časová složitost algoritmu je $O(8^k n)$

Nakonec se podívejme na prostorovou složitost. Největší část paměti zaberou samozřejmě tabulky. Každá tabulka bude mít velikost exponenciální s délkou hranice, tedy $O(2^{2k}) = O(4^k)$. Protože máme $O(n)$ plátků, tak celková paměťová složitost algoritmu je $O(4^k n)$.

1.3 Implementace

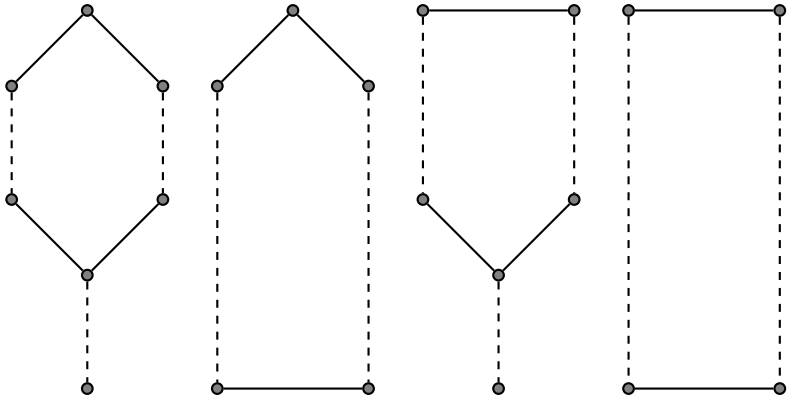
V této části se budeme zabývat implementací algoritmu, který jsme popsali v minulých částech. Algoritmus byl implementován v jazyce C. Nejprve popíšeme reprezentaci grafu a tabulek v programu. Pro reprezentaci používáme struktury definované v souboru `graph.h`, konkrétně struktury `vertex_t`, `edge_t` a `table_t`.

Začněme strukturou vrcholu, tedy `vertex_t`, a popíšeme její položky. První je položka `incident`, která nám dává odkaz na nějakou hranu incidentní s daným vrcholem. Další položka je `level`, která nám říká úroveň vrcholu. Následuje

položka `index`, která identifikuje vrchol, tedy obsahuje číslo vrcholu. Tato položka není pro funkci algoritmu nutná, můžeme ji využít pro testovací výpisy. Dále máme položku `processed`. Jedná se o pomocnou proměnou, kterou využíváme při průchodu grafu k tomu, abychom věděli, jestli už jsme daný vrchol grafu zpracovali. Předposlední položkou je `cutpoint`, kterou používáme jako čítač průchodů daným vrcholem při procházení vnější hranice dané i-komponenty. Tento údaj použijeme k tomu, abychom pak při konstrukci tabulek věděli, jestli se jedná o artikulaci (což nastane při hodnotě alespoň 2), a jestli tedy musíme zpracovávat i i-stěny, které jsou za tento vrchol připojené. Poslední položka struktury je `table`, kam ukládáme tabulku pro tento vrchol. Tuto položku využijeme pouze pokud se jedná o artikulaci nebo počáteční vrchol.

Dále popíšeme strukturu pro hranu, tedy strukturu `edge_t`. Tato struktura popisuje orientovanou hranu. Každou neorientovanou hranu grafu reprezentujeme dvěma orientovanými hranami, tedy dvěma instancemi této struktury. První položka struktury je `v`, která reprezentuje odkaz na vrchol, do kterého hrana míří. Další položkou je `type`, která nám určuje typ hrany. Typy hran jsme definovali v části s definicemi, zde pouze popíšeme jednotlivé identifikátory v programu. Typ `E_UNDEFINED` použijeme u hrany, u které jsme zatím její typ neurčili. Typ `OUTSIDE` označuje vnější hranu, `OUTSIDE_BRIDGE` označuje vnější hranu, která je zároveň i mostem. Typ `INSIDE` označuje vnitřní hranu a typ `INSIDE_FAKE` označuje nepravou vnitřní hranu, kterou jsme přidali k zajištění souvislosti i-komponent. Typ `INTER_LEVEL` označuje meziúrovňovou hranu a typ `TRIANGULATION` hranu triangulační. Dalšími položkami jsou odkazy na jiné hrany, některé z nich využijeme pouze pro některé typy hran. První odkaz je `op`, který odkazuje na hranu v opačné orientaci. Dalšími odkazy jsou `cw` a `ccw`, `cw` odkazuje na následující hranu po směru hodinových ručiček kolem počátečního vrcholu hrany a odkaz `ccw` odkazuje na následující hranu proti směru hodinových ručiček. Dále následují odkazy, které použijeme pouze pro vnější hrany i-komponenty, meziúrovňové a triangulační. Označuje-li e vnější hranu i-komponenty, pak odkaz `bor_next` odkazuje na hranu e_N a `bor_prev` odkazuje na hranu e_P. Označuje-li $e = uv$ meziúrovňovou nebo triangulační hranu ve směru od nižší úrovně k vyšší, pak odkaz `bor_next` odkazuje na první vnější hranu vnořené komponenty získanou obcházením hran proti směru hodinových ručiček kolem vrcholu v od hrany vu a `bor_prev` odkazuje na první vnější hranu ale po směru hodinových ručiček. Další odkazy použijeme pro všechny hrany, ale jejich význam bude rozdílný v závislosti na typu. Je-li e hrana i-stěny f, pak `face_next` odkazuje na hranu e_f^N a `face_prev` odkazuje na e_f^P. Je-li e meziúrovňová nebo triangulační hrana ve směru od vyšší k nižší vrstvě, pak `face_next` odkazuje na hranu e_{FN} a `face_prev` odkazuje na e_{FP}. Zbývají nám poslední dva odkazy které použijeme pouze pro hranu komponenty. Odkaz `div_point_left` odkazuje na levou dělící hranu a `div_point_right` odkazuje na pravou dělící hranu. Poslední položkou struktury je `table`, která obsahuje tabulku pro tuto hranu, pokud taková existuje.

Zbývá nám struktura `table_t`, která popisuje tabulku plátku. První položka této struktury je `type`, tedy typ tabulky. Budeme rozlišovat dva typy tabulek a to podle toho, jaké má hranice vrcholy na nejvyšší úrovni. Pro tabulku typu 0 je první vrchol levé hranice shodný s prvním vrcholem pravé hranice, což znamená, že se jedná o tabulku plátku pro vrchol. Naopak tabulka typu 1 má první vrcholy levé a pravé hranice rozdílné, takže se jedná o tabulku plátku pro hranu,



Obrázek 1.4: Druhy plátků podle hranice označené 1 až 4 zleva doprava

nebo tabulku pro sjednocení několika plátků, z nichž alespoň jeden je plátkem pro hranu. Další položka struktury je `size_b`, která udává velikost tabulky v bitech, neboli délku indexu do tabulky. Následuje položka `dif_levs`, která nám říká, kolik úrovní hranice reprezentujeme tak, že levý a pravý vrchol dané úrovně jsou různé, kde nezapočítáváme nejvyšší úroveň. Toto se nemusí shodovat se skutečným počtem úrovní s rozdílnými vrcholy, jak popíšeme dále. Poslední a nejdůležitější položka je `table`, která obsahuje položky tabulky. Jedná se o pole hodnot indexované bitovým řetězcem délky `size_b`.

Konkrétní implementace procházení grafu není tak zajímavá, detaily lze dohledat ve zdrojovém kódu v souboru `graph.c`. Co zde ale popíšeme, je implementace tabulek. Jak už jsme popsali v minulém odstavci, tabulka je reprezentována strukturou `table_t`. V této reprezentaci budeme využívat výsledku z tvrzení 5, které nám říká, že kdykoliv v nějaké úrovní hranice plátku jsou levé a pravé vrcholy rozdílné a předchozí úrovni jsou shodné, tak už pro všechny nižší úrovně jsou obě hranice shodné. To využijeme k tomu, že budeme rozlišovat několik druhů plátku, podle toho, jak vypadá jejich hranice. Druh plátku a typ plátku, definovaný v popisu struktury `table_t`, jsou různé pojmy, i když mají určitou souvislost. Jednotlivé druhy plátků jsou znázorněny na obrázku 1.4. Nechť S je plátek druhu 1 nebo 3. Pak u plátku S nastala situace, kterou popisuje tvrzení 5, tedy že na nějaké úrovni j je levý a pravý vrchol hranice rozdílný a na další úrovni $j-1$ jsou oba vrcholy shodné. Pak ale plátek S neobsahuje žádné vrcholy úrovní $j-1$ až 1, mimo těch které má na hranici. To tedy znamená, že velikost nezávislé množiny v závislosti na tom, jestli jsou vrcholy hranice úrovní $j-2$ až 1 obsaženy v množině, je ovlivněna pouze tím, jestli jsou dané vrcholy spojeny hranou nebo ne. Tato informace je ale obsažena i v tabulkách pro sousední plátky, protože jak jsme dokázali, sousední plátky mají společnou hranici. Z toho plyne, že tato informace je nadbytečná a můžeme ji z tabulky odstranit, proto můžeme hranici plátku S zařízout na úrovni $j-1$, tedy z hranice odstraníme všechny vrcholy úrovní $j-2$ až 1. Pokud toto uděláme pro všechny plátky, tak se může stát, že tato informace zmizí i ze sousedních plátků. Což ale nevadí, protože při postupném spojování vždy narazíme na nějaký plátek S', který obsahuje vnější hranu e úrovně $j-1$, která obsahuje vrchol shodný s vrcholem úrovně $j-1$ plátku S, tedy tento plátek bude obsahovat (alespoň část) informace o hranici. Toto zkracování hranice nám pomůže v tom, že nám zmenší velikosti tabulek. A protože velikost tabulky je exponenciálně závislá na délce hranice, může být toto

zmenšení tabulek velmi výrazné.

Jak jsme popsali v minulém odstavci, budeme zařezávat hranice plátků zespoda. Může ale nastat i situace, kde jsou levé a pravé vrcholy hranice shodné pro několik nejvyšších úrovní a dále už shodné nejsou. Tuto situaci mimo nejvyšší úroveň rozlišovat nebudeme a budeme ji v implementaci brát, jako kdyby tyto vrcholy hranice byly různé. A také je budeme započítávat do položky `dif_levs` struktury pro tabulku.

V minulé části s popisem algoritmu jsme zmínili, že jsme schopni střední část plátku vnější hrany $e = uv$ úrovně větší než jedna sestrojit v konstantním čase. To platí, protože v naší reprezentaci tabulek zařezáváme spodní shodnou hranici, tedy plátek pro střední část bude tvořen pouze vrcholy u, v a m a hranami mezi nimi, kde m je vrchol předchozí úrovně spojený hranami s u i v v grafu s přidanými triangulačními hranami.

Nakonec ukážeme, jak budeme indexovat tabulky, které, jak už jsme zmínili, budeme reprezentovat polem. Nejprve vrcholy hranice plátku očíslujeme a to tak, že vrcholy číslujeme od nuly v pořadí od nejvyšší úrovně k nižším a zleva doprava. Toto očíslování nám bude sloužit k určení pořadí bitů v bitovém řetězci určujícím index do tabulky tvořené polem, kde nultý bit je nejnižší bit řetězce. Tedy když chceme indexovat nějakou položku tabulky pro určitou volbu vrcholů z hranice, tak nejprve vytvoříme bitový řetězec pro tuto volbu vrcholů, který budeme brát jako číslo a použijeme ho jako index pole.

1.4 Aproximační varianta

Doteď jsme popisovali přesnou verzi algoritmu na hledání nezávislé množiny. V této části ukážeme, jak jednoduše algoritmus upravit na aproximační. Mějme rovinný graf G. Upravený algoritmus funguje následovně: Zvolíme přirozené číslo k. Pak pro každé $i = 0, \ldots, k-1$ odstraníme z grafu G vrcholy každé úrovně j, kde $j \equiv i \mod k$, a na takto upravený graf spustíme přesný algoritmus. Nakonec vybereme největší nalezenou nezávislou množinu.

Podívejme se, jak dobrou aproximaci tím získáme. Nechť S_{OPT} je největší nezávislá množina grafu G. Z holubníkového principu plyne, že existuje alespoň jedno i, $0 \leq i < k$, takové, že počet vrcholů množiny S_{OPT} úrovní kongruentních s $i \mod k$ je nanejvýš $|S_{OPT}|/k$. Pokud tedy z grafu G odstraníme všechny vrcholy úrovní kongruentních s $i \mod k$ a najdeme největší nezávislou množinu tohoto grafu, tak její velikost bude alespoň $|S_{OPT}|(k-1)/k$. Takto upravený graf s odstraněnými úrovněmi bude mít komponenty souvislosti maximálně s k úrovněmi. Časová složitost tohoto algoritmu bude $O(8^k kn)$, tedy lineární ve velikosti grafu pro pevně zvolené k.

1.5 Uživatelská dokumentace

V této části popíšeme, jak se s naším programem pracuje. Nejprve se podívejme na formát vstupu, tedy jak programu předat graf. Tento formát je shodný pro všechny algoritmy, které jsme implementovali. Programu spolu s grafem předáváme i jeho nakreslení. Na prvním řádku vstupu je číslo n, které reprezentuje počet vrcholů vstupního grafu. Program předpokládá, že vrcholy grafu jsou očíslo-

vané čísly od 0 do $n-1$. Následují další řádky, pro každý vrchol jeden v pořadí od 0 do $n-1$, kde řádek pro daný vrchol obsahuje seznam sousedů tohoto vrcholu oddělený mezerami v pořadí průchodu proti směru hodinových ručiček. Následuje příklad reprezentace grafu G z obrázku 1.1.

```
23
1 7 6
2 3 11 5 0
1 3
1 2 4 5
3 5
4 6 9 1 11 3
5 0
0 8 10
7 9 19
8 5 10
9 7 22
1 12 5
11 13 14
12 14
13 12
16 22
15 17 18 19 20 21 22
16 18
16 17
16 8 20
```

Implementace přesné verze algoritmu předpokládá, že vstupní graf dostane na svém standardním vstupu. Po dokončení výpočtu vypíše výsledek na svůj standardní výstup. Jako počáteční vrchol pro každou komponentu souvislosti vstupního grafu bereme vrchol této komponenty označený nejnižším číslem a jako počáteční hranu bereme první hranu uvedenou ve vstupu pro počáteční vrchol. Výsledek obsahuje počet úrovní vstupního grafu a velikost nalezené nezávislé množiny. Uvedeme ukázkový výstup pro graf z obrázku 1.1.

```
This graph contains 3 levels.
The size of maximum independent set is 11.
```

U aproximační verze je to trochu jiné. Té musíme kromě grafu předat i číslo k, které reprezentuje počet vrstev, jak jsme popsali v předchozí části. Tento program očekává dva argumenty na příkazové řádce, prvním je číslo k a druhým je cesta k souboru obsahující vstupní graf. Stejně jako u přesného algoritmu, po dokončení výpočtu vypíše svůj výsledek na standardní výstup. Výstup obsahuje počet úrovní grafu, pak pro každé $j = 0, \ldots, k-1$ obsahuje velikost největší nezávislé množiny při vynechání vrcholů úrovní kongruentních s $j \mod k$ a nakonec maximální velikost nalezené nezávislé množiny. Zase uvedeme ukázkový výstup pro graf z obrázku 1.1, kde zvolíme $k = 3$.

```
This graph contains 3 levels.
The size of maximum independent set when ignoring levels congruent
to 0 mod 3 is 7. The number of ignored vertices is 8.
The size of maximum independent set when ignoring levels congruent
```

```
to 1 mod 3 is 8. The number of ignored vertices is 7.
The size of maximum independent set when ignoring levels congruent
to 2 mod 3 is 7. The number of ignored vertices is 8.
Size of maximal independent set found is 8.
```

2. Další algoritmy a generátory

Algoritmus popsaný v předchozí kapitole budeme chtít experimentálně srovnávat s jinými algoritmy. V této kapitole se podíváme na algoritmy, se kterými budeme srovnávat a také na to, jak budeme generovat grafy pro testování. Stručný popis srovnávaných algoritmů uvedeme části 2.1 a v části 2.2 popíšeme generátory grafů.

2.1 Další algoritmy

Algoritmus budeme porovnávat s dvěma dalšími algoritmy, a to s algoritmem pro obecné grafy publikovaným v článku (Fomin a kol., 2009), který jsme implementovali v jazyce C++, a algoritmem implementovaným v matematickém softwaru SAGE.

První algoritmus je rekurzivním algoritmem založeným na metodě rozděl a panuj. Jedná se o vylepšenou variantu prohledávacího algoritmu, ve kterém vždy vybereme jeden z vrcholů a rozvětvíme se na případ, kdy je daný vrchol v nezávislé množině a kdy ne, a rekurzivně zavoláme na tyto dva případy stejný algoritmus. Pro popis tohoto algoritmu zavedeme následující značení. Mějme graf $G = (V,E)$, pak pro vrchol v označme $d(v)$ jeho stupeň, $N(v) = \{u \in V | uv \in E\}$ jeho otevřené okolí a $N[v] = N(v) \cup \{v\}$ jeho uzavřené okolí. Dále potřebujeme operaci složení vrcholu, kterou definujeme pro vrcholy stupně 2. Nechť u je vrchol stupně 2 a v a w jsou jeho dva sousedi. Pokud jsou vrcholy v a w spojeny hranou, tak složení vrcholu u odstraní z grafu vrcholy u, v a w. Pokud vrcholy v a w hranou spojeny nejsou, tak složení vrcholu u do grafu vloží nový vrchol u', který hranami spojí s vrcholy z množiny $N(v) \cup N(w)$ a odstraní vrcholy u, v a w. Pro složení vrcholu zavedeme následující značení. Pokud G je graf a u jeho vrchol stupně 2, pak $\widetilde{G}(u)$ označuje graf po složení vrcholu u v grafu G. Zbývá ještě definovat zrcadlení vrcholu. Mějme graf $G = (V,E)$ a jeho vrchol $u \in V$. Pak vrchol $v \in V$ je zrcadlením vrcholu u, pokud v je ve vzdálenosti 2 od vrcholu u (tedy u a v nejsou spojeny hranou a existuje vrchol w takový, že $vw \in E$ a $wu \in V$) a podgraf indukovaný vrcholy $N(u) \setminus N(v)$ tvoří (možná prázdnou) kliku. Množinu všech zrcadlení vrcholu u označíme $M(u)$. Nakonec $V(G)$ značí množinu vrcholů grafu G a $E(U)$ pro $U \subseteq V$ značí množinu hran podgrafu indukovaného vrcholy množiny U. Pseudokód tohoto algoritmu je uveden v algoritmu 1. Více informací o tomto algoritmu lze dohledat v článku (Fomin a kol., 2009).

Algoritmus 1 Největší nezávyslá množina dle (Fomin a kol., 2009)

```
1: procedure MIS(G)
2:     if |V(G)| ≤ 1 then return |V(G)|
3:     if ∃ komponenta C ⊂ G then return MIS(C)+MIS(G − C)
4:     if ∃u,v ∈ V(G) : N[u] ⊆ N[v] then return MIS(G − {v})
5:     if ∃v ∈ V(G) : d(v) = 2 then return 1+MIS(G̃(v))
6:     vybereme vrchol v maximálního stupně takový, že minimalizuje |E(N(v))|
7:     return max(MIS(G − {v} − M(v)),1+MIS(G − N[v]))
```

Dalším algoritmem, se kterým srovnáváme, je algoritmus implementovaný v matematickém softwaru SAGE. Konkrétně se jedná o algoritmus implementovaný v proceduře `independent_set()`. Tato procedura funguje tak, že nejprve problém největší nezávislé množiny převede na problém hledání největší kliky a na ten použije algoritmus implementovaný v knihovně Cliquer. Více informací o této knihovně a použitém algoritmu lze nalézt v článku (Niskanen a Östergård, 2003). Pro testování jsme používali SAGE ve verzi 7.3.

2.2 Generování grafů

Algoritmy budeme testovat na náhodných grafech. Efektivita popsaných algoritmů závisí na vstupních grafech, proto k testování algoritmů potřebujeme zvolit způsob generování grafů, který bude ovlivňovat naměřené výsledky. Není na první pohled zřejmé, jak by se tyto grafy měly generovat. Generování náhodných rovinných grafů uniformně, tedy tak, aby každý graf měl stejnou pravděpodobnost, je složité. Proto jsme pro generování grafů použili několik různých generátorů, kde každý generuje rovinné grafy jiným způsobem. Tento přístup se může hodit při praktickém použití těchto algoritmů, protože pak můžeme lépe odhadnou chování algoritmů na grafech vycházejících z nějakého praktického problému. Generátory, které zde popíšeme, generují kromě samotného grafu i jeho nakreslení. Každému generátoru předáváme dva parametry a to n jako počet vrcholů a m jako počet hran.

První z použitých generátorů je generátor implementovaný v knihovně LEDA, konkrétně se jedná o funkci `random_planar_map`. Tento generátor budeme nazývat LEDA generátor. Postup fungování tohoto algoritmu je popsán v knize (Mehlhorn a Näher, 1999) v kapitole 8.9, ale pro přehlednost ho zde také uvedeme. Generátor nejprve induktivně vytváří triangulaci. Pro $n = 3$ vrátí trojúhelník. Pro $n > 3$ mějme graf G' s $n - 1$ vrcholy vytvořený tímto generátorem. Náhodně zvolíme jednu jeho orientovanou hranu e a vezmeme stěnu f která je nalevo od hrany e. Do této stěny f vložíme nový vrchol v s spojíme ho hranou se všemi vrcholy stěny f. Tímto máme triangulaci na n vrcholech, označme ji G. Dále budeme odstraňovat hrany grafu G, dokud jich má více než m. Vždy vezmeme rovnoměrně náhodně jednu hranu G a tu odstraníme. Tento generátor není moc dobrý, protože nedokáže vygenerovat všechny rovinné grafy. Každý graf vygenerovaný tímto generátorem bude mít vrchol stupně nejvýše 3. Také způsobem, který generátor používá, vzniká v generovaném grafu hodně vrcholů nízkého stupně. Generátor jsme použili hlavně z toho důvodu, že je volně přístupný v knihovně LEDA bez nutnosti ho implementovat.

Další generátor je založený na štěpení vrcholů, který jsme implementovali v jazyce C++, a nazveme ho štěpící generátor. Funguje také induktivně. Pro $n = 3$ vrátí trojúhelník. Pokud $n > 3$, tak nechť G' je graf s $n - 1$ vrcholy vytvořený tímto generátorem. Zvolíme náhodně vrchol v grafu G' a dvě různé hrany $e = uv$ a $e' = wv$ incidentní s vrcholem v. Dále vrchol v rozštěpíme na dva vrcholy a to následujícím způsobem: Přidáme vrchol v' tak, aby hrana vv' byla následující hranou proti směru hodinových ručiček k hraně e. Všechny hrany, které jsou mezi hranami e a e' při průchodu incidentních hran s vrcholem v proti směru hodinových ručiček, přesměrujeme z vrcholu v do vrcholu v', tedy pokud se jednalo o hranu xv, tak ji předěláme na xv'. Nakonec přidáme hrany

uv' a wv'. Tímto získáme graf G, který je triangulací. Stejně jako u předchozího generátoru, dokud má graf G více hran než m, tak náhodně jednu zvolíme a z grafu G odstraníme.

Nakonec popíšeme poslední generátor, který nazveme stromový. Tento generátor jsme také implementovali v jazyce C++. Funguje tak, že nejprve vygeneruje náhodný strom G na n vrcholech tak, že každý strom má stejnou pravděpodobnost. Generování stromu začneme tak, že náhodně vygenerujeme funkci $f : [n] \to [n]$, kde $[n]$ značí množinu $\{0, \ldots, n-1\}$. Budeme konstruovat graf na množině vrcholů $[n]$. Použijeme funkci f a každý vrchol i spojíme hranou s vrcholem $f(i)$. Tím získáme graf, který označíme H, tvořený cykly (kam zahrnujeme i jednovrcholové smyčky) a stromy, které se do těchto cyklů napojují. Nechť V' označuje množinu vrcholů, které leží na cyklech v grafu H. Funkce f zúžená na množinu V' tvoří permutaci. Vytvoříme tabulku obsahující na prvním řádku čísla z V' a druhém řádku příslušné hodnoty funkce f. Sloupce tabulky setřídíme podle hodnot v prvním řádku vzestupně. Druhý řádek nám dává pořadí vrcholů z V'. Cykly v grafu H nahradíme cestou tak, že hranami spojíme sousední vrcholy na druhém řádku setříděné tabulky. To nám dá strom G. Dále do grafu G přidáváme hrany, dokud jich graf nemá dostatečné množství. Přidávání hran funguje tak, že rovnoměrně náhodně vybereme stěnu f grafu G, která má více než tři hrany, zvolíme dva vrcholy u a v této stěny f, které nejsou sousední a nakonec tuto stěnu rozdělíme hranou uv.

3. Měření a výsledky

V této kapitole uvedeme způsob měření a také naměřené výsledky. Měření probíhalo na počítači s procesorem Intel Pentium G4400 a 16GB operační paměti, na kterém běžel operační systém Ubuntu 16.04. V části 3.1 uvedeme výsledky pro přesný algoritmus a v části 3.2 pro jeho aproximační verzi.

3.1 Přesný algoritmus

V této části se podíváme na výsledky měření, kde jsme srovnávali přesnou verzi algoritmu uvedeného v kapitole 1, který budeme nazývat hlavní, s dalšími algoritmy uvedenými v podkapitole 2.1, které budeme nazývat obecný algoritmus a algoritmus SAGE. Naměřené hodnoty uvedeme v tabulkách, kde každý řádek popisuje jeden test. V každém testu jsme vybrali 100 náhodných grafů, spustili na nich jednotlivé algoritmy a změřili čas potřebný k výpočtu a velikost spotřebované paměti. Pro běh každého algoritmu jsme nastavili časový limit 900 sekund (15 minut). V tabulkách jsou pak uvedeny průměrné hodnoty přes všechny grafy testu, které pro daný algoritmus doběhly.

Nejprve popíšeme identifikátory sloupců tabulek. První sloupec n obsahuje počet vrcholů grafu, sloupec k obsahuje počet úrovní grafu, sloupec t_1 obsahuje čas běhu hlavního algoritmu v sekundách a sloupec m_1 jeho spotřebovanou paměť v kilobytech, podobně pak sloupce t_2 a m_2 obsahují hodnoty pro obecný algoritmus a sloupce t_3 a m_3 obsahují hodnoty pro algoritmus SAGE.

V této části budeme hustotou grafu označovat poměr počtu jeho hran ku počtu jeho vrcholů, například graf s hustotou 2 má počet hran rovný dvojnásobku počtu vrcholů. Pro každý generátor grafů uvedeme 4 tabulky, každou pro jinou hustotu generovaných grafů. Testovali jsme grafy s hustotami 1, 1,5, 2 a 2,5.

Nejprve se podívejme výsledky pro grafy generované generátorem LEDA. Tento generátor generuje grafy s malým počtem úrovní v porovnání s ostatními použitými generátory. Měřili jsme hodnoty pro grafy s počtem vrcholů 1000 až 100000. Z naměřených hodnot plyne, že hlavní algoritmus je vždy lepší, než ostatní použité algoritmy, někdy i velmi výrazně. Například pro grafy s 50000

n	k	t_1	m_1	t_2	m_2	t_3	m_3
1000	4,45	0	3333	0	4717	1,3	170849
2500	5,13	0	4412	0,03	9061	1,41	172951
5000	5,7	0,01	6209	0,09	19308	1,7	176896
7500	5,97	0,02	8044	0,19	35045	2,05	181845
10000	6,19	0,03	9866	0,34	48972	2,71	184165
25000	6,92	0,08	20962	1,65	194811	7,96	215227
50000	7,44	0,19	39575	5,05	501792	26,55	317332
75000	7,73	0,32	58464	9,71	822398		
100000	8,02	0,42	77286	15,69	1311160		

Tabulka 3.1: Hodnoty naměřené přesnými algoritmy pro grafy generované generátorem LEDA s husotou 1

n	k	t_1	m_1	t_2	m_2	t_3	m_3
1000	5,11	0	3544	0,01	5127	1,26	171006
2500	5,87	0	4939	0,04	10063	1,45	172473
5000	6,37	0,02	7302	0,13	21885	1,77	177670
7500	6,7	0,03	9760	0,26	38166	2,24	183092
10000	6,99	0,04	12089	0,44	52363	3,01	185875
25000	7,53	0,14	26717	1,96	183741	9,58	219166
50000	8,17	0,3	51561	5,65	462413	33,02	354202
75000	8,41	0,46	76319	11,54	861546		
100000	8,77	0,66	101938	17,73	1262173		

Tabulka 3.2: Hodnoty naměřené přesnými algoritmy pro grafy generované generátorem LEDA s hustotu 1,5

n	k	t_1	m_1	t_2	m_2	t_3	m_3
1000	5,55	0	3655	0,01	5061	1,27	171117
2500	6,22	0,01	5192	0,05	9628	1,49	173402
5000	6,79	0,02	7831	0,14	19424	1,88	178515
7500	7,15	0,04	10499	0,27	31147	2,44	183931
10000	7,3	0,06	13187	0,46	46172	3,18	187783
25000	7,91	0,17	29481	1,92	153795	10,79	222990
50000	8,5	0,38	57020	5,7	382623	38,84	391659
75000	8,81	0,64	84550	11,85	758720		
100000	9,07	0,79	113255	15,54	955460		

Pro jeden graf velikosti 50000 nedoběhl algoritmus SAGE v časovém limitu.

Tabulka 3.3: Hodnoty naměřené přesnými algoritmy pro grafy generované generátorem LEDA s hustotu 2

n	k	t_1	m_1	t_2	m_2	t_3	m_3	s_3
1000	5,74	0	3705	0,01	4910	1,29	171330	0
2500	6,54	0,01	5329	0,05	9123	1,48	174290	0
5000	6,96	0,03	8031	0,14	17503	1,93	179696	0
7500	7,14	0,05	10728	0,33	28145	3,71	184989	0
10000	7,48	0,07	13505	0,47	39246	4,57	189400	1
25000	8,2	0,2	30260	1,86	121137	12,72	232284	4
50000	8,79	0,45	58539	5,24	292248	49,8	429586	9
75000	9,07	0,68	87131	8,96	503490			
100000	9,27	0,9	115424	13,43	694804			

Sloupec s_3 obsahuje počet grafů, pro které algoritmus SAGE nedoběhl v časovém limitu.

Tabulka 3.4: Hodnoty naměřené přesnými algoritmy pro grafy generované generátorem LEDA s hustotu 2,5

vrcholy a 100000 hranami je hlavní algoritmus v průměru 15 krát rychlejší než obecný algoritmus a dokonce 102 krát rychlejší než algoritmus SAGE, což plyne z tabulky 3.3.

Dále se podívejme na výsledky pro grafy generované štěpícím generátorem. Zde už je situace trochu jiná. Pro grafy s hustotou 1 a 1,5 je hlavní algoritmus vždy nejlepší, což plyne z tabulek 3.5 a 3.6. Pro hustší grafy to už je horší. Pro grafy s hustotou hran 2 je hlavní algoritmus srovnatelný s obecným algoritmem pro grafy do velikosti 1000 a pro větší grafy je horší a dokonce 10 krát horší pro grafy velikosti 7500, jak vidíme z tabulky 3.7. Kromě časové nevýhodnosti hlavního algoritmu pro hustší grafy s větším počtem vrcholů, je problém také s pamětí. Jak vidíme v tabulce 3.8, grafy s 7500 vrcholy a 18750 hranami mají průměrně kolem patnácti úrovní, a to je pro nás problém, protože tabulka pro plátek úrovně 15 může zabírat až 4GB paměti, proto také hlavní algoritmus pro 9 grafů této velikosti nedoběhl z důvodu nedostatku paměti.

Nakonec se podívejme na na výsledky pro grafy generované stromovým generátorem. Nejprve se podívejme na grafy s hustotou 1, pro které výsledky nalezneme v tabulce 3.9. Pro tyto grafy je hlavní algoritmus jasným vítězem. Je to způsobeno tím, že tyto grafy mají nízký počet úrovní, maximálně 2. To se dá očekávat vzhledem ke způsobu generovaní. Generátor nejprve vytvoří náhodný strom, proto tento graf má jen jednu úroveň a počet hran je o jedna menší než počet vrcholů. Proto přidáním poslední hrany vznikne maximálně jedna další úroveň.

Pro grafy s hustotou 1,5 je situace podobná. Sice už tyto grafy nemají málo vrstev, ale pořád je pro ně hlavní algoritmus mnohem lepší než ostatní, například pro grafy s 2500 vrcholy je hlavní algoritmus průměrně 107 krát lepší než obecný algoritmus a 24 krát lepší než algoritmus SAGE.

Pro grafy s hustotami 2 a 2,5 už výsledky dopadly jinak. V této situaci už se nedá určit jednoznačný vítěz, každý z algoritmů je pro nějaké velikosti lepší a pro nějaké horší. Dá se ale říci, že tyto grafy dávají zabrat všem algoritmům, protože pro mnoho z těchto grafů algoritmy nedoběhly v časovém limitu nebo skončily pro nedostatek paměti. Například pro grafy s 5000 vrcholy a hustotou 2,5 algoritmus SAGE doběhl v časovém limitu pouze pro 4 grafy ze 100.

3.2 Aproximační verze

V této části budeme srovnávat aproximační verzi algoritmu s jeho přesnou verzí a změříme, jak přesnou aproximaci nám algoritmus dává a jak dlouho mu to trvá. Testování probíhalo tak, že jsme pro každý graf nejprve zjistili velikost největší nezávislé množiny přesným algoritmem a pak jsme pro různé volby parametru k popsaném v podkapitole 1.4 spustili aproximační algoritmus a spočítali relativní velikost nalezené nezávislé množiny, tedy podíl velikosti nezávislé množiny nalezené aproximačním algoritmem a velikosti největší nezávislé množiny.

V každém testu jsme vygenerovali 100 náhodných grafů a naměřené hodnoty pro každý test uvedeme v samostatné tabulce, kde každá hodnota je vždy průměr přes všechny testované grafy. Ve sloupci k je uvedena volba parametru k aproximačního algoritmu, kde jsme pro každý test volily hodnoty 2 až 14. Ve sloupci s_{rel} je relativní velikost nezávislé množiny. Sloupce t a t_{rel} obsahují čas běhu aproximačního algoritmu v sekundách a relativní čas běhu, což je poměr

n	k	t_1	m_1	t_2	m_2	t_3	m_3
500	4,05	0	2993	0	4010	1,16	170200
750	4,56	0	3172	0	4716	1,18	170664
1000	4,59	0	3332	0,01	5640	1,21	170843
2500	5,75	0	4435	0,06	14896	1,4	173047
5000	6,6	0,01	6295	0,23	40718	1,78	177125
7500	7,14	0,02	8219	0,54	88831		

Tabulka 3.5: Hodnoty naměřené přesnými algoritmy pro grafy generované štěpícím generátorem s hustotu 1

n	k	t_1	m_1	t_2	m_2	t_3	m_3
500	6,08	0	3190	0	4609	1,16	170231
750	6,88	0	3569	0,01	5978	1,2	170745
1000	7,38	0	4081	0,02	7966	1,22	171035
2500	8,94	0,06	10556	0,15	29007	1,44	172766
5000	10,73	0,33	49356	0,62	95335	1,89	178112
7500	11,7	0,62	76490	1,43	192437		

Tabulka 3.6: Hodnoty naměřené přesnými algoritmy pro grafy generované štěpícím generátorem s hustotu 1,5

n	k	t_1	m_1	t_2	m_2	t_3	m_3	f
500	7,5	0	3790	0	4784	1,19	170342	0
750	8,33	0,01	4699	0,01	6379	1,2	170852	0
1000	9,03	0,02	6240	0,03	8588	1,23	171209	0
2500	11,23	0,31	38453	0,2	32794	1,5	173775	0
5000	12,76	4,03	353128	0,86	108600			1
7500	14,08	20,85	1098479	2,01	226360			5

Sloupec f obsahuje počet grafů, pro které hlavní algoritmus nedoběhl z důvodu nedostatku paměti.

Tabulka 3.7: Hodnoty naměřené přesnými algoritmy pro grafy generované štěpícím generátorem s hustotu 2

n	k	t_1	m_1	t_2	m_2	t_3	m_3	f
500	8,29	0,02	5369	0,01	4760	6,21	170554	0
750	9,37	0,03	6843	0,02	6299	1,23	171015	0
1000	9,89	0,07	12519	0,05	8499	1,83	171457	0
2500	12,22	1,15	95682	0,59	32048	25,87	176418	0
5000	14,22	14,71	1181625	5,42	112007			6
7500	15,24	66,66	1502558	12,6	227172			9

Sloupec f obsahuje počet grafů, pro které hlavní algoritmus nedoběhl z důvodu nedostatku paměti.
Algoritmus SAGE nedoběhl v časovém limitu pro 46 grafů velikosti 2500.

Tabulka 3.8: Hodnoty naměřené přesnými algoritmy pro grafy generované štěpícím generátorem s hustotu 2,5

n	k	t_1	m_1	t_2	m_2	t_3	m_3	s_2
500	1,99	0	2926	0,01	5819	1,22	171399	0
750	2	0	3070	0,03	8696	1,24	171853	0
1000	2	0	3250	0,1	13089	1,24	172011	0
2500	2	0	4180	10,11	63265	1,42	174143	1
5000	2	0	5561	79,98	238067	1,75	176801	11

Sloupec s_2 obsahuje počet grafů, pro které obecný algoritmus nedoběhl v časovém limitu.

Tabulka 3.9: Hodnoty naměřené přesnými algoritmy pro grafy generované stromovým generátorem s hustotu 1

n	k	t_1	m_1	t_2	m_2	t_3	m_3	s_2
500	6,89	0	3205	0,01	5053	1,25	171425	0
750	7,77	0	3609	0,04	7135	1,26	171852	0
1000	8,34	0	4106	0,08	9966	1,21	172224	0
2500	10,61	0,06	11492	6,51	42733	1,46	174385	1
5000	13,02	0,9	75170	109,36	154121	2,7	177748	26

Sloupec s_2 obsahuje počet grafů, pro které obecný algoritmus nedoběhl v časovém limitu.
Hlavní algoritmus nedoběhl pro jeden graf velikosti 2500 z důvodu nedostatku paměti.
Algoritmus SAGE nedoběhl v časovém limitu pro 2 grafy velikosti 5000.

Tabulka 3.10: Hodnoty naměřené přesnými algoritmy pro grafy generované stromovým generátorem s hustotu 1,5

n	k	t_1	m_1	t_2	m_2	t_3	m_3	s_3
500	8,45	0	3802	0,01	4789	1,22	171474	0
750	9,86	0,04	8641	0,04	6353	1,27	172096	0
1000	10,89	0,05	8147	0,13	8955	1,27	172530	0
2500	13,95	2,08	224220	7,49	36201	15,54	175556	10
5000	16,38	27,32	1190867	52,32	126428	24,2	185382	42

Sloupec s_3 obsahuje počet grafů, pro které algoritmus SAGE nedoběhl v časovém limitu.
Hlavní algoritmus nedoběhl pro jeden graf velikosti 2500 a 4 grafy velikosti 5000
z důvodu nedostatku paměti.
Obecný algoritmus nedoběhl v časovém limitu pro 6 grafů velikosti 2500
a 8 grafů velikosti 5000.

Tabulka 3.11: Hodnoty naměřené přesnými algoritmy pro grafy generované stromovým generátorem s hustotu 2

n	k	t_1	m_1	t_2	m_2	t_3	m_3	s_3
500	10,52	0,07	8777	0,02	4638	1,25	171670	0
750	11,4	0,16	18092	0,15	6071	4,67	172514	4
1000	12,56	1,06	74497	0,36	8296	2,99	173568	7
2500	16,42	36,78	1417829	27,78	31665	34,51	185165	57
5000	15,47	113,46	4624079	128,8	108809	12,55	241923	96

Sloupec s_3 obsahuje počet grafů, pro které algoritmus SAGE nedoběhl v časovém limitu.
Obecný algoritmus nedoběhl v časovém limitu pro 8 grafů velikosti 2500
a 37 grafů velikosti 5000.
Hlavní algoritmus nedoběhl pro 8 grafů velikosti 2500 a 19 grafů velikosti 5000
z důvodu nedostatku paměti.

Tabulka 3.12: Hodnoty naměřené přesnými algoritmy pro grafy generované stromovým generátorem s hustotu 2,5

mezi časem běhu aproximačního algoritmu a časem běhu přesného algoritmu. Podobně sloupce m a m_{rel} obsahují spotřebovanou paměť v kilobytech a relativní spotřebovanou paměť.

Když se podíváme na hodnoty relativního času v uvedených tabulkách, uvidíme, že některé hodnoty jsou větší než 1, někdy i výrazně. To na první pohled může připadat nelogické, protože bychom předpokládali, že aproximace bude trvat kratší dobu. Je to ale způsobeno tím, že v každém běhu aproximační verze spouštíme k krát přesný algoritmus na grafy, kde vynecháme některé vrstvy. Dokonce pokud má graf méně vrstev, než je zvolený parametr k, tak vlastně během chodu aproximační verze spouštíme i přesnou verzi algoritmu, možná několikrát.

Z popisu aproximačního algoritmu v podkapitole 1.5 je zřejmé, že pro pevně zvolené k nám algoritmus garantuje aproximační poměr alespoň $(k-1)/k$. Protože v tabulkách s naměřenými výsledky vždy uvádíme naměřený aproximační poměr, tak v tabulce 3.13 pro srovnání uvedeme minimální aproximační poměr, který vždy algoritmus dosáhne. Jak vidíme z naměřených hodnot, skutečný aproximační poměr je vyšší.

Nejprve se podívejme na výsledky pro grafy z generátoru LEDA. S tímto generátorem jsme testovali na grafech s 100000 vrcholy a hustotou 2 (tabulka 3.14) a 2,5 (tabulka 3.15). Takto velké grafy jsme zvolili z toho důvodu, protože i přesný algoritmus na nich běžel rychle. Pro tyto grafy není aproximační algoritmus moc dobrý, už pro volbu $k = 3$ běží aproximace pomaleji, než přesný algoritmus, takže je mnohem lepší pro ně použít přesný algoritmus.

Dále se podívejme na grafy generované štěpícím generátorem, kde jsme měřili grafy s 2500 vrcholy a 6250 hranami (tabulka 3.16) a v druhém měření grafy s 5000 vrcholy a 10000 hranami (tabulka 3.17). Zde už je situace lepší a jeho použití dává smysl. Například pro $k = 6$, což garantuje aproximační poměr alespoň 83,3 %, nalezl aproximační algoritmus v obou případech nezávislou množinu velikost průměrně 91 % velikosti největší nezávislé množiny za 18 % nebo 15 % času, který by potřeboval přesný algoritmus.

Nakonec zhodnotíme aproximační algoritmus pro grafy generované stromovým generátorem, který jsme testovali pro grafy s 2500 vrcholy a 5000 (tabulka 3.18) nebo 6250 hranami (tabulka 3.19). Pro tyto grafy nám aproximační algoritmus

k	garantovaný aproximační poměr
2	0,5000
3	0,6667
4	0,7500
5	0,8000
6	0,8333
7	0,8571
8	0,8750
9	0,8889
10	0,9000
11	0,9091
12	0,9167
13	0,9231
14	0,9286

Tabulka 3.13: Garantovaný aproximañní poměr aproximačního algoritmu v závislosti na parametru k

k	s_{rel}	t	t_{rel}	m	m_{rel}
2	0,55	0,49	0,89	84461	0,74
3	0,75	0,62	1,14	88057	0,78
4	0,87	0,78	1,44	92493	0,81
5	0,95	1,06	1,95	97719	0,86
6	0,98	1,29	2,39	103219	0,91
7	0,98	1,55	2,86	109172	0,96
8	0,99	1,82	3,36	111884	0,99
9	0,99	2,07	3,84	112558	0,99
10	1	2,34	4,32	112786	0,99
11	1	2,58	4,78	112815	1
12	1	2,84	5,26	112826	1
13	1	3,09	5,71	112834	1
14	1	3,35	6,2	112844	1

Tabulka 3.14: Naměřené hodnoty aproximačního algoritmu pro grafy velikosti 100000 s hustotou 2 generované generátorem LEDA, kde grafy měly průměrně 8,93 úrovní a přesný algoritmus na nich běžel průměrně 0,53 s a spotřeboval 112820 kB paměti

k	s_{rel}	t	t_{rel}	m	m_{rel}
2	0,56	0,56	0,93	88934	0,77
3	0,75	0,7	1,16	92540	0,8
4	0,87	0,87	1,44	97171	0,84
5	0,95	1,13	1,88	101847	0,88
6	0,98	1,36	2,25	106668	0,92
7	0,99	1,59	2,64	111631	0,96
8	0,99	1,84	3,06	114378	0,99
9	0,99	2,08	3,45	115179	0,99
10	1	2,33	3,87	115447	0,99
11	1	2,57	4,28	115504	0,99
12	1	2,81	4,68	115499	1
13	1	3,06	5,1	115510	1
14	1	3,31	5,51	115499	0,99

Tabulka 3.15: Naměřené hodnoty aproximačního algoritmu pro grafy velikosti 100000 s hustotou 2,5 generované generátorem LEDA, kde grafy měly průměrně 9,32 úrovní a přesný algoritmus na nich běžel průměrně 0,6 s a spotřeboval 115554 kB paměti

k	s_{rel}	t	t_{rel}	m	m_{rel}
2	0,63	0	0	4904	0,19
3	0,76	0	0	4988	0,2
4	0,83	0,01	0,07	5089	0,2
5	0,88	0,01	0,09	5246	0,21
6	0,91	0,03	0,18	5538	0,22
7	0,94	0,06	0,38	6288	0,25
8	0,96	0,13	0,7	8514	0,32
9	0,98	0,28	1,24	14070	0,46
10	0,98	0,57	1,99	25161	0,67
11	0,99	1,07	2,88	42956	0,86
12	0,99	1,88	3,85	67302	0,97
13	1	2,73	4,77	91827	1,02
14	1	3,63	5,7	116937	1,04

Tabulka 3.16: Naměřené hodnoty aproximačního algoritmu pro grafy velikosti 2500 s hustotou 2,5 generované štěpícím generátorem, kde grafy měly průměrně 12,09 úrovní a přesný algoritmus na nich běžel průměrně 0,99 s a spotřeboval 120409 kB paměti

k	s_{rel}	t	t_{rel}	m	m_{rel}
2	0,59	0,01	0,03	6848	0,12
3	0,74	0,01	0,03	7011	0,13
4	0,81	0,02	0,05	7198	0,13
5	0,87	0,03	0,09	7479	0,14
6	0,91	0,06	0,15	7949	0,14
7	0,93	0,11	0,27	9180	0,17
8	0,96	0,24	0,5	12626	0,22
9	0,97	0,53	0,9	21469	0,32
10	0,98	1,19	1,51	42053	0,5
11	0,99	2,4	2,29	84280	0,71
12	0,99	4,55	3,13	156683	0,86
13	0,99	8,04	4,05	249151	0,94
14	1	12,89	4,99	386597	1

Tabulka 3.17: Naměřené hodnoty aproximačního algoritmu pro grafy velikosti 5000 s hustotou 2 generované štěpícím generátorem, kde grafy měly průměrně 12,95 úrovní a přesný algoritmus na nich běžel průměrně 4,01 s a spotřeboval 395757 kB paměti

k	s_{rel}	t	t_{rel}	m	m_{rel}
2	0,59	0	0	4672	0,22
3	0,74	0	0	4780	0,22
4	0,82	0	0,03	4876	0,23
5	0,87	0,01	0,09	5002	0,23
6	0,91	0,02	0,19	5263	0,25
7	0,93	0,05	0,39	5834	0,27
8	0,95	0,12	0,72	7597	0,34
9	0,97	0,27	1,25	11546	0,45
10	0,98	0,58	1,97	20229	0,62
11	0,98	1,13	2,81	35046	0,78
12	0,99	1,9	3,72	54807	0,9
13	0,99	2,79	4,64	73223	0,96
14	0,99	3,7	5,58	80279	0,99

Tabulka 3.18: Naměřené hodnoty aproximačního algoritmu pro grafy velikosti 2500 s hustotou 2 generované stromovým generátorem, kde grafy měly průměrně 13,53 úrovní a přesný algoritmus na nich běžel průměrně 0,98 s a spotřeboval 80217 kB paměti

k	s_{rel}	t	t_{rel}	m	m_{rel}
2	0,63	0	0	4809	0,03
3	0,75	0	0	4907	0,03
4	0,83	0,01	0,01	5004	0,03
5	0,87	0,01	0,01	5167	0,03
6	0,9	0,03	0,02	5470	0,04
7	0,92	0,09	0,05	6277	0,04
8	0,94	0,24	0,12	8827	0,06
9	0,95	0,7	0,28	16095	0,1
10	0,96	2,08	0,58	37001	0,19
11	0,97	6,02	1,09	94099	0,35
12	0,98	15,17	1,77	236849	0,55
13	0,98	33,83	2,59	530958	0,74
14	0,99	66,26	3,5	940435	0,89

Tabulka 3.19: Naměřené hodnoty aproximačního algoritmu pro grafy velikosti 2500 s hustotou 2,5 generované stromovým generátorem, kde grafy měly průměrně 16,57 úrovní a přesný algoritmus na nich běžel průměrně 40,84 s a spotřeboval 1378375 kB paměti

také dává dobré výsledky. V prvním měření nám vyšlo, že jsme schopni nalézt nezávislou množinu velkou průměrně 74 % velikosti největší nezávislé množiny za méně než 1 % času přesného algoritmu. U druhého měření to dopadlo o malinko lépe, tam jsme schopni za méně než 1 % času přesného algoritmu nalézt nezávislou množinu velkou průměrně 75 % velikost největší nezávislé množiny. Pokud chceme velikost alespoň 90 %, tak stejně jako u předchozího generátoru můžeme zvolit $k = 6$ a dostaneme nezávislé množiny velikosti průměrně 91 % (u prvního testu) nebo 90 % (u druhého testu) za 19 % nebo 2 % času přesného algoritmu.

Závěr

V této práci jsme se zabývali hledáním největší nezávislé množiny v rovinných grafech a její aproximací. Uvedli jsme algoritmus, který hledá největší nezávislou množinu v čase lineárním s počtem vrcholů ale exponenciálním s počtem úrovní grafu. Tento algoritmus nazýváme hlavní. Dále jsme uvedli, jak tento algoritmus převést na aproximační. Provedli jsme experimentální srovnání naší implementace přesné verze hlavního algoritmu s dalšími dvěma algoritmy, jeden z nichž jsme také implementovali a u druhého jsme použili existující implementaci v matematickém softwaru SAGE. Aproximační verzi algoritmu jsme srovnávali s jeho přesnou verzí a měřili jsme, jak přesnou aproximaci nám dává a v jakém čase běží.

Z našich naměřených výsledků plyne, že ve většině případů je hlavní algoritmus lepší než ostatní srovnávané algoritmy. Pro grafy z generátoru LEDA je přesný algoritmus výrazně lepší než ostatní testované algoritmy. Tento generátor ale není dobrým reprezentantem, protože, jak jsme uvedli u popisu generátorů, nedokáže vygenerovat všechny rovinné grafy a většinou generuje grafy, kde je velký počet vrcholů nízkého stupně. U ostatních dvou generátorů je situace přibližně následující. Pro grafy s hustotou do 1,5 je hlavní algoritmus lepší než zbylé algoritmy, někdy i výrazně. Pro grafy s vyšší hustotou je hlavní algoritmus většinou srovnatelný s ostatními algoritmy, někdy trochu horší.

Dále jsme zjistili, že pro grafy z generátoru LEDA nemá smysl používat aproximační algoritmus, protože už i přesná verze je na těchto grafech rychlá a protože při aproximační verzi spouštíme přesnou verzi na menší grafy, ale několikrát, proto je časová náročnost aproximační verze ve většině případů horší než přesná verze. Pro ostatní generátory už má aproximační verze smysl. Například pro $k = 6$ jsme ve všech měřeních nalezli nezávislou množinu velikosti průměrně alespoň 90 % velikosti největší nezávislé množiny v čase maximálně 19 % času přesné verze.

Dále uvedeme možná vylepšení, která jsme už neimplementovali. U aproximační verze by bylo možné použít rovnou přesnou verzi v případě, kdy zvolené k je větší, než počet úrovní v daném grafu. Následující možné vylepšení by mohlo zlepšit aproximační poměr, sice ne teoreticky, ale v praxi nejspíše ano. Nechť máme graf $G = (V,E)$ a číslo k, pak $V_i \subseteq V$ označuje množinu vrcholů úrovní kongruentních s $i \mod k$. Aproximační algoritmus hledá největší nezávislou množinu v grafu G s vynechanými vrcholy V_i. Tuto největší nezávislou množinu označíme S_i. Označíme V_i' množinu vrcholů, která vznikne z množiny V_i odebráním všech sousedů vrcholů z množiny S_i. Mohli bychom pak k této nezávislé množině S_i přidat největší nezávislou množinu v podgrafu indukovaném vrcholy V_i'. Protože tento podgraf má pouze jednu úroveň, byli bychom schopni tuto největší nezávislou množinu najít rychle přesným algoritmem. Sjednocení těchto dvou nezávislých množin tvoří nezávislou množinu grafu G.

Hlavní slabinou hlavního algoritmu je jeho paměťová náročnost. Tento problém jsme se snažili řešit pomocí zařezávání hranic plátků, které sice nezmenšilo asymptotickou složitost, ale pro praktické použití to nějaké úspory přinese. Dalším způsobem řešení tohoto problémů by mohlo být prořezávání tabulek. Nejspíše by se dalo ukázat, že některé položky tabulek jsou zbytečné, tedy buď se nikdy nepoužijí, nebo se dají jednoduše odvodit z jiných položek. Vynecháním těchto položek by přineslo další zmenšení tabulek, i když asi také ne asymptoticky.

Seznam použité literatury

Baker, B. S. (1994). Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, **41**(1), 153–180. ISSN 0004-5411. doi: 10.1145/174644.174650. URL http://doi.acm.org/10.1145/174644.174650.

Bondy, J. A. (1976). *Graph Theory With Applications*. Elsevier Science Ltd., Oxford, UK, UK. ISBN 0444194517.

Bourgeois, N., Escoffier, B., Paschos, V. a Van Rooij, J. (2012). Fast Algorithms for max independent set. *Algorithmica*, **62**(1), 382–415. doi: 10.1007/s00453-010-9460-7. URL https://hal.archives-ouvertes.fr/hal-01509542.

Fomin, F. V., Grandoni, F. a Kratsch, D. (2006). Measure and conquer: A simple $O(2^{0.288n})$ independent set algorithm. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 18–25, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics. ISBN 0-89871-605-5. URL http://dl.acm.org/citation.cfm?id=1109557.1109560.

Fomin, F. V., Grandoni, F. a Kratsch, D. (2009). A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, **56**(5), 25:1–25:32. ISSN 0004-5411. doi: 10.1145/1552285.1552286. URL http://doi.acm.org/10.1145/1552285.1552286.

Garey, M. R. a Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, first edition edition. ISBN 0716710455.

Garey, M. R., Johnson, D. S. a Stockmeyer, L. (1974). Some simplified NP-complete problems. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 47–63, New York, NY, USA, 1974. ACM. doi: 10.1145/800119.803884. URL http://doi.acm.org/10.1145/800119.803884.

Håstad, J. (1999). Clique is hard to approximate within $n^{1-\varepsilon}$. *Acta Math.*, **182**(1), 105–142. doi: 10.1007/BF02392825. URL http://dx.doi.org/10.1007/BF02392825.

Lipton, R. J. a Tarjan, R. E. (1979). A Separator Theorem for Planar Graphs. *SIAM Journal on Applied Mathematics*, **36**(2), 177–189. URL http://www.jstor.org/stable/2100927.

Mehlhorn, K. a Näher, S. (1999). *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, New York, NY, USA. ISBN 0-521-56329-1.

Niskanen, S. a Östergård, P. R. J. (2003). Cliquer user's guide, version 1.0. *Communications Laboratory, Helsinki University of Technology, Espoo, Finland, Tech. Rep. T48*.

Seznam tabulek