FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Implementace detekce anomálií v grafech
Student:	Bc. Ondřej Filip
Vedoucí:	Ing. Jaroslav Kuchař, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Problematika hledání anomálií v datech reprezentovaných jako graf je poměrně neprozkoumaná oblast, která v porovnání s detekcí anomálií v běžných datech, přináší řadu nových výzev. Cílem práce je seznámit se s metodami řešícími problém hledání anomálií v grafech, implementovat vybrané metody za použití frameworku Gephi Toolkit a následně vyhodnotit výsledky.
Pokyny:
1. Seznamte se s metodami pro hledání anomálií v datech reprezentovaných jako graf a popište je.
2. Porovnejte dostupné metody a vyberte vhodné algoritmy pro následnou implementaci.
3. Navrhněte, implementujte a otestujte knihovnu zahrnující vybrané algoritmy. Využijte framework Gephi toolkit.
4. Otestujte implementaci na vhodných datech a diskutujte výsledky.
5. Pokuste se provést modifikaci některé z metod nebo navrhnout vlastní postup, který by přinášel lepší výsledky. Návrhy vyhodnoťte.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 13. února 2018

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Implementace detekce anomálií v grafech

Bc. Ondřej Filip

Vedoucí práce: Ing. Jaroslav Kuchař, Ph.D.

7. května 2018

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 7. května 2018 .....

Odkaz na tuto práci

Filip, Ondřej. *Implementace detekce anomálií v grafech.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Práce se zabývá problematikou detekce anomálií v grafech. Popisuje a zhodnocuje několik existujících postupů řešící tuto poměrně novou a málo prozkoumanou problematiku. Dále je v práci představena dvojice vlastních navržených postupů, první je rozšířením již existujícího algoritmu a druhý založený na změnách entropie grafu při jeho editaci. Výsledkem práce je implementovaná knihovna obsahující některé vybrané existující postupy společně se dvěma vlastními metodami. Veškeré implementované metody jsou vyhodnoceny a porovnány na syntetických grafech a na grafech z reálného prostředí.

Klíčová slova anomálie, detekce anomálií v grafech, Gephi Toolkit

Abstract

Subject of this thesis is anomaly detection in graph. Thesis describes and evaluates several existing approaches dealing with this new and very little studied problem domain. Two new and own approaches are presented, fist one extending one of existing algorithm and second one based of graph entropy changes during its updates. One of the results is a library implementation containing some selected existing methods along with two new one. All implemented methods are evaluated and compared using synthetic and real-world graphs.

Keywords anomaly, graph anomalies detection, Gephi Toolkit

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Grafy jsou důležitým způsobem reprezentace dat, který je běžně používán v mnoha různých vědních oborech již dlouhá desetiletí. S postupným rozvojem počítačových technologií a především s nástupem Internetu, se grafová data stávají čím dál tím významnější, především díky tomu že velké množství dat které každým dnem vznikají jsou implicitně v grafové podobě nebo jsou do ní velmi snadno transformovatelná.

V mnoha aplikacích kde se taková data využívají jako jsou telekomunikace, platební systémy, datová infrastruktura nebo online sociální sítě je bezpečnost a identifikace podezřelého chování vitální úloha, ve které detekce anomálií přichází na scénu. Primárním problémem je rozpoznání vzorů v datech, které se vymykají jejich běžnému chování. Takové vzory se obvykle nazývají anomálie či odlehlé hodnoty. Anomálie se mohou vyskytovat v mnoha různých podobách, které jsou obvykle silně závislé na problémové doméně ze které data pochází.

Problému hledání anomálií je jedním z odvětví vytěžování znalostí z dat. Znalost vzorů, které se v datech vyskytují výjimečně je, v závislosti na zkoumané doméně, často mnohem důležitější a zajímavější než zkoumání častých struktur. Detekcí anomálií v nestrukturovaných více-dimenzionálních datech se věnovala spousta výzkumných prací, které se věnovali různým aplikačním doménám a existují desítky metod, které se tímto problémem zabývají. Výzkumů které by cílili na problematiku hledání anomálií v datech reprezentovaných jako graf je podstatně méně.

Propojení mezi jednotlivými entitami do dat přináší velké množství nových informací, které je možné efektivně využít při hledání abnormálních výskytů v datech, což také byla jedna z hlavních motivací proč se touto problematikou vědci začali v posledních několika letech zabývat hlouběji.

Cíl práce

Cílem této práce jsou následující:

- Analýza doposud známých metod zabývající se řešením problematiky detekce anomálií v grafech.
- Porovnání metod a výběr vhodných kandidátů pro implementaci.
- Návrh, implementace a otestování knihovny implementující vybrané algoritmy za využití frameworku Gephi toolkit.
- Otestování implementovaných metod na vhodných datech.
- Experimentální návrh zlepšení některých z prozkoumaných metod, který by zlepšil získané výsledky a jeho následné vyhodnocení.

Kapitola 1

Hledání anomálií v grafech

Tato kapitola je věnována definicím několika základních pojmů a notace z teorií grafů, které budou využívány v následujícím textu. Dále je zde vysvětlen pojem anomálie a jeho souvislosti v kontextu grafových dat.

1.1 Definice použitých pojmů z teorie grafů

Základním pojmem v teorii grafů je samozřejmě graf. Neformálně se jedná o množinu vrcholů (uzlů) a hran, které mezi vrcholy vytváří propojení, také označováno jako incidenci. Pokud mezi dvěma vrcholy u a v existuje hrana, jsou nazývány sousedními vrcholy.

Pokud nebude řečeno jinak, pod pojmem graf se v této práci bude považovat obyčejný neorientovaný graf, ve kterém mezi dvěma různými vrcholy může existovat nejvýše jedna hrana.

Definice 1.1. (Neorientovaný graf). Graf G je uspořádaná dvojice (V, E), kde V je neprázdná množina a V je množina neuspořádaných dvojic z množiny V. Prvky V jsou nazývány **vrcholy** (z angl. vertices) a prvky E jsou nazývány **hrany** (z angl. edges).

Tento typ grafu je tím nejzákladnější typem, využívaným, při detekci anomálií v grafově orientovaných datech. Jednotlivá entity jsou reprezentována vrcholy grafu a hrany vyjadřují vztah mezi nimi. Graf však umožňuje vytvořit hranu jejíž oba konce směřují do jednoho vrcholu (tzv. smyčku), což v mnohých aplikacích nemusí dávat smysl, například v sociálních sítích nemůže být uživatel přítel sám se sebou, nebo v telekomunikačním systému nemůže volat sám sobě. Z toho důvodu je obvykle nutné se omezit na obyčejný graf.

Definice 1.2. (Obyčejný neorientovaný graf). Graf $G = (V, E)$ nazveme obyčejným neorientovaným grafem pokud je neorientovaný graf a zároveň pro žádný vrchol v z V neexistuje hrana v, v z E. Tyto hrany se obvykle nazývají **smyčka**.

V dnešní éře tzv. „velkých dat“ jsou analyzované grafy na tolik rozsáhlé, že není technicky možné v jednu chvíli pracovat s grafem jako celkem, ale je nutné analýzu zaměřit jen na část tohoto grafu. Proto je zde zaveden pojem podgrafu.

Definice 1.3. (Podgraf). Graf $G' = (V', E')$ je **podgrafem** grafu $G = (V, E)$, pokud platí že $V' \subseteq V$ a $E' \subseteq E$

V některých aplikačních doménách sebou propojení mezi entitami nesou určitou dodatečnou informaci o míře své významnosti. Například v systému monitorující pohyb uživatelů po webových stránkách, intenzita přechodů z jedné konkrétní stránky na druhou významnou roli a proto by na ní měl být brán zřetel při následné analýze grafu. Tyto informace o významnosti hran v grafu je možné vyjádřit pomocí vážených hran.

Definice 1.4. (Vážený graf). Váženým grafem se rozumí graf $G = (V, E)$ společně s funkcí $w : E \to \mathbb{R}$, ohodnocující jednotlivé hrany jejich váhami.

1.2 Detekce anomálií

Přesto že detekci anomálií bylo věnováno spousta výzkumů a bádání, je stále poměrně složité definovat pojem anomálie obecným a formálním způsobem.

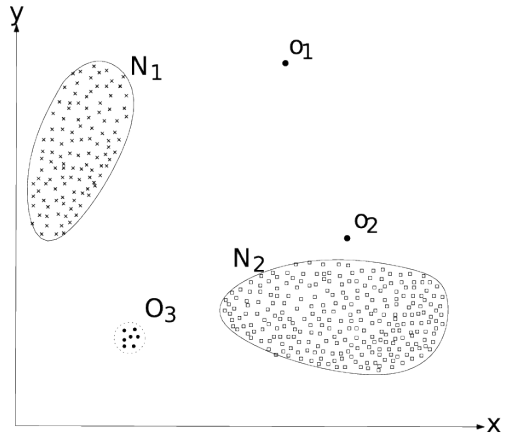
Ve spojení s vytěžováním znalostí z dat je anomálie obvykle vnímána jako hodnota v množině dat, které se svou povahou liší a neodpovídá tak očekávané struktuře nebo zbytku dat. Tyto anomální pozorování, které se v datech vyskytují, jsou obvykle generovány jiným mechanizmem než zbytek dat.

Pro účely této práce bude použit intuitivní pojem pro anomálii jakožto neočekávané nebo vymykající se pozorování od zbytku dat.

V praxi to může znamenat například množinu bankovních transakcí vygenerovanou skupinou běžných uživatelů jakožto jedna skupina pozorovaných dat, kterou by bylo možné označit za očekávanou strukturu dat odpovídající normálnímu chování. Druhou skupinou by byly transakce útočníka, který se snaží narušit systém a obohatit se na něm. Je pravděpodobné, že útočníkovo chování bude odlišné od běžného uživatele a v datasetu se jeho transakce projeví jako anomálie. Při vizualizaci těchto dat budou útočníkovy transakce ležet mimo skupiny transakcí běžných uživatel, proto se těmto anomálií také často říká odlehlá hodnota (angl. outlier).

Je běžné, že tento odlišný mechanismus nevygeneruje pouze jedeno pozorování, ale hned několik podobných, které v datasetu vytvoří vlastní shluk, čímž se detekce těchto anomálií značně komplikuje. V takovém případě již detekce jednotlivých pozorování není dostačující a je nutné jako anomálii označit i celý shluk dat.

Na obrázku 1.1 z článku [5] je ilustrativní zobrazení různých druhů anomálií v jednoduchém dvou dimenzionálním data setu. Data mají dva normální



Obrázek 1.1: Jednoduchý příklad vizualizace anomálií

regiony N_1 a N_2, vzhledem k tomu že většina pozorování leží v těchto oblastech. Body které leží podstatně dále od normálních oblastí (o_1 a o_2) jsou pravděpodobně anomální pozorování. Stejně tak shluk pozorování O_3 je pravděpodobně vygenerován jiným mechanismem než normální oblasti a lze jej považovat za anomální.

1.3 Anomálie v grafech

Ne vždy je možné k získaným pozorování přistupovat jako k obyčejným bodům ve více rozměrném prostoru. Data v sobě mohou obsahovat tzv. vnitřní závislosti, které by měli být brány v potaz při detekci anomálií jakožto významná doplňující informace.

Příkladem takových dat může být webový portál ve kterém mají uživatelé možnost hodnotit produkty. Pokud budou tato data o hodnocení analyzována a budou v nich hledány anomálie, jsou spojení mezi uživatelem a produktem který hodnotí stěžejní. Pokud se hodnocení jednoto uživatele bude vymykat tomu jak stejné produkty hodnotí ostatní, nemusí být jeho hodnocení tak důvěryhodné.

Přirozenou reprezentací těchto dat je graf, kde vrcholy grafu jsou jednotlivá pozorování a závislost mezi nimi je reprezentována hranou. Doplňující informace o pozorování jsou v grafu reprezentována jako atributy nebo typy hran a uzlů.

1.4 Typy anomálií v grafech

Anomálie se v grafech může projevit několika triviálními způsoby [6]

1. Hrana obsažená v grafu je neočekávaná

2. Vrchol obsažený v grafu je neočekávaný
3. Atribut u hrany je jiný než očekávaný
4. Atribut u vrcholu je jiný než očekávaný
5. Očekávaná hrana v grafu chybí
6. Očekávaný vrchol v grafu chybí

Je zřejmé, že jednotlivé typy anomálií je možné generalizovat i na podstruktury grafu, tedy provázané skupiny vrcholů a hran.

Jednotlivé typy anomálií odpovídají základním modifikacím: vložení, modifikaci a odstranění. Konkrétně případy 1. a 2. odpovídají vložení, 3. a 4. modifikace a 5. a 6. odstranění.

Anomálie, které se v grafech mohou vyskytovat lze rozřadit do několika kategorií. V závislosti na aplikační doméně se typy anomálií mohou lišit. Nyní rozebereme několik druhů anomálií, které se obvykle vyskytují napříč různými doménami.

1.4.1 Podle povahy anomálie

Bodové anomálie

Bodové anomálie se v datech projevují pokud ojedinělý datový objekt vykazuje odlišné vlastnosti od zbytku dat. Přesto že se jedná o jednodušší typ projevu anomálie, definovat vhodnou míru anomálnosti, který by popisovala jak moc se objekt liší od zbytku, je stále problém.

Skupinové anomálie

Skupinová anomálie jsou protikladem bodových anomálií. Přesto že bodovým anomáliím je věnována větší pozornost, alespoň co se vědeckých prací týče, v některých aplikačních doménách (obzvláště v sociálních sítích) mohou být skupinové anomálie mnohem zajímavější.

Například pokud se skupina uživatel rozhodne snížit průměrné hodnocení nějakého produkty a vytvoří falešné recenze, může se toto chování na úrovni jednotlivce jevit jako normální.

1.4.2 Podle dopadu anomálie

Globální anomálie

Globální anomálie se v grafu projevují při analýze grafu jako celku. Jedná se o běžnější variantu zkoumaných anomálií.

Lokální anomálie

Jak název napovídá, lokální anomálie tvoří protipól globálních anomálií a jsou studovány relativně k jejich blízkému sousedství.

Příkladem může být graf reprezentující skupinu jedinců, kde propojení vyjadřuje přátelství a u každého z nich je evidován jeho plat. Pokud jeden z uživatelů bude mít relativně nižší plat ve srovnání s lidmi, se kterými se přátelí, může se jednat o anomálii. Tato anomálie se však na globální úrovni nemusí projevit, protože jeho plat může být v rámci celého grafu průměrný.

1.4.3 Podle dynamické/statické povahy grafu

Dynamické anomálie

Dynamické anomálie lze v grafu detekovat na základě analýzy průběžných změn grafu v čase. Vyskytují se tedy pouze v dynamických grafech, které se časem vyvíjí.

Statické anomálie anomálie

Detekce anomálií pouze v jednom konkrétním okamžiku ignorující faktor času.

1.4.4 Doménově závislé anomálie

Výše je výčet pouze nejzákladnějších typů anomálií, které lze v grafech obecně pozorovat. Pokud bychom se zaměřili na některou konkrétní problémovou doménu, mohou pro ni být zajímavé další druhy anomálií.

Velmi zajímavou a často skloňovanou problémovou doménou v kontextu detekce anomálií v grafech jsou v poslední době sociální sítě. Pokud bychom se zaměřili pouze na analýzu sociálních sítí, mohou nás zajímat některé z anomálních fenoménů, které se v nich běžně vyskytují [7].

Anomálie bílé vrány

Vzniká pokud se jeden datový objekt velice silně liší od všech ostatních. Například pokud v datech nalezneme záznam o uživateli, který jako svou výšku zadal 25 metrů, což je nemožné, poté to považujeme za takzvanou anomálii bílé vrány.

Anomálie v přestrojení

Tento druh anomálie se vyskytuje jako drobná odchylka v datech od normálního vzoru. Například pokud se někomu podaří získat kontrolu nad cizím uživatelským účtem na některé ze sociálních sítí, může se snažit svým chováním napodobit chování opravdového majitele účtu a předejít tak podezření. Tento druh anomálií je velmi těžké detekovat.

Hvězdy/kliky v grafu

Velmi silně provázané podgrafy tvořící téměř kliku nebo naopak velmi rozpojené podgrafy tvořící hvězdy, jsou v sociálních sítích obvykle považovány za anomálii.

Vysoká lokální váha

Vysoké váhy hran v oblasti některých uzlů nebo skupin, jsou v sociálních sítích také považovány za podezřelé.

1.5 Výzvy při detekci anomálií

Na obecné úrovni jsou za anomálii považovány taková data, které svou strukturou neodpovídají normálnímu chování. Přímočarý přístup při hledání anomálií představuje detekci oblastí v analyzovaném data setu, které odpovídají normálnímu chování a data která nespadají do těchto oblastí považovat za anomálii.

Tento postup je však velmi idealizovaný a v praxi naráží na velké množství faktorů, které celou problematiku dělají výrazně složitější[5].

- **Definice běžného chování**, které by pokrývalo všechny možné způsoby jakým se běžné chování může projevovat je velice složité. Navíc hranice mezi běžným chováním a anomálií nejsou ostře vymezené. Mezi těmito dvěma oblastmi mohou vznikat překryvy a anomální pozorování blížící se těmto hranicím, může být snadno chybně považováno za normální a naopak.

  Některé algoritmy proto nereprezentují svůj výstup binárně (pozorování je nebo není anomální), ale definují určitou hladinu anomálnosti.

- Pokud je anomálie výstupem nějakých úmyslných škodlivých akcí, útočník se je často snaží **zamaskovat jako normální chování**, které bude odpovídat zbytku analyzovaného data setu.

- V některých aplikačních doménách se **normální chování vyvíjí s časem** a jeho momentální podoba nemusí v budoucnosti odpovídat realitě. Toto se může projevit dvěma způsoby.

  Prvním způsobem je periodická změna chování, která může být způsobena změnou provozu v analyzovaném systému. Například normální množství telefonických spojení mezi jednotlivými uživateli se může opakovat v denním či týdenním cyklu na což je třeba brát zřetel pro návrhu metod, které jsou při analýze použity.

  Druhým typem jsou postupné změny, způsobené například pozvolným růstem analyzovaného systému, obecnou mentalitou uživatelů kteří jej využívají, technologickým vývojem apod.

- Dalším problémem při detekci anomálií je jejich silná **doménová závislost**. Například ve zdravotnictví může malá odchylka v měřené teplotě pacienta znamenat anomální pozorování. Na druhou stranu, pokud na takovouto drobnou odchylku narazíme ve vývoji cen produktů, může se jednat o normální chování. To má za následek, že ne všechny postupy je možné přímočaře aplikovat na různé problémové domény.
- **Dostupnost trénovacích dat**, ve kterých by byli označeny anomální záznamy je obvykle velmi závažný problém.
- **Šum obsažený v datech** může být svou povahou často velice podobný anomálnímu chování a proto je obtížné je od sebe odlišit.

1.5.1 Výzvy v grafově orientovaných datech

Výše zmíněné výzvy se týkaly hledání anomálií v datech obecně. Detekce anomálií v grafových datech sebou však nese několik jim specifických výzev.

- **Vnitřní provázanost dat** sama o sobě zvyšuje míru složitosti při vymezení míry anomálnosti jednotlivých grafových objektů. Zatím co v běžné detekci anomálií jsou data obvykle považována za nezávislá z identického rozdělení, u objektů v grafech je nutné brát v potaz i vztahy mezi nimi.
- **Rozdílnost v definicích anomálie** v grafech je mnohem větší než je tomu v běžných datech. Příčinou je především velké množství typů grafů a jejich variant (orientovaný, neorientovaný, atributovaný, vážený, statický, dynamický...), společné v kombinaci doménovou závislostí.
- **Velikost prohledávaného prostoru** je samozřejmě problém i v běžných datech. V grafech je však jednou z hlavních výzev hledání anomálních podstruktur, což svou komplexitou vede na velmi rozsáhlý stavový prostor. Výčet všech podstruktur je kombinatorický problém, který dělá problém hledání anomálií mnohem složitější.

Kapitola 2

Srovnání známých metod

Ačkoliv je problematika detekce anomálií v grafových datech poměrně málo prozkoumaná, existuje několik známých metod, které ji řeší. Tato kapitola se věnuje popisu a srovnání některých z nich.

Kapitola je rozdělena do několika části, nejprve jsou definovány nároky, které jsou kladeny na jednotlivé metody a na základě, kterých jsou následně hodnoceny. V další části je definováno několik kategorií do kterých je metody možné rozdělit na základě způsobu přístupu k problematice. Následuje popis jednotlivých metod a na závěr je shrnuto jejich srovnání.

2.1 Srovnávací kriteria

Jak bylo řečeno v předchozí kapitole algoritmy řešící problém detekce anomálií v grafech se potýkají s celou řadou výzev. Za tu největší z nich lze považovat fakt, že neexistuje žádná obecně uznávaná formální definice pro problém detekce anomálie. Z čehož plyne, že různé výzkumné práce k tomuto problému přistupují různě a mají i různé výsledky.

Další velkou překážkou je nedostatek testovacích dat, ve kterých by byly anomálie označeny a dali se použít při vyhodnocení kvality jednotlivých metod. Většina článků popisující tyto metody se uchyluje buď k použití synteticky generovaných dat, nebo využívají reálná data, ve kterých však anomálie nejsou označeny a výsledky se snaží nějakým způsobem interpretovat a vyjádřit z nich kvalitu metody.

Z toho důvodu není jednoduché definovat několik jasných kritérií na základě, kterých by bylo možné metody rovnocenně porovnat. Přesto zde uvedu několik vlastností, které by mohli být směrodatné, při výběru některé z metod pro aplikaci na konkrétní problém.

Schopnost detekovat anomální uzel

Detekce anomálního uzlu je jedním ze základních problémů, na který se zaměřuje velké množství metod. Algoritmus by měl být schopen detekovat a označit konkrétní uzel jako anomální.

Schopnost detekovat anomální hranu

Detekce anomální hrany je možné rozdělit na dva typy

1. **Abnormální váha hrany** - Váha je jednou ze základních vlastností, kterou je hraně možné přiřadit. Detekce spočívá v porovnání reálné váhy hrany s očekávanou, která je z normálního rozložení vah v grafu.

2. **Neočekávaná hrana** - Hrana mezi dvěma uzly kde hrana nebyla (resp. byla) očekávaná a v grafu se vyskytuje (resp. nevyskytuje)

Možnost aplikace na podgraf

Analyzované grafy mohou být na tolik rozsáhlé, že není technicky možné v jednu chvíli pracovat s grafem jako celkem, ale je nutné analýzu zaměřit jen na část tohoto grafu. Schopnost aplikovat algoritmus pouze na podgraf analyzovaného grafu a získat smysluplné výsledky proto může být klíčová.

Definovat jakým způsobem graf rozdělit aniž by se narušila „anomálnost“ struktur jeho částí, nemusí být triviální.

Výpočetní složitost

Vychází z rozměrů grafů pro analýzu. Ideálně vyžadujeme lineární nebo ještě lépe sublineární růst výpočetní komplexity s rostoucí velikostí grafu.

Výstup

Co se výstupu týče, lze algoritmy rozdělit do dvou skupin

1. **Algoritmy s binárním výstupem** - Objektům v grafu pouze přiřazují dvoustavově zda jsou či nejsou anomální.

2. **Algoritmy se spojitým výstupem** - Snaží se objektům v grafu přiřadit určitou míru „anomálnosti“, které umožňuje další mnohem jemnější zpracování. Obvykle je tato míra pozitivní buď z intervalu $[0, 1]$ nebo $[0, \infty)$.

Možnost aplikovat na běžná data

Běžnými daty jsou v tomto kontextu myšlena více dimenzionální data, která neobsahují žádné vnitřní propojení. Tato vlastnost se může zdát zbytečná s přihlédnutím k tomu, že existují stovky algoritmů, které se přímo specializují na tuto problematiku. Přesto tyto grafové algoritmy mohou v datech detekovat vnitřní závislosti a dosahovat srovnatelných výsledků.

V tomto případě je zároveň vyžadováno, aby měla metoda i jasně definovaný způsob jakým tabulková data převést na graf.

Typ aplikovatelných grafů

Metody jsou omezeny typem grafu, na který lze použít.

- Vážený × bez vah
- Atributovaný × bez atributů
- Orientovaný × neorientovaný

Parametrizace algoritmu

Některé algoritmy po uživateli vyžadují zadání parametrů.

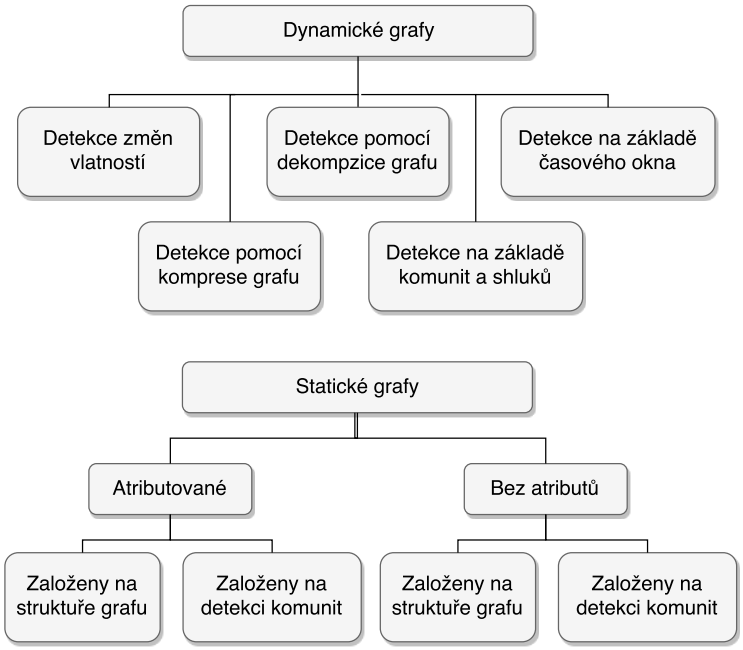
2.2 Kategorizace metod

V této sekci je popsán způsob jakým lze existující metody detekce anomálií v grafech rozdělit do několika kategorií. Dělení je možné provést ve dvou rovinách:

1. **Na jaký druh grafu lze metody aplikovat** - Hlavní dvě kategorie grafů, které jsou v tomto článku rozpoznávány jsou dynamické a statické grafy. Dále je možné provést jemnější dělení těchto skupin podle toho zda v sobě graf nese atributy, zda jsou jeho hrany vážené apod.

2. **Jakým způsobem metoda k problému přistupuje** - Tyto přístupy jsou silně odlišné v závislosti na tom na jaký druh grafu jsou použity a budou popsány v dalších sekcích.

Diagram na obrázku 2.1 naznačuje jakým způsobem jsou metody v tomto textu kategorizovány. V následujících sekcích jsou kategorie podrobněji rozebrány.



Obrázek 2.1: Digram kategorizace metod

2.2.1 Dynamické grafy

V reálném světě se grafy neustále vyvíjejí a mění. Toto chování je zachyceno takzvanými dynamickými grafy. Pod pojmem dynamický graf se skrývá časová řada grafů (snapshotů konkrétního grafu v různých časech), které zachycují vývoj analyzovaného grafu v čase.

Detekce anomálie v dynamickém grafu se obecně dá definovat následovně.

Definice 2.1. Detekce anomálie v dynamickém grafu. [8] Pro zadanou sekvenci (atributovaných nebo bez atributů) grafů nalezněte

1. časovou značku odpovídající změně nebo události, společně s
2. top-k vrcholů, hran nebo částí grafu, které nejvíce přispívají ke změně.

V závislosti na aplikační doméně se požadavky na algoritmus mohou lišit, zde jsou popsány některé z nich, které jsou obvykle společné pro různé domény[8]

- **Škálovatelnost** - Vzhledem k rychlosti jakým graf může denně růst, škálovatelnost algoritmu hraje klíčovou roli. Ideálně by jeho časová komplexita měla růst lineárně s rychlostí růstu vstupního grafu. V kontextu dynamického grafu je obvykle požadováno aby byla složitost algoritmu lineární vzhledem k velikosti změny ve vstupním grafu.

- **Citlivost na strukturální a kontextuální změny** - Algoritmus by měl být schopen detekovat strukturální změny mezi dvěma porovnávanými stavy grafu (jako jsou chybějící/nová hrana/vrchol, změny ve vahách hran) a stejně tak i změny ve vlastnostech grafu jako jsou změny atributů vrcholů nebo uzlů.

- **Povědomí o významu změny** - Algoritmus by měl být citlivý na rozsah a dopad změn. Změny ve významných vrcholech, hranách nebo grafových atributech by měli mít vyšší ohodnocení anomálnosti než méně významné změny.

2.2.1.1 Detekce změn vlastností grafu

Založeny na detekci změn v různých metrikách grafu. Pokud jsou rozdíly v extrahovaných metrikách dostatečně nízké, jsou považovány za podobné. Existují různé metriky, které se využívají při při detekci anomálií jako jsou například [9]

- Editační vzdálenost grafů (angl. Error correction graph matching distance), definována jako počet operací potřebných pro převedení jednoho grafu na druhý

- Maximální společný podgraf (MCS)

- Hammingova vzdálenost matic sousednosti

Metrik definujících podobnost dvou grafů je poměrně mnoho. Problémem však může být, že mnoho z nich bylo vyvinuto pro jiné účely než pro detekci anomálie v grafu, proto ne vždy musí být pro tento problém vhodné. Navíc obvykle ke grafu přistupují jako k celku a nedokáží detekovat jaká změna v grafu zapříčinila jeho abnormálnost.

2.2.1.2 Detekce pomocí komprese grafu

Metody které pro reprezentaci grafu využívají *Minimum description length (MDL)* a komprimují ho za využití častých vzorů, které se v něm vyskytují. Anomálie jsou poté definovány jako podgrafy, které nelze dobře zkomprimovat.

2.2.1.3 Detekce pomocí dekompozice grafu

Tato kategorie metod detekuje anomálie, tak že časově závislou posloupnost grafů reprezentuje jako matici na kterou aplikuje faktorizaci nebo redukci dimenzí.

2.2.1.4 Detekce na základě komunit a shluků

Místo zabývání se grafem jako jedním celkem tyto metody metody zkoumají změny na úrovní komunit nebo shluků v grafu.

2.2.1.5 Detekce na základě časového okna

Metody detekující změny vážící se k omezenému časovému období v časem se vyvíjejícím grafu. V jednoduchosti, místo toho aby se porovnávali pouze dva nebo více po sobě jdoucích snapshotů grafu, se na základě n předchozích snapshotů (kde n je velikost časového okna) sestaví model normálních struktur grafu a porovná se s následujícím snapshotem.

2.2.2 Statické grafy

Statickým grafem je myšlen jeden konkrétní snapshot grafu v konkrétním čase.

2.2.2.1 Atributované grafy

Hrany či vrcholy grafu mohou obsahovat atributy jakožto dodatečné informace. Atributovaný graf lze formálně definovat následujícím způsobem.

Definice 2.2. Atributovaný graf se spojitými atributy uzlu. [10] Graf $G = (V, E)$ je obyčejný neorientovaný graf. Každý $v \in V$ je popsán vektorem $(o_1, ...o_d) \in \mathbb{R}$ v d-dimenzionálním spojitém prostoru, kde atributy jsou pojmenovány $A = A_1, ..., A_d$.

Analogicky je možné definici rozšířit i na atributované hrany nebo diskrétní atributy.

Metody pro detekci anomálií v atributovaných grafech lze dále dělit na 2 skupiny.

- **Založeny na struktuře grafu** - Hlavním cílem je identifikovat podstruktury v grafu, které jsou strukturálně odlišné s ohledem jak na provázanost uzlů, tak na atributy, které se k nim váží.
- **Založeny na detekci komunit** - Metody se snaží detekovat silné provázané oblasti grafu, které dohromady tvoří shluk a na základě hodnot atributů uzlů v rámci jednotlivých shluků detekovat, které z nich jsou ve shluku abnormální.

2.2.2.2 Bez atributů

Jedná se o běžný graf skládající se z uzlů a hran, popřípadě vážených hran.

- **Založeny na struktuře grafu** - Využívají různé grafové metriky jako jsou stupně uzlů, centrality, egonety apod. Tyto vlastnosti jsou z grafu extrahovány a společné s dalšími informacemi tvoří prostor pro detekci anomálií.

- **Založeny na detekci komunit** - Fungují na podobném principu jako v atributovaných grafech, jen v tomto případe jsou anomálie detekovaná na základě jejich provázanosti v grafu, například silné provázání s více shluky, nebo obtížná zařaditelnost do některého z existujících shluků.

2.3 Známé metody

Přesto že je problematika hledání anomálií v grafových datech poměrně novým tématem, za posledních několik let vzniklo pár desítek metod, které se jí zabývají.

V této sekci je vypsaná zhruba desítka z těch nejpoužívanějších. Detekce anomálií v grafech je poměrně rozsáhlé téma a vzhledem k množství kombinací různých typů grafu a způsobů přístupů bylo nutné se omezit na nějakou konkrétní oblast. Kvalitní, poměrně rozsáhlý a aktuální (březen 2015) přehled metod pro dynamické grafy je zpracován v tomto článku [11]. Z toho důvodu se text bude věnovat pouze metodám zabývajícími se statickými grafy.

Metody jsou seřazeny dle své kategorie do které patří, přičemž pořadí kategorií je dáno jejich pořadím definovaných kategorií v sekci 2.2. Pro každou metodu je stručně popsáno do jaké kategorie patří, jakým způsobem k problému přistupuje, jaké jsou její silné popřípadě slabé stránky a nakonec jakým způsobem si stojí v kritériích definovaných v sekci 2.1.

2.3.1 Subdue

Kategorie - Graf s atributy, založeno na vlastnostech grafu

Jeden z prvních článků [12], který se v roce 2003 zabýval problematikou anomálií v grafově orientovaných datech, na který se odkazuje velká spousta navazujících výzkumů. Tento postup nemá žádný konkrétní název, je však součástí systému Subdue[13], jehož jádrem je stejnojmenný algoritmus, pro detekci opakujících se vzorů v grafu. Proto je tato metoda pro detekci anomálií v této práci pojmenována stejně.

V článku jsou nastíněny dva různé postupy, první se zaměřuje na detekci anomálních vzorů a druhý na detekci anomálních podgrafů.

Detekce anomálních vzorů

Jednodušší ze dvou zmíněných přístupů, který je založen na extrakci všech vzorů, které se v grafu vyskytují a jejich následné ohodnocení na základě jejich

četnosti výskytu, kde intuitivně čím méně často se vyskytují tím anomálnější jsou.

Tento jednoduchý postup však čelí zásadnímu problému. V případech kde je vzor velmi malý (co do počtu uzlů), bude se v grafu logicky vyskytovat mnohem častěji a nebude tedy považován za anomální. V případech kdy je vzor příliš velký bude se v grafu vyskytovat podstatně méně krát a proto má vysokou šanci na to být anomální i když anomální být nemusí. Například vzor kde počet jeho uzlů se rovná počtu uzlů grafu (vzor je celý graf) bude vždy anomální, protože jeho četnost je vždy 1.

Z toho důvodu je v článku předvedena jednoduchá heuristika, která tento problém řeší následujícím způsobem.

$$F(S,G) = Size(S) * Instances(S,G)$$

kde S je vzor v grafu G, funkce $F(S,G)$ přiřazuje anomální ohodnocení vzoru S v grafu G, $Size(S)$ je počet uzlů struktury S a $Instances(S,G)$ je počet instancí struktury S v grafu G.

Detekce anomálních podgrafů

Druhá metoda detekuje anomální podgrafy a využívá kompresi grafu pomocí *Minimum description length* (MDL). Je založena na faktu, že podgrafy obsahující mnoho společných vzorů jsou obecně méně anomální, než podgrafy obsahující málo společných vzorů.

Algoritmus pracuje iterativně a v každé iteraci zkouší jednotlivé podgrafy a porovnává jak moc je možné graf zkomprimovat celý a s odejmutím vybraného podgrafu. Pokud je rozdíl (ozn. A) velký, je podgraf v grafu méně častý. V každém kroku iterace jsou z grafu vyjmuty podgrafy s nejnižším A, a případně kondenzovány do jednoho uzlu. V posledních iteracích graf obsahuje jen nejméně časté podgrafy.

V článku je také nastíněn postup jakým způsobem lze metody použít na běžné tabulková data, transformovaná do grafu. Transformace spočívá ve vygenerování hvězdy pro každý řádek v tabulce, kde centrální blank uzel je bez atributu a na něj jsou napojeny uzly, kde každý nese hodnotu sloupce odpovídajícího řádku jako svůj atribut. Tyto hvězdy jsou poté spojeny do jedné velké hvězdy, přidáním dalšího uzlu na který jsou napojeny všechny blank uzly řádků.

Vyhodnocení autoři článku provedli na 1999 KDD cup[14] datové sadě, která obsahuje provoz na webové sítí společně s označenými útoky, jedná se o tabulková data která transformovali. Výsledky jsou relativně kvalitní, ale jsou uvedeny jen z části provedeného měření a navíc poměrné vágně popsány.

Tabulka 2.1: SUBDUE - vyhodnocení na základě def. kritérií

Schopnost detekce uzlu	Ano
Schopnost detekce hrany	Ne
Aplikace na podgraf	Ano za předpokladu, že rozdělení podobných vzorů napříč grafem je rovnoměrné
Výpočetní složitost	Není uvedena, z popisu metody však plyne že je vyšší než lineární
Výstup	Ohodnocení podgrafu z intervalu $[0, 1]$
Aplikace na běžná data	Ano
Typ aplikovatelných grafů	Statický, Atributovaný, Neorientovaný
Parametrizace algoritmu	Není

2.3.2 GBAD

Kategorie - Graf s atributy, založeno na vlastnostech grafu

Vědecká práce [15] navazuje na Subdue 2.3.1, ale k problému přistupuje trochu jiným způsobem. Útočníci se často snaží imitovat běžné chování, čím úspěšnější jsou, tím je pravděpodobnost jejich detekce nižší. Autoři článku berou v potaz tento hlubší náhled na věc a místo toho aby se snažili detekovat málo se vyskytující vzory v grafu, hledají spíše vzory, které jsou velmi podobné ale ne zcela stejné jako normativní vzory grafu.

GBAD se zaměřuje na tři základní typy anomálií v atributovaném grafu: vložení vrcholu/hrany, odstranění vrcholu/hrany a modifikace atributu uzlu/hrany. Definuje tři různé algoritmy pro detekci anomálií GBAD-MDL, GBAD-P a GBAD-MPS.

GBAD-MDL

Algoritmus využívá *Minimum description length* (MDL) pro detekci nejčastější podstruktury v grafu a následné zkoumá všechny ostatní a hledá podobný vzor. Tento postup je velmi podobný Subdue 2.3.1 a hledá obtížně komprimovatelné vzory v grafu.

GBAD-P

Funguje na podobném principu jako GBAD-MDL, ale místo detekce podobných podgrafů, hledá podobná rozšíření podgrafu (rozšíření ve smyslu přidání uzlu nebo hrany do grafu s určitou pravděpodobností), čímž dokáže detekovat hranu/uzel, který by v grafu měl být ale není.

Tabulka 2.2: GBAD - vyhodnocení na základě def. kritérií

Schopnost detekce uzlu	Ano
Schopnost detekce hrany	Ano
Aplikace na podgraf	Ne
Výpočetní složitost	Není uvedena, z popisu metody však plyne že je vyšší než lineární, navíc pro kompletní analýzu grafu je nutné použít všechny 3 algoritmy
Výstup	Ohodnocení podgrafu z intervalu $[0, \infty]$
Aplikace na běžná data	Ano
Typ aplikovatelných grafů	Statický, Atributovaný, Neorientovaný
Parametrizace algoritmu	Není

GBAD-MPS

Nejprve detekuje pomocí MDL podgrafy nejlépe popisující normální rozdělení podgrafů. Následně se snaží nalézt podgrafy, které je možné jejich transformací (přidáváním/ubíráním uzlů/hran nebo úpravy atributu) převést na některé grafy z normálního rozdělení. Podgrafy s nejmenším počtem nutných transformací jsou považovány za anomální, protože jsou velmi blízké normálnímu rozdělení, ale přesto se do něj nedostaly.

Tato trojice algoritmů pokrývá všechny 3 typy anomálních úprav grafu popsaných výše. Uvažované anomálie v článku se vždy skládají jen z jednoho druhu operace vložení, odstranění a modifikace, ale s anomáliemi skládajícími se z posloupnosti např. modifikace a vložení by metoda mohla mít problém.

Přesto na testovacích datech, které použili měla metoda poměrně dobré výsledky. Opět je nutné podotknout, že stejně jako 2.3.1, je možné tuto metodu použít na běžná tabulková data, s obdobnou transformací na graf.

2.3.3 GOutRank

Kategorie - Graf s atributy, založeno na shlukování

Autoři článku [10] se zaměřují na detekci anomálií v atributovaném grafu s vysokou dimenzialitou atributů. Hlavní hypotézou algoritmu GOutRank je, že identifikovat komplexní anomálii v grafu je možné pouze za předpokladu využití všech dostupných informací o datech. Z toho důvodu zkoumají atributovaný graf na dvou rovinách

1. Struktura grafu - Provázanost vrcholů grafu poskytuje určitý kontext ve kterém se anomálie může vyskytovat

Tabulka 2.3: GOutRank - vyhodnocení na základě def. kritérií

Schopnost detekce uzlu	Ano
Schopnost detekce hrany	Ne
Aplikace na podgraf	Ano
Výpočetní složitost	V závislosti na použitém algoritmu pro shlukování (není přímo uveden v článku)
Výstup	Ohodnocení uzlu z intervalu $[0, \infty]$
Aplikace na běžná data	Ne
Typ aplikovatelných grafů	Statický, Atributovaný, Neorientovaný
Parametrizace algoritmu	Není

2. Prostor atributů - Množina atributů ve které se anomálie může vychylovat od zbytku vrcholů v určitém kontextu.

Anomálie ve více dimenzionálním prostoru nemusí být snadno rozpoznatelná, protože se od zbytku dat může vymykat pouze v podmnožině všech atributů a zbylé např. atributy s náhodným rozdělením, mohou anomálii zamaskovat. Tento jev se obecně nazývá prokletí dimenzionality[16]. Zjednodušeně řečeno jsou jednotlivé body v prostoru s vysokou dimenzí vzájemně přibližně stejně vzdáleny, což způsobuje že se jeví vzájemně podobně rozdílné/stejné.

Postup algoritmu lze rozdělit do dvou fází, detekce shluků v grafu v kombinaci s atributy vrcholů a ohodnocení vrcholů mírou jejich anomálnosti. První fázi se v článku přímo nezabývají, místo toho představují několik již existujících algoritmů, které se tomuto problému věnují a které původně nebyli navrženy k detekci anomálií. Druhé fázi se věnují podstatně podrobněji. Popisují tři různé přístupy jakými vypočíst anomální skóre uzlu v grafu na základě jeho odlišnosti ve shluku. První zaměřující se pouze na atributy vrcholu, druhá přihlížející ke stupni daného vrcholu a třetí, která v experimentálním vyhodnocení dává nejlepší výsledky, do anomálního skóre započítává *eigveinvalue centrality* vrcholu v rámci shluku.

Podle vyhodnocení metoda udává, v porovnání s ostatními známými metodami pracujícími na podobném principu, velmi dobré výsledky.

2.3.4 FocusCO

Kategorie - Graf s atributy, založeno na shlukování

Metoda popsaná v článku [17] využívá cíleného shlukování uzlů grafu na základě uživatelem definovaných preferencí. Tím se výrazně liší od většiny ostatních metod, které obvykle pracují autonomně a spíše se snaží množství parametrů, které uživatel musí definovat minimalizovat. Tento postup, kdy má uživatel nad algoritmem větší kontrolu však může být velmi vhodný, v případě

Tabulka 2.4: FocusCO - vyhodnocení na základě def. kritérií

Schopnost detekce uzlu	Ano
Schopnost detekce hrany	Ne
Aplikace na podgraf	Ano
Výpočetní složitost	Lineární
Výstup	Ohodnocení uzlu binární hodnotou
Aplikace na běžná data	Ne
Typ aplikovatelných grafů	Statický, Atributovaný, Neorientovaný
Parametrizace algoritmu	Množina exemplárních uzlů

kdy o problémové doméně máme hlubší porozumění, které by autonomní algoritmus nemusel odhalit.

Uživatel jako vstup algoritmu může zadat sadu několika uzlů, které považuje za podobné těm, které ho zajímají. Algoritmus detekuje atributy, které jsou podobné pro jednotlivé exemplární uzly a podle jejich podobnosti jim přiřadí váhy (čím podobnější hodnoty atributů napříč exempláři, tím vyšší váha). Atributy s vahou přesahující určitou mez jsou označeny jako tzv. *focus atributy* a na základě nich je následně provedeno shlukování. Algoritmus se primárně zabývá uzly, které jsou výrazně podobné uživatelem zadaným uzlům a formuje shluky kolem nich.

Anomální uzly jsou poté definovány jako uzly, které strukturálně spadají do některého z vyhledaných shluků, ale svými *focus* atributy se od sousedů liší.

Výhodou cíleného shlukování je, že není nutné vždy analyzovat celý graf a je možné se zaměřit pouze na uzly, které uživatele zajímají, čímž se může výrazně snížit komplexita algoritmu. Časová složitost algoritmu roste lineárně s velikostí grafu.

2.3.5 OddBall

Kategorie - Graf bez atributů (vážený), založeno na vlastnostech grafu

Metoda je popsána v článku [3] a je učena pro detekci anomálních uzlů ve vážených grafech. Graf je zkoumán na úrovni jednotlivých egonetů a snaží se nalézt ty, které se nějakým způsobem vymykají od zbytku grafu.

Egonet je pojem převzatý z Analýzy sociálních sítí (angl. Social network analysis), definovaný jako podgrafu uzlů ve vzdálenosti 1 od centrálního uzlu společné se všemi hranami, které tyto uzly spojují. Centrálnímu uzlu se obvykle říká *ego*.

V článku je definováno několik vybraných metrik, které jsou pro jednotlivé *egonety* vyextrahovaných z grafu počítány a na základě kterých jsou *egonety* dále analyzovány. Konkrétně se jedna o tyto 4.

1. N_i počet vrcholů v *egonetu* odpovídající *egu i*
2. E_i počet hran v *egonetu* odpovídající *egu i*
3. W_i součet vah hran v *egonetu* odpovídající *egu i*
4. λ_i hlavní vlastní číslo vážené matice sousednosti v *egonetu* odpovídající *egu i*

Místo toho aby se tyto metriky zkoumaly jako celek, autoři článků popisují závislosti mezi dvojicemi těchto metrik získaných na základě pozorování grafů z reálného prostředí. Například závislost mezi váhou *egonetu* a počtem hran, definovaným jako

$$W_i \propto E_i^{\beta}, \beta \geq 1.$$

V podobném duchu jsou definovány další 3 závislosti, které se společně zaměřují na detekci některé ze 4 základních anomálií, které jsou

1. *Near-start*, podgrafy velmi blízké hvězdě
2. *Near-clique*, podgrafy velmi blízké klice
3. *Heavy vicity*, podgrafy s vysokou vahou v poměru se zbytkem grafu
4. *Dominant edge*, jedna hrana s abnormálně vysokou vahou vzhledem k počtu hran

Přesto že se jedná o poměrně jednoduchou metodu, stojí za jejím vznikem spousta analýzy grafů z reálného prostředí a pečlivého výběru vhodných metrik, které je možné využít pro detekci anomálních uzlů. Metoda má díky tomu poměrně dobré výsledky na grafech, které následují tato pozorování. Problém může nastat, pokud se v grafu nebude následovat definované závislosti, které jsou obvyklé například pro sociální sítě, ale nemusí být běžné pro jiné domény.

2.3.6 OutRank

Kategorie - Graf bez atributů (vážený), založeno na vlastnostech grafu

OutRank [18] je stochastický algoritmus, který při detekci anomálií využívá Markovův řetězový model (angl. Markov chain model), který staví nad analyzovaným grafem. Markovův řetězový model odpovídá náhodné procházce po grafu a myšlenka této metody je taková že pokud má objekt grafu nízkou provázanost vůči zbytku grafu, je s větší pravděpodobností anomálií.

Většina algoritmů se při detekci anomálií zaměřují na určité okolí jednotlivých uzlů v grafu, pracují nad celým grafem, ale vždy se probírají jednotlivými uzly a analyzují jejich blízké okolí, egonet, podobné sousedy apod. V tomto se

Tabulka 2.5: Oddball - vyhodnocení na základě def. kritérií

Schopnost detekce uzlu	Ano
Schopnost detekce hrany	Anomální hrany se projeví zvýšenou anomálností uzlů v jejich okolí, ale sami o sobě nejsou detekovány.
Aplikace na podgraf	Pouze pokud podgraf bude následovat definované závislosti stejně jako cely graf, což obvykle platí.
Výpočetní složitost	Lineární s velikostí grafu.
Výstup	Ohodnocení uzlu z intervalu $[0, \infty)$
Aplikace na běžná data	Ne
Typ aplikovatelných grafů	Statický, Vážený, Neorientovaný
Parametrizace algoritmu	Parametry exponentů pro výpočet anomálního skóre

od nich OurRank liší. Model založený na náhodné procházce definuje anomálnost objektu na základě vlastností celého grafu (nahlíží na anomálnost objektů z globální perspektivy).

Metoda je aplikovatelná i na běžná tabulková data a autoři článku se poměrně podrobně zabývají způsobem převodu těchto dat na graf. Konkrétně je tato konverze definována následovně.

- Objekty v datech jsou reprezentovány jako vrcholy grafu.
- Pro každou dvojici datových objektů je vypočítána jejich podobnost a je reprezentována jako váha hrany v grafu, mezi odpovídajícími vrcholy.

Volba podobnostní metriky není striktně daná, v článku se zabývají dvěma typy - kosinovou podobností a RBF podobností a následně jejich kvalitu pro tento problém diskutují ve vyhodnocení kvality metody.

Algoritmus graf reprezentuje jako váženou matici sousednosti, kterou převede na stochastickou matici. Postup lze rozdělit do následujících dvou kroků.

1. Převod matice sousednosti na stochastickou matici, ve které je součet každé řádky roven 1.
2. Výpočet stacionárního rozdělení matice.

Vrcholy s nejnižšími hodnotami ve stacionárním rozdělení jsou následně označeny jako anomální.

Díky tomu, že je metoda aplikovatelná na běžná tabulková data, bylo možné využít trénovací datasety, kterých je pro běžná data podstatně více něž pro grafy. Podle vyhodnocení v článku dává OutRank velice dobré výsledky.

Tabulka 2.6: OutRank - vyhodnocení na základě def. kritérií

Schopnost detekce uzlu	Ano
Schopnost detekce hrany	Ne
Aplikace na podgraf	Ne
Výpočetní složitost	Obecně vyšší než lineární
Výstup	Ohodnocení uzlu z intervalu $[0, 1]$
Aplikace na běžná data	Ano
Typ aplikovatelných grafů	Statický, Vážený, Neorientovaný
Parametrizace algoritmu	Parametr rozdílu dvou posledních výsledků pro zastavení algoritmu, damping faktor

2.3.7 SCAN

Kategorie - Graf bez atributů, založeno na shlukování

Algoritmus SCAN [19] se snaží o detekci speciálních druhů anomálií v grafu, které se nazývají *hub* a *outlier*.

Metoda pracuje na principu rozdělení uzlů do skupin. Pro tento účel definuje velmi efektivní algoritmus pro optimální detekci shluků v grafu. Algoritmus při shlukování grafů využívá okolí jednotlivých uzlů a uzly které sdílí velkou část svého okolí jsou shluknuty do stejné skupiny. Tím se nijak zásadně neliší od většiny ostatních postupů pro detekci shluků v grafech. Většina běžných postupů však nepočítá s anomáliemi, které obecně svou povahou mohou silně ovlivnit podobu nalezených clusterů, což je jejich hlavní rozdíl vůči algoritmu SCAN, který je navržen tak aby s anomáliemi implicitně počítal.

Dalším rozdílem SCAN je, že většina shlukovacích algoritmů jako vstupní parametr vyžaduje počet shluků do kterých má být graf rozdělen, což při hledání anomálií není obvykle známo a špatná volba by mohla mít výrazný vliv na kvalitu výsledku.

SCAN pracuje směrem zespoda nahoru. Nejprve hledá speciální uzly, které nazývá jádro (angl. core), na které postupně připojuje další uzly na základě jejich provázanosti a buduje tak jednotlivé shluky. Poté co jsou všechny uzly analyzovány a rozřazeny do shluků, jsou identifikovány 2 skupiny uzlů, které svou povahou nezapadají do zbytku grafu.

1. **Hub** je vrchol grafu, který propojuje dva nebo více různých shluků
2. **Outlier** je vrchol grafu, který nepatří do žádného shluku a ani není *Hubem*

Jak již bylo zmíněno, jednou z výhod algoritmu je, že není nutné definovat počet výsledných shluků avšak algoritmus není zcela bez vstupních parametrů.

Je nutné definovat dvojici parametrů μ, minimální počet podobných sousedů, který uzel musí mít aby mohl být označen jako *core* a ε spodní mez podobnosti 2 uzlů pro to aby mohli být považovány za podobné.

Tabulka 2.7: SCAN - vyhodnocení na základě def. kritérií

Schopnost detekce uzlu	Ano
Schopnost detekce hrany	Ne
Aplikace na podgraf	Ne
Výpočetní složitost	Lineární s velikostí grafu.
Výstup	binární označení uzlu
Aplikace na běžná data	Ne
Typ aplikovatelných grafů	Statický, Neorientovaný
Parametrizace algoritmu	Parametr po min. velikost shluku a spodní mez podobnosti

2.3.8 AutoPart

Kategorie - Graf bez atributů, založeno na shlukování

Aloritmus Autopart [20] detekuje anomálie v grafu na základě jejich shlukování podle vnitřní struktury grafu. Shlukuje k sobě uzly jejichž okolí jsou silně provázána. Hrany které nejsou v rámci jednoho shluku (propojují více shluků) jsou následně označeny za anomálie.

Obecný princip je algoritmu je tedy poměrně přímočarý, autoři však v článku popisují poměrně zajímavý přistup jakým v grafu detekují shluky. Graf je reprezentovaný pomocí $n \times n$ matice sousednosti. Shluky jsou hledány pomocí přehazování řádků a sloupců v grafu tak aby podobné uzly v matici sousedili a snaží se v ní vytvořit čtverce či obdélníky jedniček (jednička v matici reprezentuje hranu mezi dvěma uzly).

Algoritmus se v iteracích snaží v matici zkonstruovat co největší a co nejvíce homogenní čtverce. Tyto dva požadavky však jdou proti sobě, pokud by preferoval homogenní čtverce, bylo by nejvýhodnější n uzlů rozdělit do n^2 perfektně homogenních čtverců velikosti 1×1. Naopak pokud by byla preferována maximální velikost čtverců, budou tyto čtverce v sobě obsahovat spoustu nul, protože uzly v jednotlivých skupinách budou zároveň často sousedit s uzly z jiných skupin.

Pro vyrovnání tohoto *trade-off*u je použita *Minimum description length* (MDL), která se matici snaží komprimovat na co možná nejnižší počet bitů a algoritmus se iterativně snaží tuto hodnotu minimalizovat. Jak již bylo zmíněno, anomální hrany jsou ty, které propojují více skupin uzlů.

Tabulka 2.8: AutoPart - vyhodnocení na základě def. kritérií

Schopnost detekce uzlu	Ne
Schopnost detekce hrany	Ano
Aplikace na podgraf	Ne
Výpočetní složitost	Lineární
Výstup	Ohodnocení hrany binární hodnotou
Aplikace na běžná data	Ne
Typ aplikovatelných grafů	Statický, Neorientovaný
Parametrizace algoritmu	Není

Algoritmus má řadu výhod mezi ty nejvýznamnější patří:

- Možnost online dopočítávání v případě přidání dalšího uzlu s hranami. Je možné využít dosavadní výsledek, přidat do matice nový uzel a provést další iterace pro minimalizaci MDL.
- Kompletní bezparametričnost – nevyžaduje žádný vstup od uživatele
- Dobrá škálovatelnost, podle odhadu komplexity a experimentů v článku roste výpočetní složitost toho algoritmu lineárně s velikostí grafu.

2.3.9 NF and AD in bipartite Graphs

Kategorie - Graf bez atributů (bipartitní), založeno na shlukování

Tato metoda [21] se zabývá detekcí anomálií pouze v bipartitních grafech.

Definice 2.3. Bipartitní graf. Graf $G = (V, E)$ kde $V_1, V_2 \subseteq V$ taková že $V = V_1 \cup V_2, V_1 \cap V_2 = \emptyset$ a zároveň pro každou hranu $(u, v) \in E$ platí že $u \in V_1 \wedge v \in V_2$ nebo $v \in V_1 \wedge u \in V_2$, se nazývá bipartitní.

Jinými slovy jsou jeho uzly rozděleny do 2 skupin, přičemž propojení hranami nemůže existovat v rámci jedné skupiny.

Co do obecnosti použití je na tom tato metoda tedy podstatné hůře v porovnání s ostatními, v tomto textu uvedenými. Avšak v praxi je tento typ grafů poměrně běžný. Prakticky ve všech systémech kde spolu interoperují nějaké dvě skupiny entit, můžeme v nějaké formě pozorovat strukturu biparitního grafu. Například graf autorů vědeckých výzkumů a publikací na kterých se podíleli, zákazníci a hodnocení produktů, různé P2P systémy apod. Aplikací je opravdu mnoho a z toho důvodu je metoda zařazena do tohoto přehledu.

Algoritmus lze rozdělit do 2 základních fází (podle těchto fází je i pojmenován v tomto textu, sám o sobě algoritmus nemá žádné pojmenování). *Neighborhood formation (NF)* a *Anomaly detection (AD)*

Tabulka 2.9: NF and AD in bipartite Graphs - vyhodnocení na základě def. kritérií

Schopnost detekce uzlu	Ano
Schopnost detekce hrany	Ne
Aplikace na podgraf	Ne
Výpočetní složitost	Do výpočetní složitost algoritmu nejvíce přispívá výpočet *Neighborhood formation*, v případě použití aproximativního algoritmu je složitost lineární
Výstup	Ohodnocení uzlu z intervalu $[0, \infty]$
Aplikace na běžná data	Ne
Typ aplikovatelných grafů	Statický, Bipartitní, Neorientovaný
Parametrizace algoritmu	Není

Neighborhood formation

První fáze na kterou lze nahlížet jako na shlukování v bipartitních grafech. Cílem NF je pro každý vrchol ve skupině V_1 vypočíst jeho relevanční skóre vůči všem ostatním uzlům ve skupině V_1. Čím vyšší toto skóre je tím si jsou podobnější.

Algoritmus kalkulující tato skóre je založen na principu náhodné procházky a pro každý uzel a z V_1 počítá kolikrát jsou navštíveny všechny ostatní uzly z V_1, pokud procházka začala v a. Jinými slovy počítá pravděpodobnost návštěvy jednotlivých uzlů při startu z jednoho konkrétního.

V článku popisují dva různé postupy pro výpočet NF, jeden exaktní ale s vysokou výpočetní náročností a druhý rychlejší ale pouze aproximující.

Anomaly detection

Na základě vypočítaných relevančních skóre jednotlivých uzlů z V_1 v první fázi algoritmu lze následně vypočíst anomální skóre uzlů v V_2.

Zjednodušeně se anomální skóre pro každý uzel t z V_2 určí tak, že se extrahuje množina uzlů S_t obsahující uzly z V_1 sousedící s t. Následně se porovnají relevanční skóre mezi jednotlivými uzly v S_t. Intuitivně by si tyto uzly v S_t měli být navzájem podobné, jelikož jsou všechny společně propojeny uzlem t z opačné skupiny. Vysoká podobnost uzlů je indikována vysokým vzájemným relevančním skóre a tedy uzel t je normální pokud vzájemná relevanční skóre uzlů z S_t je vysoké.

Vyhodnocení algoritmu je v článku prováděno na několika reálných bipartitních grafech, ve kterých anomálie nejsou označeny, společně se syntetickém zanesením anomálií. Výhodou bipartitních grafů vůči obecným je že díky jejich menší komplexitě je snazší výsledky detekce anomálií intuitivně interpretovat

a výsledky které tato metoda vykazuje se v experimentech projevili jako poměrně kvalitní.

2.4 Shrnutí

V předchozí sekcích této kapitoly byly uvedeny různé druhy postupů pro detekci anomálií v grafech. Vzhledem k tomu, že by bylo velmi obtížné pokrýt všechny různé techniky, vážící se k této problematice, zaměřil jsem se především na ty nejdůležitější a nejzajímavější.

Jednotlivé metody byly rozděleny do čtyř hlavních kategorií na základě dvou kritérií. Na jaký typ grafu je lze aplikovat (atributovaný graf × bez atributů) a jakým způsobem k problému přistupuje (detekce na základě metrik získaných z grafu × na základě shlukování a komunit v grafu). Přehled kategorizace metod je uveden v tabulce 2.10.

Tabulka 2.10: Kategorizace metod

	Vlastnosti grafu	**Shlukování**
Bez atributů	OddBall 2.3.5 OutRank 2.3.6	SCAN 2.3.7 AutoPart 2.3.8 NF and AD in BipG 2.3.9
S atributy	Subdue 2.3.1 GDAB 2.3.2	GOutRank 2.3.3 FocusCO 2.3.4

I když to z tohoto přehledu nemusí být zřejmé, větší pozornosti (alespoň co do počtu existujících metod) se věnuje postupům zaměřených na shlukování entit v grafu do shluků a komunit.

To je pravděpodobně způsobeno tím, že pro detekci shluků v grafech již existuje mnoho algoritmů, které se na problém detekce anomálií dají aplikovat. Avšak výrazně rozdílné entity v grafu mohou mít silný dopad na formaci shluků v grafu nebo naopak anomálie které dobře imitují normální chování v grafu mohou být zařazeny do některého ze shluků bez povšimnutí. To má obvykle neblahý dopad na výsledky při použití běžných algoritmů pro shlukování jako prostředek pro detekci anomálií a metody které tyto algoritmy používají bez jejich úpravy a adaptace pro tuto problematiku většinou nedosahují vysoké efektivity.

Druhým důvodem mohou být sociální sítě, které se stali fenoménem posledního desetiletí a co se produkování grafových dat týče je možné se s klidným svědomím tuto oblast označit za jednoho z největších producentů. Uživatelé sociálních sítí svým chováním přirozeně vytvářejí shluky a komunity, na čemž některé z metod přímo staví. Lze celkem intuitivně předpokládat, že jedinci kteří nejsou členem žádného shluku, nebo přemosťují více shluků jsou do jisté míry anomální (toho využívají např. SCAN či AutoPart) a nebo že uzly které

jsou z pohledu struktury členem nějakého shluku, ale svými atributy se od ostatních vymykají jsou pravděpodobně také abnormální.

Tabluka 2.11 obsahuje přehledné porovnání vlastností jednotlivých metod. Význam jejích sloupců je následující: Uzel - zda je metoda schopná detekovat anomální uzel, Hrana - zda je metoda schopná detekovat anomální hranu, Podgraf - zda je metoda snadno aplikovatelná na podgraf zadaného grafu, Lineární složitost - zda časová komplexita roste lineárně s rostoucí velikostí grafu, Výstup - popis výstupu metody, Běžná data - zda metodu lze aplikovat na běžná tabulková data, s tím že jsou nejprve jasně definovaným způsobem transformována do grafu, Parametrizace - zda algoritmus vyžaduje zadané parametry od uživatele.

Tabulka 2.11: Porovnání metod

Algoritmus	Uzel	Hrana	Podgraf	Lineaární složitost	Výstup	Běžná data	Parametrizace
Subdue	A	N	A	N	Ohod. podgrafu z $[0, 1]$	A	N
GDAB	A	N	N	N	Ohod. podgrafu z $[0, \infty]$	A	N
GOutRank	A	N	A	-	Ohod. uzlu z $[0, \infty]$	N	N
FocusCO	A	N	A	A	Bin. ohod. uzlu	N	Množina exemplárních uzlů
OddBall	A	N	A	A	Ohod. uzlu z $[0, \infty]$	N	Parametry exponentů pro výpočet anomálního skóre
OutRank	A	N	N	N	Ohod. uzlu z $[0, 1]$	A	Ukončovací parametr, duping factor
SCAN	A	N	N	A	Bin. ohod. uzlu	N	Min. velikost shluku, spodní mez podobnosti
AutoPart	N	A	N	A	Bin ohod. hrany	N	N
NF and AD in Bip G	A	N	N	A	Ohod. uzlu z $[0, \infty]$	N	N

Kapitola 3

Analýza

Tato kapitola je zaměřena na analýzu použitých nástrojů a návrhem modulů, které budou součástí implementace výsledné knihovny. Při implementaci bude využit framework Gephi Toolkit, na který je primárně zaměřena analýza v této kapitole a na základě jeho schopností budou následně definovány požadavky kladené na výslednou knihovnu a její moduly.

3.1 Výběr metod pro implementaci

V předchozí kapitole je představeno a popsáno několik metod, řešící problém detekce anomálií ve statických grafech. Nyní je nutné vybrat některé z nich, které budou implementovány jako součást výsledné knihovny. Při výběru je dbán důraz především na efektivitu algoritmu co do kvality výsledných řešení a efektivitu co do výpočetní složitosti, analyzované grafy mohou dosahovat velikostí statisíců uzlů a použít nad nimi metodu s exponenciální složitostí nepřichází k úvahu. Dále je také brán zřetel na to jak jsou jednotlivé metody populární a jak často jsou tyto metody zmiňovány v ostatních vědeckých pracích a jejich výsledky porovnávány s ostatními.

Rád bych vybral jednoho zástupce metod, které řeší problém na základě detekce metrik v grafu a jednoho zástupce který využívá shlukování dle tabulky 2.10.

Volba metody, která využívá metrik grafů je poměrně snadná. Algoritmus **OddBall** je velmi vhodný kandidát jak z pohledu jeho vlastností - je možné jej použít na podgraf, jeho složitost je lineární, poskytuje kvalitní výsledky apod. - tak i z pohledu popularity - článek popisující tuto metodu je často odkazovaný v ostatních pracích.

Při volbě shlukových metod je to podstatně těžší, žádná metoda tak výrazně nevyčnívá nad ostatní jako v případě OddBall. Navíc jsem v tomto případě byl nucen přihlédnou i k tomu jak kvalitně jsou metody v článku popsány, protože některé popisy jsou poměrně dost vágní a implementace na základě nich by byla poměrně složitá. Z těchto důvodů jsem nakonec zvolil

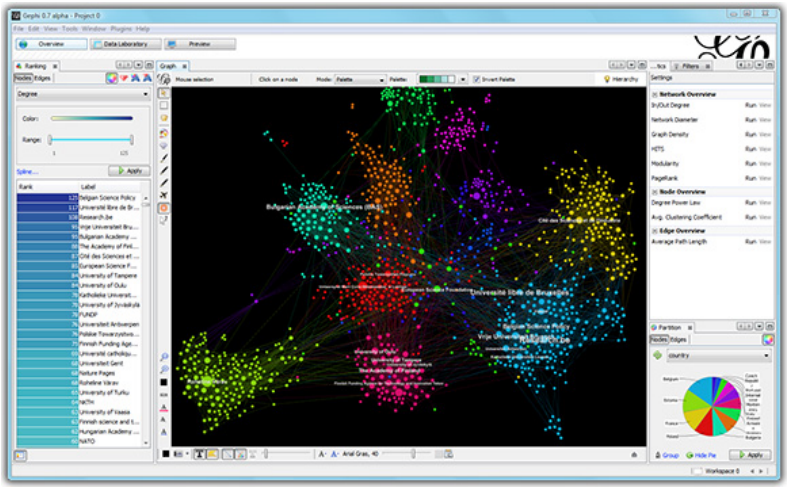
algoritmus **SCAN**. Jeho vlastnosti možná nejsou tak dobré - nelze jednoduše aplikovat na podgraf, vyžaduje parametrizaci - ale jeho výsledky jsou kvalitní a metoda je v článku dobře popsána.

3.2 Analýza technologií

3.2.1 Gephi

Gephi[22] je open-source software určený pro vizualizaci a analýzu sítí, komplexních systémů, dynamických a hierarchických grafů. Nástroj je volně dostupný a zcela bezplatný. První veřejně dostupná verze vznikla v roce 2008, původně jako výsledek práce francouzských studentů na *University of Technology of Compiègne*. V roce 2010 vzniklo Gephi Concortium, které se nyní stará o vývoj a podporu Gephi.

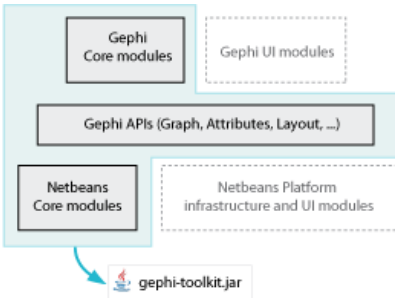
Nástroj je kompletně napsán v jazyce Java a postaven na rozšířitelné modulární architektuře NetBeans platform. Díky tomu je snadné přidávat novou funkcionalitu, momentálně existuje zhruba 60 pluginů, které doplňují základní funkce. Před půl rokem v roce 2017 vyšla verze číslo 0.9, které byla významným krokem ve vývoji Gephi, obsahující změny v samotném jádře aplikace a není plně zpětně kompatibilní s předchozími verzemi. Z toho důvodu se většina dostupných pluginů stala nefunkčními v nové verzi a bylo je nutné je přepsat. Jejich momentální počet se zhruba 40. Přesto je Gephi v současnosti široce používaným software.



Obrázek 3.1: Ukázka grafického uživatelského rozhraní Gephi[1]

3.2.2 Gephi Toolkit

Gephi je aplikace určená pro vizualizaci sítí. Poskytuje uživateli grafické rozhraní a mnoho modulů obsahující logiku pro práci s ním. Jedná se tedy o samo-



Obrázek 3.2: Schámatiká struktura Gephi Toolkit

statnou aplikaci, kterou není možné použít v jiných projektech. Z toho důvodu vývojáři Gephi poskytují tzv. Gephi Toolkit [23] obsahující pouze esenciální moduly pro práci s grafy a který je jednoduše distribuován jako jeden .jar soubor. Díky tomu jej lze jednoduše přidat do libovolného Java projektu jako knihovnu a využívat tak s veškerými funkcemi, které Gephi poskytuje.

Jak již bylo řečeno Gephi je navrženo jako modulární aplikace. Každá funkcionalita je zabalena do svého separátního modulu, například modul pro grafovou strukturu, modul pro algoritmus upravující rozložení uzlů v grafu apod. Navíc moduly obsahující bussiness logiku jsou odděleny od těch pro uživatelské rozhraní, což je schématicky nastíněno na obrázku 3.2. To umožňuje využít pouze bussiness logiku a odstranit UI. Toto je přesně účel Gephi Toolkitu.

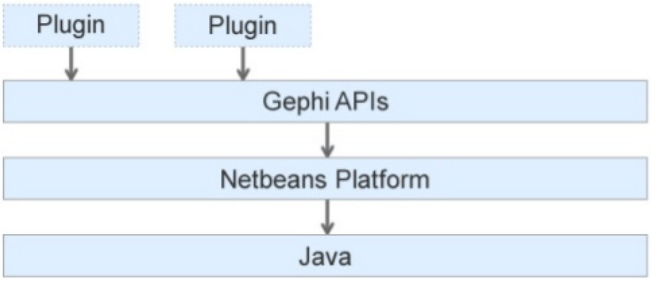
3.2.3 NetBeans platform

Gephi je postaveno na *NetBeans platform*, což je framework pro vývoj modulárních aplikací, který je vyvíjen od roku 1996 v rámci NetBeans projektu. V aplikacích postavených na tomto frameworku, je každá funkcionalita oddělena do svého modulu, které vzájemně komunikují prostřednictvím veřejných API. Každý modul může mít své grafické uživatelské rozhraní a nebo může poskytovat čistě funkcionalitu, která probíhá na pozadí bez grafického výstupu.

Gephi obsahuje oba typy modulů, jak ty s GUI tak i ty bez něj. Gephi Toolkit je v podstatě derivát z Gephi, který obsahuje pouze ty moduly, které nedisponují grafickým rozhraním. Díky tomu je možné jej bez problémů importovat do svého projektu jako knihovnu bez návaznosti na GUI Gephi.

Samotná NetBeans platforma je rozdělená do několika modulů. Základním kamenem aplikace postavené na NetBeans platformě je *Runtime Container*, který je zodpovědný za běh aplikace. Skládá se ze šesti modulů: *Startup*, *Bootstrap*, *Filesystem API*, *Module System API*, *Lookup API* a *Utilities API*.

Tím nejzajímavějším modulem pro účely této práce je *Lookup API*, které zajišťuje vazbu mezi jednotlivými moduly a umožňuje jim vzájemně komunikovat. Pro tuto komunikaci modul nejprve musí definovat veřejné rozhraní.



Obrázek 3.3: Propojení NetBeans platform a Gephi[2]

Lookup API poté poskytuje přistup k tomuto rozhraní a jeho implementace může být získána pomocí statické třídy `Lookup`.

Na obrázku 3.3 je schématicky znázorněna architektura Gephi a jakým způsobem využívá *NetBeans platform*.

3.2.4 Rozhraní Gephi modulů

Veškerá funkcionalita Gephi a tedy i Gephi Toolkit je rozdělena do vzájemně spolupracujících modulů. Gephi poskytuje několik rozhraní, které každý z modulů v něm obsažených implementuje. Při přidání nových modulů je nutné nejprve vybrat některá rozhraní, které nejlépe vyhovují potřebám a implementovat je ve svém modulu. Dostupná rozhraní jsou následující:

- **Graph API** [24] - Definuje základní strukturu grafu. Poskytuje kompletní sadu funkcí pro práci s grafem: přístup k uzlům, přistup k hranám, získání základních metrik z grafu, strukturální úpravy a mnoho dalšího.

  Interně nerozlišuje mezi různými druhy grafu (neorientovaný, neorientovaný, hierarchický, ...), ale umožňuje zvolit si pohled jakým je na graf nahlíženo, tudíž nejsou potřeba žádné konverze.

- **Layout API** [25] - Poskytuje kontrolu nad rozložením grafu. Umožňuje změny provádět real-time inkrementálně a uživatel může pozorovat jak se rozložení grafu mění v průběhu času.

- **Attributes API** [26] - Rozhraní umožňující přiřazení libovolných atributů uzlům a hranám grafu.

- **Statistics API** [27] - Poskytuje možnost asynchronního spuštění statistického algoritmu nad grafem. Výsledek může být zobrazen buď formou grafického reportu v okně Gephi nebo vizualizací atributů grafů, které algoritmus přiřadil.

- **Import API** [28] - Importování grafových dat z různých zdrojů. Mezi nejčastěji používané zdroje patří soubory v různých formátech (csv, gefx,...), streamy nebo databáze.
- **Export API** [29] - Exportování grafových dat, obvykle do souborů různých formátů. Výsledný graf lze exportovat i ve vizuální formě jako obrázek například v pdf, png nebo svg.
- **Tools API** [30] - Definují operace, které uživatel může interaktivně aplikovat na graf.
- **Filters API** [31] - Poskytuje funkce pro vytvoření podgrafu. Umožňuje definovat sadu filtrovacích dotazů nad grafem, které z něj odstraní elementy, které dotazu neodpovídají a výsledek vrátí jak podgraf.
- **Generator API** [32] - Rozhraní pro generování grafů. Tvorba grafu na základě zadaných parametrů.
- **Project API** [33] - Umožňuje manipulaci s projektem a pracovním prostorem (workspace). Umožňuje přístupy k libovolným modulům a umožňuje sdílení dat mezi nimi. Většina modulů toto rozhraní využije při sledování životního cyklu aplikace.
- **LongTask API** [34] - Rozhraní určeno pro moduly, které provádějí dlouhotrvající operace. Poskytuje bezpečný způsob pro spuštění úlohy ve vlákně na pozadí s nativní vizualizací průběhu a možnosti přerušení. Zároveň také poskytuje možnost sledování a vypořádání se s chybami v průběhu úlohy.

Moduly pro detekci anomálií implementované ve výsledné knihovně nejvíce odpovídají Gephi statistikám a tedy by měli definovat **Statistics API**. Dále jelikož algoritmus může pracovat delší dobu, měli by moduly také definovat rozhraní **LongTask API**.

3.2.5 Gephi architektura

Funkcionalita Gephi je rozdělena do tzv. *Controllerů* (co modul to jeden *controller*). Controller je singletone třída a může být získán pomocí `Lookup`: `Lookup.getDefault()`. `Lookup` je tedy v podstatě pouze kontejner, který obsahuje jednotlivé moduly. Ostatní moduly mohou být vyhledány pomocí jejich jména třídy. Např.:

```
ProjectController pc = Lookup.getDefault()
    .lookup(ProjectController.class);
pc.newProject();
Workspace workspace = pc.getCurrentWorkspace();
```

V ukázce kódu je naznačena práce s **ProjectController**, který umožňuje tvorbu projektů a *workspaces*. Projekt musí být vždy vytvořen a je pouze jeden. Vytvořením projektu se automaticky vytvoří i *workpsace*, která obsahuje veškerá data.

Další kontrolery odpovídají rozhraním popsaným v sekci 3.2.4. V Gephi tedy můžeme nalézt např. *LayoutController*, *AttributesController*, *GraphController*, ...

3.3 Definice požadavků

Na jednotlivé implementované metody, které budou součástí výsledné implementované knihovny je kladeno několik společných požadavků. Většina těchto požadavků vychází z Gephi Toolkit, jeho architektury a rozhraní která poskytuje.

3.3.1 Funkční požadavky

1. Každý modul implementuje metodu pro řešeni detekce anomálií ve statických grafech.
2. Algoritmům je při spuštění možné předat sadu potřebných uživatelských parametrů.
3. Výsledkem algoritmu budou jasně označené anomálie, nacházející se v grafu. Způsob označení anomálií vychází z konkrétního aplikovaného algoritmu (ohodnocení uzlů/hran anomálním skóre, binární hodnota, ...).
4. Knihovna by měla být schopna číst vstupní data ze vstupního souboru a výsledky zapsat do výstupního souboru.

3.3.2 Nefunkční požadavky

1. Při implementaci bude použit framework Gephi Toolkit.
2. Každý algoritmus bude implementovat rozhraní připravená v Gephi Toolkit.
3. Algoritmy budou implementovány v programovacím jazyku Java.
4. Knihovna bude distribuována jako jeden .jar soubor.
5. Výsledek algoritmu by měl být ve strojově čitelné podobě, umožňující další zpracování.

Kapitola 4

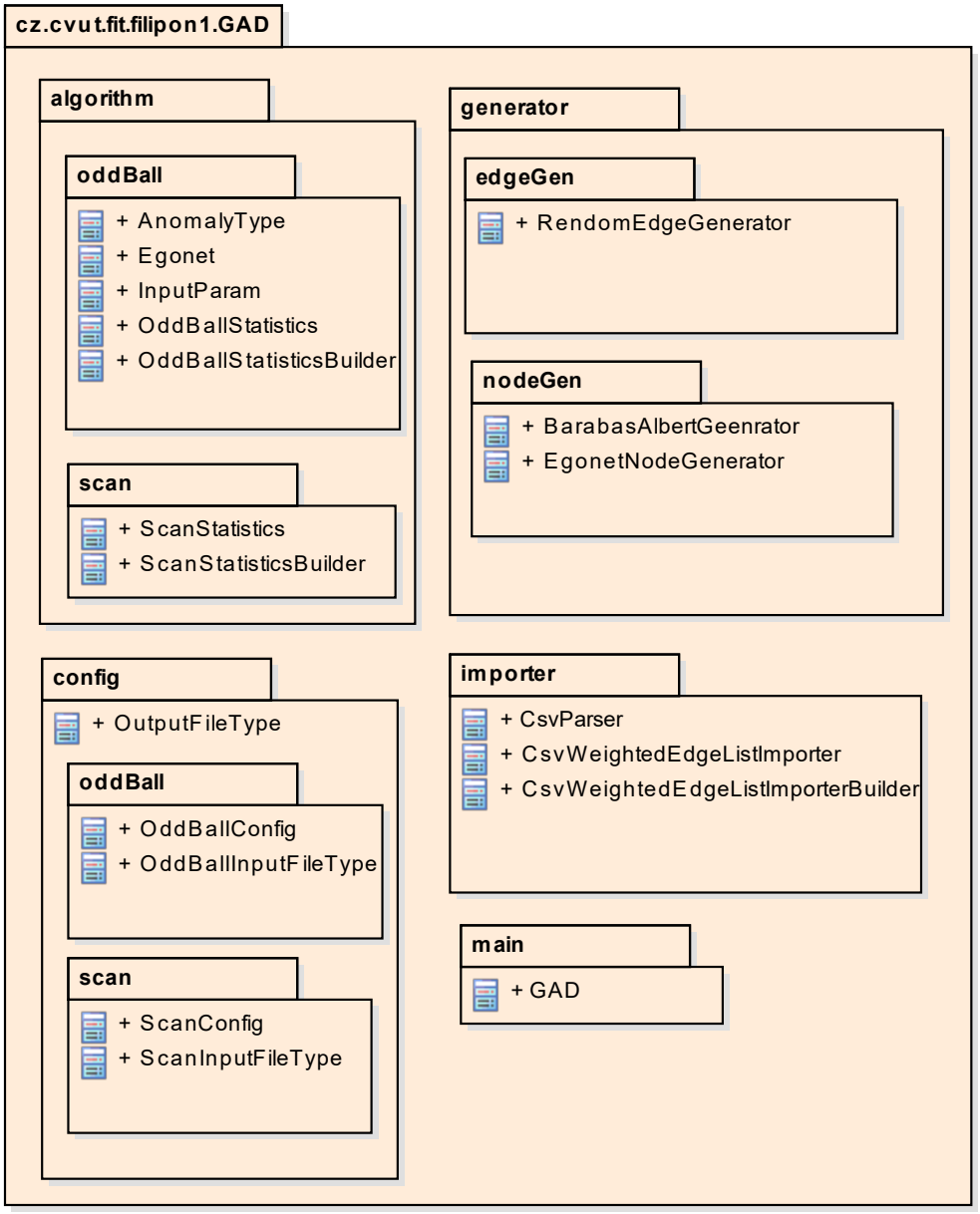
Návrh a realizace

Tato kapitola se zabývá návrhem a implementací knihovny implementované pomocí frameworku Gephi Toolkit, která obsahuje metody pro detekci anomálií popsané v předchozí kapitole. Pracovní název knihovny je GAD (z anglického *graph anomaly detection*). Nejprve je popsána struktura knihovny jako celku a jakým způsobem v ní je využit Gephi Toolkit. Následně je nastíněn společný základ jednotlivých modulů. Nakonec je popsán návrh a realizace zvolených metod, konkrétně OddBall a SCAN. V knihovně je také obsažena vlastní navržená metoda pro detekci anomálií v grafech, jejíž návrhu a realizaci je však věnována samostatná kapitola.

4.1 Struktura knihovny

Tato sekce se věnuje popisu implementované knihovny jako celku a její struktury. Při návrhu jednotlivých modulů pro detekci anomálií v grafech byl kladen důraz na to aby správně reflektovaly strukturu předepsanou frameworkem Gephi Toolkit a bylo je případně možné snadno rozšířit a vytvořit z nich plugin určený pro Gephi včetně uživatelského rozhraní. Moduly proto implementují všechna potřebná rozhraní, která byla popsána v sekci 3.2.4 a v případě jejich rozšíření do podoby Gephi pluginu by bylo nutné doplnit pouze implementaci spojenou s grafickým uživatelským rozhraním.

Aby bylo možné knihovnu importovat do vlastního projektu jako .jar a snadno i používat bez toho aby bylo nutné podrobně znát funkcionalitu Gephi, bylo nutné vytvořit několik vlastních tříd pro přístup k modulům, načtení vstupních dat a jejich transformace do podoby potřebné pro Gephi, inicializaci algoritmů apod. Na obrázku 4.1 je možné si prohlédnou jejich strukturování do packagů projektu. Nyní se na ně podívejme podrobněji.



Obrázek 4.1: Struktura balíčků knihovny GAD

4.1.1 Package algorithm

Tento package obsahuje moduly s implementací zvolených metod pro detekci anomálií v grafech. Podrobně se jim budu věnovat v samostatných sekcích.

4.1.2 Package generator

Package `generator` obsahuje několik algoritmů určených pro generování uzlů a hran do existujících grafů. Tyto algoritmy se používají při testování a bude jim věnována větší pozornost v kapitole o testování.

4.1.3 Package importer

Package `importer` obsahuje Gephi modul pro importování dat a jejich transformace do vhodné podoby pro další zpracování. Algoritmus OddBall na vstupu vyžaduje graf s váženými hranami a Gephi samo o sobě je schopné importovat csv soubor grafu ve kterém hrany obsahují váhy. V Gephi Toolkit se mi však tuto funkcionalitu nepodařilo najít. Z toho důvodu byl vytvořen modul pro jejich import nazvaný `CsvWeightedEdgeListImporter`.

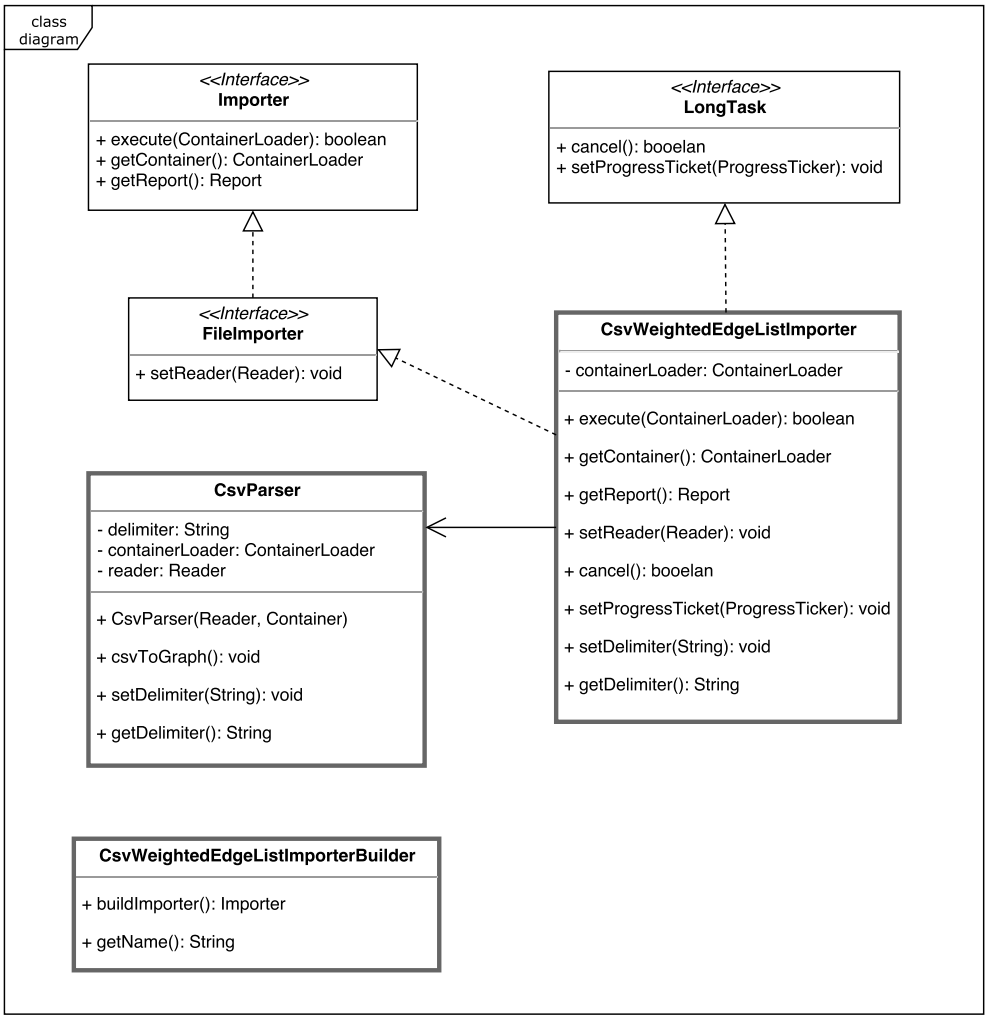
4.1.3.1 CsvWeightedEdgeListImporter

Načítaná data jsou očekávana v běžném CSV formátu, kde první sloupec obsahuje identifikátor zdrojového uzlů, druhý sloupec je identifikátor cílového uzlu a třetí sloupec obsahuje váhu hrany. Soubor může obsahovat i další sloupce ale ty jsou při importu ignorovány. Modul se skládá ze tří tříd, jejich struktura je na obrázku 4.2:

`CsvWeightedEdgeListImporterBuilder` je builder třída využívaná pro tvorbu instance importéru. Je anotovaná anotací *ServiceProvider* a díky tomu ji Gephi je schopné vyhledat pomocí `Lookup API` a poskytnout její implementaci.

`CsvWeightedEdgeListImporter` implementuje Gephi rozhraní `FileImporter`, které vyžaduje implementaci několika metod.

- `void setReader(Reader reader)` - Slouží k nastavení zdroje pro čtení dat.
- `ContainerLoader getContainer()` - Data nejsou načítána přímo do grafu, ale do třídy `ContainerLoader`, která poskytuje metody pro tvorbu nového uzlu/hrany, umožňuje definovat jakým způsobem má být na graf nahlíženo (orientovaný graf, orientovaný,...) a následně je předána třídě `ImportController`, který zařídí načtení nastřádaných elementů do grafu.



Obrázek 4.2: Class diagram CsvWeightedEdgeListImporter

- `boolean execute(ContainerLoader cl)` - Metoda pro spuštění samotného importu.

`CsvParser` je třída obsahující veškerou logiku spojenou s převodem načtených dat do grafové struktury.

4.1.4 Package config

Package `config` obsahuje konfigurační třídy určené k parametrizaci algoritmů při jejich spuštění. Každý algoritmus pro detekci anomálií má svou konfigurační třídu. Všechny konfigurace mají společný základ. Tím je nastavení vstupního a výstupního souboru, společně s nastavením jejich formátů. Dále každá umožňuje nastavit uživatelské vstupní parametry využívané v algoritmech.

4.1.5 Třída GAD

Třída `GAD` tvoří interface celé knihovny a propojuje všechny její části. `GAD` se stará o načítání dat ze souboru, inicializaci algoritmů a jejich spuštění, zápis výsledků apod.

Hlavní funkcí této třídy je spouštění jednotlivých algoritmů pro detekci anomálií. Rozhraní pro jejich spuštění je poměrně jednoduché skládající se z jediné metody pro každý z algoritmů. Tato metoda vyžaduje předání příslušné konfigurační třídy podle které jsou nastaveny parametry algoritmu a jsou z ní získány vstupní a výstupní soubory, včetně jejich formátu.

Spuštění algoritmu SCAN může vypadat následujícím způsobem:

```
GAD gad = new GAD();
ScanConfig config = new ScanConfig()
        .setInput(new File("/path/to/file.csv"),
                ScanInputFileType.CSV)
        .setOutput(new File("/path/to/file.dexf"),
                OutputFileType.DEXF)
        .setMinSimilarityThreshold(0.42)
        .setMinSizeOfCluster(3);
gad.runScanStatistics(config);
```

Na začátku je vytvořena instance třídy `GAD`. Následně je vytvořena instance třídy `ScanConfig` s konfigurací běhu algoritmu. Postupně ji jsou nastaveny hodnoty pro vstupní soubor, výstupní soubor a parametry algoritmu. Nakonec stačí zavolat metody pro start běhu algoritmu v tomto případě `gad.runScanStatistics(config)`. Spuštění ostatních algoritmů je analogické.

4.2 Společné rysy modulů

Tato sekce se zaměřuje na to co mají jednotlivé modul v knihovně společné. Tento společný základ vychází především z použitých technologií, tedy primárně z Gephi Toolkit.

4.2.1 Statistics API

Jak již bylo zmíněno v předchozí kapitole, *Statistics API* poskytuje možnost asynchronního spuštění statistického algoritmu nad grafem. Výsledek může být zobrazen buď formou grafického reportu v okně Gephi nebo vizualizací atributů grafů, které algoritmus přiřadil.

Tím se stává jasným kandidátem pro rozhraní, který by modul pro detekci anomálií v grafech měl implementovat. Jelikož knihovna bude využívat pouze Gephi Toolkit funkcionalitu a nebude tedy mít přístup ke grafickému rozhraní Gephi, bude výsledek předán pouze jako atributy elementů grafu.

4.2.1.1 Statistics

Třída implementující algoritmus musí implementovat rozhraní `Statistics`. Toto rozhraní obsahuje dvě metody:

- `void execute(Graphmodel graphModel)` - Metoda obsahující samotný běh algoritmu. Jako parametr dostane `GraphModel` nad kterým výpočet provádí.
- `String getReport()` - Vrací report, který je získán po skončení běhu algoritmu a je zobrazen v panelu grafického rozhraní Gephi. V tvořené knihovně nemá hodnota vracená touto metodou moc velký smysl, protože se nebude nikde zobrazovat.

4.2.1.2 StatisticsBuilder

`StatisticsBuilder` je třída, která se využívá při tvorbě nové instance statistiky a drží informace o celém modulu. Pomocí anotace *SrviceProvider* informuje Gephi o tom že poskytuje funkcionalitu pro statistickou funkci a může být vyhledána pomocí `Lookup API`. Obsahuje tyto metody:

- `String getName()` - Vrací název statistiky.
- `Statistics getStatistics()` - Vrací novu instanci dané statistiky.
- `Class<? extends Statistics> getStatisticsClass()` - Vrací třídu dané statistiky.

4.2.2 LonkTask API

Rozhraní určeno pro moduly, které provádějí dlouhotrvající operace. Poskytuje bezpečný způsob pro spuštění úlohy ve vlákně na pozadí s nativní vizualizací průběhu a možnosti přerušení. Zároveň také poskytuje možnost sledování a vypořádání se s chybami v průběhu úlohy.

4.2.2.1 LongTask

Rozhraní `LongTask` by měla implementovat každá ze tříd obsahující algoritmus. Toto rozhraní obsahuje dvě metody:

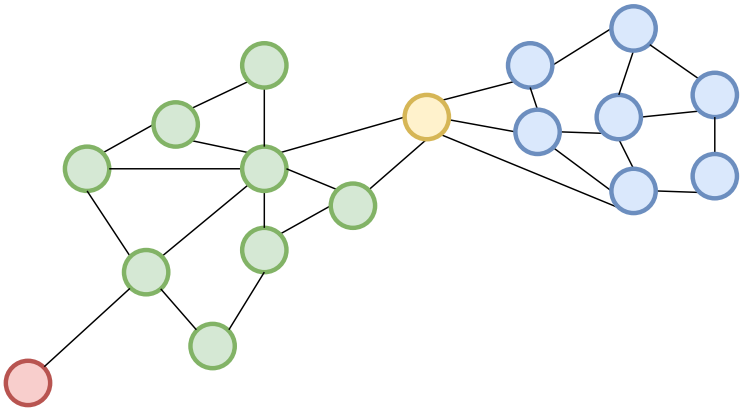
- `boolean cancel()` - Přerušení běhu metody z vnějších zdrojů, obvykle přerušení uživatelem.
- `void setProgressTicket(ProgressTicket progressTicket)` - Pomocí parametru `ProgressTicker` lze předávat informaci o aktuálním průběhu algoritmu.

4.3 SCAN

4.3.1 Popis metody

Shlukování v grafech je jedním ze základních způsobů pro detekci skrytých struktur v grafu a je používáno v mnoha různých oblastech počítačové vědy. Obecný princip je poměrně jednoduchý, zakládá se na detekci silně provázaných uzlů, které pospolu tvoří tzv. shluk (angl. cluster). Existuje řada metod kterými lze shluky v grafu nalézt, avšak tyto metody obvykle nerozlišují jakou roli jednotlivé uzly v grafu mají.

Na obrázku 4.3 inspirovaným z [19] je znázorněno několik různých druhů uzlů, které se v grafu mohou vyskytovat. Graf na obrázku očividně obsahuje dvě silně provázené skupiny uzlů neboli shluky, které jsou obarveny zeleně a modře. Zbylé dva uzly jsou speciální. Žlutě obarvený uzel by se dal považovat za součást obou shluků a tvoří přemostění mezi nimi. Tento typ uzlu je nazýván *hub*. Poslední červeně obarvený uzel nelze přidat do žádného ze shluků a zároveň ani neslouží jako *hub*. Tento typ uzlu je označen jako *outlier*.



Obrázek 4.3: SCAN - graf obsahující *outlier*, *hub* a 2 shluky

Při použití metod pro shlukování, které nepřihlížejí k těmto různým typům uzlů, by výsledkem byli pravděpodobně pouze 2 shluky jeden obsahující všechny zelené uzly společně s červeným a druhý který by obsahoval zbytek. Toto může pro některé některé aplikace v pořádku, v jiných však uzly se speciálními typy mohou hrát klíčovou roli.

Algoritmus SCAN se snaží tyto typy uzlů identifikovat tak, jak je popsáno na obrázku. Přichází proto s novým způsobem detekce shluků, který staví především na podobnosti v sousedství dvou uzlů. Čím více společných sousedů dva uzly mají tím je jejich šance, že ve výsledku budou součástí stejného shluku vyšší. Při uvažování například sociálních sítích, dává tato myšlenka smysl. Lide kteří mají mnoho společných přátel tvoří komunitu a čím více společných přátel mají tím více svázaná tato komunita je. Naopak někteří lidé

nemají tolik přátel a proto nejsou součástí žádné komunity by byli považování za *outlier*. Nakonec tu jsou lidé, kteří mají mnoho přátel z různých komunit ale do žádné ve skutečnosti nepatří (například politici), ti by byli považováni za *hub*.

4.3.2 Vstupní/výstupní parametry

Algoritmus je omezen na neorientovaný graf bez vah. Vyžaduje dva uživatelsky zvolené parametry:

- ε - určuje minimální hranici podobnosti takovou aby dva uzly mohli být považovány za podobné.
- μ - určuje minimální velikost shluku.

Výstupem je ohodnocení všech uzlů. Ohodnocení může nabývat 3 hodnot: *hub*, *outlier* nebo uzel obsažen ve shluku.

4.3.3 Postup algoritmu

Nejprve jsou všechny uzly neoznačené. Algoritmus postupně prochází všechny uzly a identifikuje mezi nimi tzv. *core* (jádro), které se stávají základním kamenem shluku, na který se postupně nabalují ostatní sousední uzly.

Před tím než je možné přejít k samotným krokům algoritmu je nutné definovat několik pojmů.

Definice 4.1. Strukturální podobnost.

$$\sigma(u, v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{\sqrt{|\Gamma u||\Gamma v|}}$$

Kde u a v jsou uzly grafu $G = (V, E)$ a $\Gamma(u)$ je definována jako množina sousedních uzlů uzlu u, formálně

$$\Gamma(u) = \{w | v, w \in E\} \cup u$$

Jak již bylo řečeno, SCAN stojí na podobností sousedství dvou uzlů. Strukturální podobnost je základní metrika, kterou využívá pro vyjádření podobnosti dvou uzlů. Čím více podobných sousedů budou mít tím vyšší bude jejich strukturální podobnost. Tuto podobnost můžeme zdola omezit parametrem ε a o dvou uzlech přesahující tuto hodnotu prohlásit, že patří do stejného ε-sousedství

Definice 4.2. ε-sousedství.

$$N_\varepsilon = \{w \in \Gamma(v) | \sigma(v, w) \geq \varepsilon\}$$

kde v a w jsou uzly z grafu $G = (V, E)$ a $\varepsilon \in \mathbb{R}$.

Pokud je mohutnost sousedství nějakého uzlu dostatečně velké, stává se počátkem shluku a je nazýván *core*. „Dostatečně velké“ je určeno druhým uživatelem zadaným parametrem μ.

Definice 4.3. ***Core***

$$CORE_{\varepsilon,\mu}(v) \Leftrightarrow |N_\varepsilon(v)| \geq \mu$$

kde v je uzel z grafu $G = (V, E)$, $\varepsilon \in \mathbb{R}$ a $\mu \in \mathbb{N}$.

Pokud jsou dva uzly ve stejném ε-sousedství, měli by být zároveň i ve stejném shluku. Tuto myšlenku lze použít při postupném rozšiřování shluku směrem od *core*.

Definice 4.4. Přímá dosažitelnost

$$DirReach_{\varepsilon,\mu}(u, v) \Leftrightarrow CORE_{\varepsilon,\mu}(u) \wedge v \in N_\varepsilon(v)$$

kde U a v je uzel z grafu $G = (V, E)$, $\varepsilon \in \mathbb{R}$ a $\mu \in \mathbb{N}$.

Toto je jen několik základních pojmů, které jsou v článku [19] definovány, ale pro potřeby popisu algoritmu je toto dostačující. Ještě je třeba zmínit alespoň neformálně, že uzel označený jako *hub* je takový uzel, který není součástí žádného shluku a zároveň sousedí s uzly patřícími do více různých shluků. Nakonec uzel označený jako *outlier* je takový, který není ani součástí žádného shluku a není ani *hub*.

Nyní již lze přejít k popisu konkrétních kroků algoritmu 1. Na začátku jsou všechny uzly neklasifikované (nepatří do žádné ze skupin). První for smyčka postupně prochází všechny dosud neklasifikované uzly a kontroluje zda je aktuální uzel *core*. Pokud ano začne od něj expandovat nový shluk. Jeho sousedy vloží do fronty. Postupně vybírá uzly z fronty a pro všechny jeho *přímo dosažitelné* sousedy je buď přiřadí do shluku nebo je přiřadí do shluku a zároveň vloží do fronty. Takto pokračuje dokud fronta není prázdná. Pokud aktuální uzel nebyl *core*, označí ho jako *non-member* (uzel který nepatří do žádného shluku).

Druhá for smyčka již pouze projde všechny *non-member* uzly a označí je jako *hub* nebo jako *outlier*.

4.3.4 Realizace

Během realizace modulu obsahujícího algoritmu SCAN byla vytvořena třída `ScanStatisticsBuilder` builder instance statistiky a `ScanStatistics` obsahující samotný algoritmus. Obě třídy implementují rozhraní tak jak bylo popsáno v sekci 4.2.

Algoritmu je možné nastavit 2 parametry:

Algorithm 1 SCAN

```
1: procedure SCAN(ε, μ)
2:     Označení všech uzlů jako neklasifikované
3:     for each neklasifikovaný uzel u do
4:         if u je core then // expanze nového shluku
5:             vygeneruj nové clusterId
6:             do front Q vlož všechny uzly sousedící s u
7:             while Q není prázdná do
8:                 v ← pop(Q)
9:                 V ← uzly s přímou dosažitelností z v
10:                for each uzel w z V do
11:                    if w je neklasifikovaný nebo je non-member then
12:                        přiřaď w do clusterId
13:                    if w je neklasifikovaný then
14:                        vlož w do Q
15:        else
16:            označ u jako non-member
17:    for each non-member uzel x do //
18:        if alespoň 2 z jeho sousedů maji rozdílné clusterId then
19:            označ x jako hub
20:        else
21:            označ x jako outlier
```

- `minSizeOfCluster` - Integer - představující minimální velikost tvořených clusterů, v sekci 4.3.2 označeno jako μ

- `minSimilarityThreshold` - Double - představuje minimální hranici podobnosti , v sekci 4.3.2 označeno jako ε

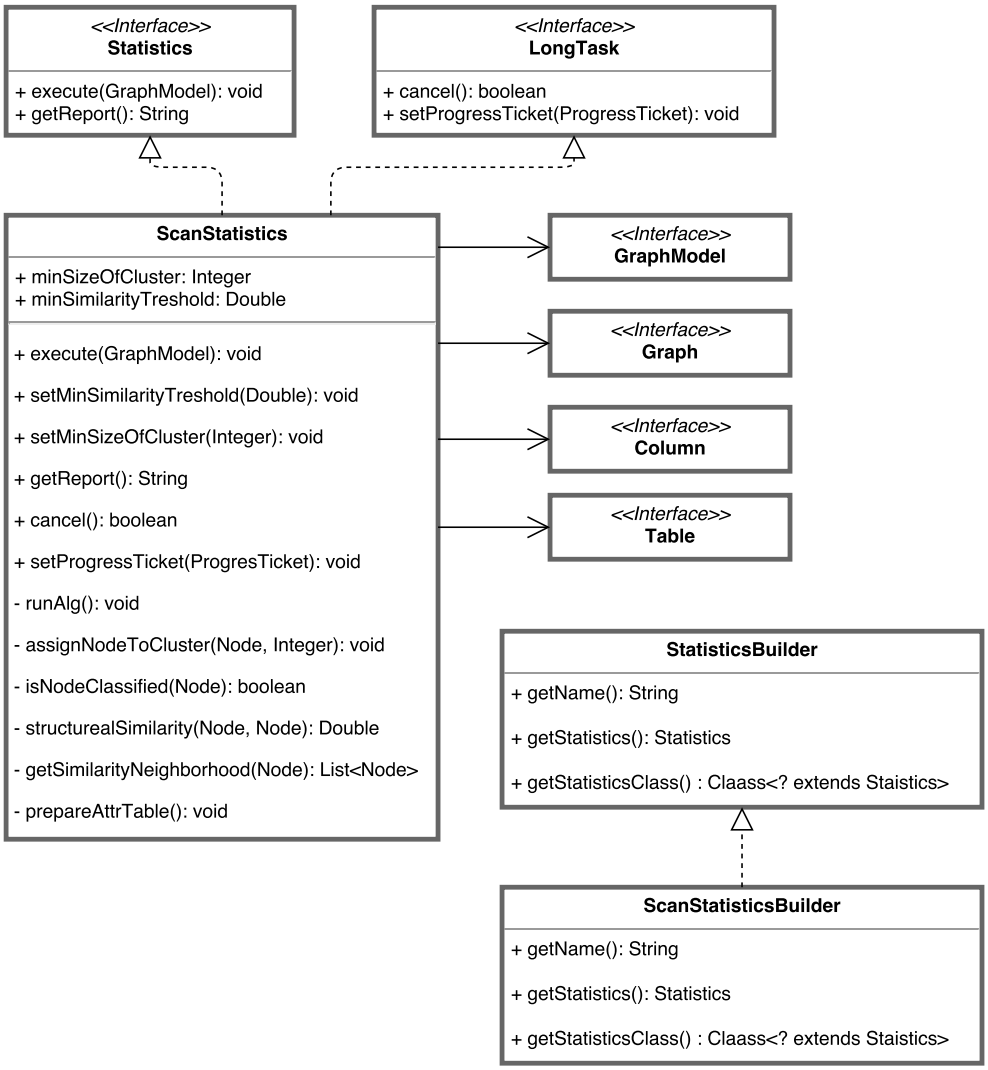
Před spuštěním algoritmu tak jak byl popsán výše je nejprve nutné definovat několik atributů přiřazených uzlům, které se jednak používají v průběhu výpočtu algoritmu, ale hlavně se některé z nich využívají jako výstup. Atributy jsou následující:

- `SCAN_cluster_id` - Integer - Představuje identifikátor shluku do kterého uzel patří, využívá se primárně v průběhu algoritmu, ale může být využit i jako výstupní hodnota pro zpětnou validaci toho, který uzel patří do jakéhu shluku.

- `SCAN_is_clustered` - Boolean - Příznak popisující zda je uzel členem nějakého shluku. Pro průběh algoritmu nemá žádný význam, jedná se pouze o výstupní hodnotu.

- `SCAN_is_hub` - Boolead - Příznak označující uzly typu *hub*, v průběhu algoritmu se nevyužívá je pouze pro výstup.
- `SCAN_is_outlier` - Boolean - Příznak označující uzly typu *outlier*, v průběhu algoritmu se nevyužívá je pouze pro výstup.
- `SCAN_is_non_member` - Boolean - Příznak zda je *non-member*, využívá se v průběhu algoritmu. V posledních krocích algoritmu by všechny *non-member* uzly měli být identifikovány jako *hub* nebo *outlier*.

Samotný průběh algoritmu je implementován v souladu s postupem popsaným v předchozí sekci, není tedy nutné jej zde hlouběji rozebírat.

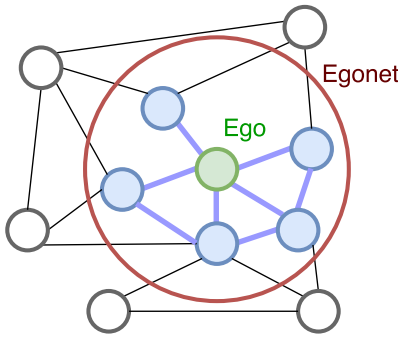


Obrázek 4.4: Class diagram modulu SCAN

4.4 OddBall

4.4.1 Popis metody

Algoritmus OddBall je aplikovatelný na vážený graf bez atributů. Od většiny ostatních metod se liší tím, že analyzovaný graf zkoumá na úrovni jednotlivých *egonetů*. *Egonet* je pojem převzatý z Analýzy sociálních sítí (angl. Social network analysis) a neformálně je definován jako jedno-krokové okolí uzlu, včetně tohoto uzlu samotného a všech hran propojující jeho přímé sousedy. Centrálnímu uzlu kolem kterého je *egonet* konstruován se nazývá *ego*. Příklad *egonetu* je vyobrazen na obrázku 4.5.



Obrázek 4.5: Ego a egonet

Hlavní otázkou kterou se v článku zabývají je, které vlastnosti *egonetu* jsou zajímavé, při zkoumání a detekci anomálií v grafech. Zvolené vlastnosti by měli být snadno extrahovatelné a zároveň napomoci detekovat druh anomálií, který nás zajímá. V článku se zaměřili na čtyři druhy anomálií, které je možné vidět na obrázku 4.6:

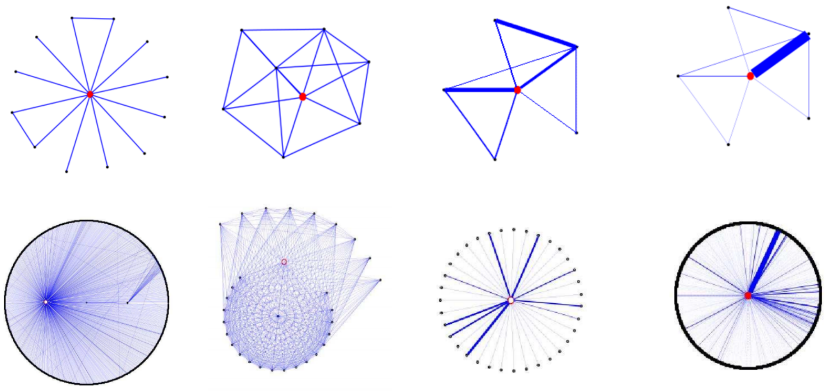
- *Near-star* - Podgraf blízký hvězdě. Jeden uzel který má vazbu na na ostatní, ale ti již nemají vazby mezi sebou lze považovat za anomální. V běžných grafech můžeme pozorovat jev který se nazývá *Triadic closure* [35], což je vlastnost mezi třemi uzly, která říká že pokud máme uzly u, v, w a existuje propojení mezi $u - v$ a propojení mezi $u - w$ má smysl očekávat i propojení mezi $w - v$.

- *Near-clique* - Silně provázané skupiny uzlů jsou v grafech poměrně běžnou záležitostí, na které staví mnoho algoritmů pro detekci anomálií a považují ji za zcela normální. OddBall však detekuje skupiny uzlů, které jsou provázány tak silně (tvoří téměř kliku), že je to možné považovat za abnormální chování.

- *Heavy vicitnity* - Vysoký součet vah v rámci podgrafu indikuje vysokou aktivitu v rámci jednoho *egonetu*, kterou OddBall může identifikovat jako anomální.

- *Dominant edge* - Jedna hrana, které svou vahou výrazně převyšuje ostatní v rámci jednoho *egonetu*.

Tyto vzory byly vypozorovány v rámci zkoumání mnoha různých grafů ze skutečného světa a na základě těchto pozorování bylo definováno několik zákonů, kterými se reálné grafy běžně řídí a které budou popsány v následujících odstavcích. Je zde však nutné podotknout, že všechny zkoumané grafy měli formu blízkou sociální síti, tedy se v ním pohybovali a vzájemně interagovali skuteční lidé. V některých obecných grafech proto tyto zákony nemusí platit a tedy pro ně OddBall nelze moc efektivně použít.



(a) Near-star (b) Near-clique (c) Heavy vicinity (d) Dominant edge

Obrázek 4.6: OddBall : typy detekovaných anomálií [3]

Dále v článku identifikovali čtveřici vlastností *egonetů*, na základě kterých je možné detekovat představené typy anomálií:

- N_i počet vrcholů v *egonetu* odpovídající *egu* i
- E_i počet hran v *egonetu* odpovídající *egu* i
- W_i součet vah hran v *egonetu* odpovídající *egu* i
- λ_i hlavní vlastní číslo vážené matice sousednosti v *egonetu* odpovídající *egu* i

Tyto vlastnosti jsou následně studovány v párech, pro které je definováno několik pozorování. Každý studovaný pár je určen k detekci konkrétního typu anomálie. Analyzování těchto vlastností v párech přináší několik výhod, než kdyby byly zkoumány jako sjednocení ve 4-dimenzionálním prostoru. Zaprvé to usnadňuje vizualizaci výsledků, která je důležitá při analýze výsledků, které mohou být snadno vyneseny v 2-dimenzionálním grafu. Druhou výhodou je, že nízká dimensionalita prostoru umožňuje snadnou interpretaci výsledků a na základě toho jaký ze zákonů uzel při analýze porušuje, lze určit o jaký typ anomálie se jedná.

Nyní se konečně dostáváme k již několikrát zmíněným zákonům, které dle pozorování popsaných v článku grafy z reálného světa následují. Tato pozorování jsou popsána jako několik mocninných zákonů mezi zmíněnými dvojicemi vlastností *egonetů*.

Pozorování 4.1. Hustota Egonetu.

$$E_i \propto N_i^{\alpha}, 1 \geq \alpha \geq 2$$

kde N_i je počet uzlů *egonetu* i a E_i je počet jeho hran.

Egonety vychylující se z tohoto pozorovaná jsou s největší pravděpodobností *near-clique* nebo *near-star*, v závislosti na to kterým směrem se vychylují.

Pozorování 4.2. Váha Egonetu.

$$W_i \propto E_i^{\beta}, \beta \leq 1$$

kde W_i je součet vah hran v *egonetu* i a E_i je počet jeho hran.

Na základě této metriky je možné detekovat *egonety* s vysokou vahou (*HeavyVicinity*).

Pozorování 4.3. Dominance hrany Egonetu.

$$\lambda_i \propto W_i^{\gamma}, 0.5 \geq \gamma \geq 1$$

kde λ_i je hlavní vlastní číslo vážené matice sousednosti *egonetu* i a W_i je součet vah jeho hran.

Pomocí tohoto pozorování lze v grafu detekovat dominantní páry (*DominantPair*).

Přiřazení anomálního skóre jednotlivým uzlů je pak již poměrně jednoduché. *Ego* uzel jehož *egonet* se vychyluje od definovaných pozorování je považován za anomální a čím více se vychyluje od očekávané hodnoty, tím anomálnější je.

Definice 4.5. Anomálnost uzlu. Označme y hodnotu uzlu i jako y_i a x hodnotu uzlu i jako x_i pro pár vlastností *egonetu* $f(x, y)$. Pro daný mocninný zákon $y = Cx^{\theta}$ pro funkci $f(x, y)$ definujeme anomální skóre uzlu i jako

$$outLine(i) = \frac{max(y_i, Cx_i^{\theta})}{min(y_i, Cx_i^{\theta})} * log(|y_i - Cx_i^{\theta}| + 1)$$

Hodnota anomálního skóre je tedy 0, v případě že *egoent* přesně odpovídá očekávané hodnotě. Čím více se poté hodnota liší tím vyšší je i anomální skóre.

4.4.2 Vstupní/výstupní parametry

Algoritmus je aplikovatelný na vážený neorientovaný graf bez atributů.

Jako vstupní parametry je nutné zvolit hodnoty exponentů pro jednotlivé zákony v předchozí sekci označeny jako α, β a γ.

Výstupem je ohodnocení všech uzlů v grafu anomálním skóre z intervalu $[0, \infty]$.

4.4.3 Postup algoritmu

Postup algoritmu přímo vychází z jeho popisu v předchozích sekcích. Dal by se rozdělit do následujících kroků:

1. Extrakce *egonetů* v grafu.
2. Evaluace představených vlastností *egonetů* k následující analýze. Konkrétně jsou to: stupeň *ego* uzlu, počet hran v *egonetu*, součet vah hran v *egonetu* a hlavní vlastní číslo vážené matice sousednosti *egonetu*.
3. Výpočet anomálních skóre jednotlivých *ego* uzlů na základě definovaných pozorování a formule pro výpočet skóre.

4.4.4 Realizace

Při implementaci algoritmu bylo vytvořeno několik tříd, zde je jejich výčet a popis jejich funkcionality.

OddBallStatisticsBuilder

Builder třída, která se stará o vytvoření instance statistiky, tak jak bylo popsáno v sekci 4.2.

AnomalyType

`AnomalyType` je Java enum, který definuje trojici anomálií, který je OddBall schopen detekovat.
Konkrétně: `START_CLIQUE`, `HEAVY_VICINITY` a `DOMINANT_EDGE` viz předchozí sekce.

OddBallStatistics

Třída obsahující samotný algoritmus. Přijímá uživatelské parametry inicializuje běh algoritmu, extrahuje *egonety* a následně zapisuje výsledky do tabulky atributů uzlů.

OddBall je schopen detekovat více druhů anomálií a k detekci každé z nich je nutné z *egonetu* extrahovat rozdílné metriky. Někdy však uživatel nemusí vyžadovat detekci všech typů anomálií a některé metriky by mohli být počítány zbytečně což může být problém hlavně při výpočtu vlastního čísla matice sousednosti, která je časově nejnáročnější. Z toho důvodu má uživatel možnost přesně definovat, které typy anomálií ho zajímají.

Egonet

`AnomalyType` je třída představující jeden konkrétní **egonet**. Stará se především o extrakci potřebných metrik a výpočet svého anomálního skóre.

4.4.4.1 Vstupní parametry

Algoritmus má 2 uživatelsky definované parametry:

- Seznam detekovaných anomálií společně s hodnotou exponentu do výpočtu anomálního skóre. Parametr se předává jako seznam objektů třídy `InputParam`, která nemá žádnou funkcionalitu, pouze obsahuje zmíněné dvě hodnoty.
- Hloubka *egonetu*. OddBall implicitně operuje s *egonety* hloubky jedna (jednokrokové okolí *ego* uzlu), je však snadné jej zobecnit i pro širší okolí.

4.4.4.2 Kroky implementovaného algoritmu

Implementovaný algoritmus je možné rozdělit do několika kroků:

1. **Extrakce egonetů** - Algoritmus prochází všechny uzly grafu a pro každý z nich extrahuje jeho *egonet* pomocí průchodu BFS s omezenou uživatelem nastavenou hloubkou.

   K tomuto je s výhodou využit tzv `GraphView` poskytovaný frameworkem Gephi Toolkit. *Graph view* definuje podgraf originálního grafu. Každý `GraphModel` má alespoň jedno view nazvané *main view*, které obsahuje všechny jeho elementy. Ostatní *view* mohou definovat podgrafy obsahující redukovaný počet uzlů a hran bez nutnosti úpravy grafu nebo kopírování části struktury.

   Každý extrahovaný *egonet* je reprezentován svou instancí třídy `Egonet`.

2. *Extrakce metrik egonetů a výpočet anomálního skóre* - Pro každý z *egonetů* je nutné extrahovat potřebné metriky, které se odvíjejí z typů anomálií, které mají být detekovány.

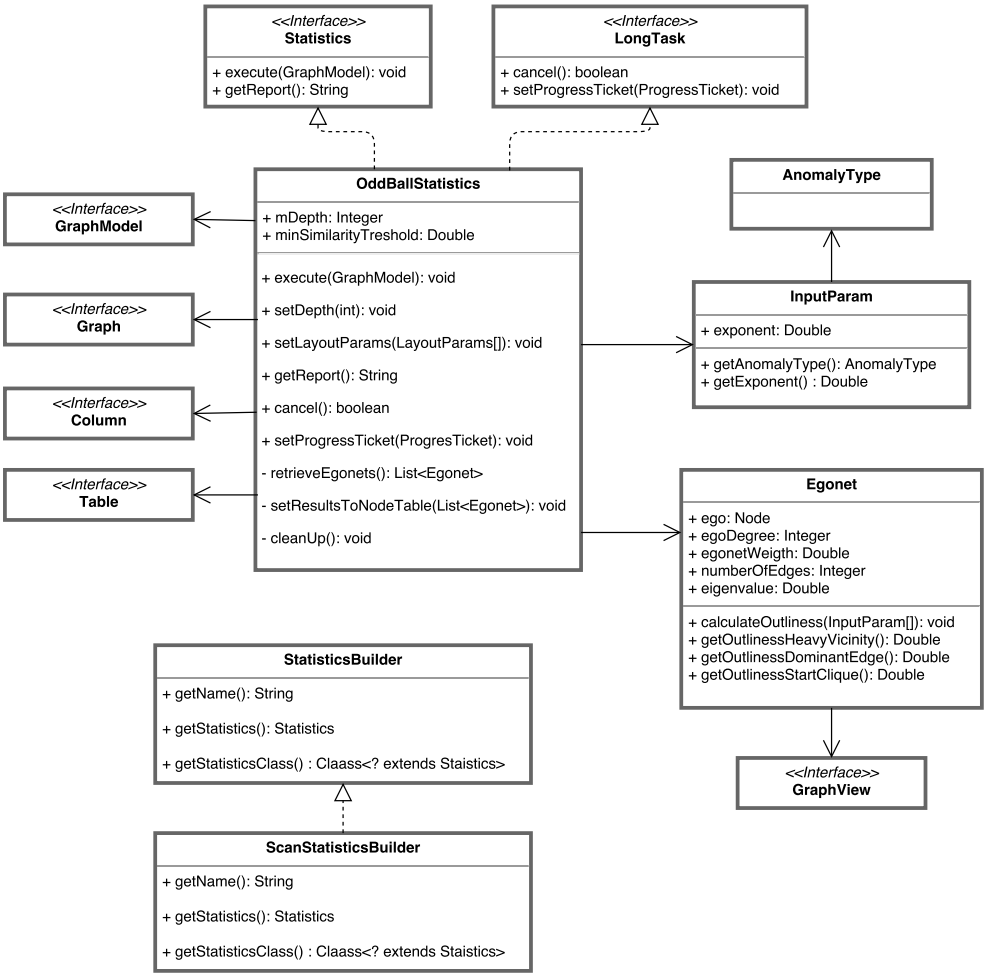
   Díky tomu že každý z *egonetů* je reprezentován jako `GraphView`, výpočet metrik jako jsou počet uzlů, suma vah hran a počet hran jsou jednoduché a samo Gephi pro ně má vlastní metody. S výpočtem hlavního čísla matice sousednosti je to složitější a pro jeho výpočet se v implementaci využívá knihovny Jama [36] pro práci s maticemi v jazyku Java.

3. **Zápis výsledků** - Výsledky jsou reprezentovány jako atributy, přiřazené uzlům. Jejich kompletní výčet je v následující sekci.

4.4.4.3 Výstupní parametry

Výstup je zapsán formou atributů přiřazených všem uzlům. Kromě vypočtených anomálních skóre jsou k uzlům přiřazeny i metriky extrahované z jejich *egonetů* v průběhu výpočtu, pro možnost zpětné validace výsledků. Atributy jsou následující:

- `OddBall_egoDegree` - Integer - Stupeň *ego* uzlu.
- `OddBall_numOfEdges` - Integer - Počet hran v *egonetu*.
- `OddBall_egonetWeight` - Double - Součet vah hran *egonetu*.
- `OddBall_eigenvalue` - Double - Hlavní vlastní hodnota matice sousednosti *egonetu*.
- `OddBall_heavy_vicinity_outliness` - Anomální skóre pro typ anomálie *HeavyVicinity*.
- `OddBall_dominant_edge_outliness` - Anomální skóre pro typ anomálie *DominantEdge*.
- `OddBall_star_clique_outliness` - Anomální skóre pro typ anomálie *StarClique*.



Obrázek 4.7: Class diagram modulu OddBall

Kapitola 5

Vlastní metody

Jelikož je problém detekce anomálií v grafech relativně nová a málo prozkoumaná oblast, je v ní poměrně velký prostor pro nové přístupy. Tato kapitola je věnována experimentování a navržení nových metody pro detekování anomálií. Konkrétně jsou v následujícím textu popsány dvě metody první je založené na změnách entropie v grafu a druhá je modifikací metody založené na shlukování v grafu SCAN.

Kapitola je rozdělena následujícím způsobem. Nejprve je popsáno co bylo motivací při navržení metody a jaké jsou na metodu kladeny nároky. Následně je představena metoda samotná a je popsána její realizace, která je i součástí výsledné knihovny pro detekci anomálií v grafech popsané v předchozích kapitolách. Vyhodnocení kvality metody je zpracováno ve zvláštní kapitole, ve které jsou zároveň vyhodnoceny a porovnány všechny implementované metody v knihovně.

5.1 Metoda založena na změnách entropie

Jedním z měřítek které je nutné brát v potaz při detekci anomálií v datech je jejich regularita. Obecně je očekáváno, že většina vygenerovaných dat bude mít určité regulární a předvídatelné rozložení ze kterého se anomálie budou vymykat. Čím očekávatelnější rozložení běžných dat bude, tím snazší bude detekce anomálií.

Entropie je dobře známý koncept, který zjednodušeně řečeno měří počet bitů nutných k popsání nějaké nastalé události. Ve spojení s regularitou, má entropie opačný význam něž regularita, tedy čím vyšší entropie je tím nižší je regularita. Díky této vlastnosti je entropii možné využít k odhadnutí předvídatelnosti dat v grafu a zkoumat které z entit v grafu mají na celkovou entropii největší dopad.

5.1.1 Související práce

Existuje několik podobných prací, které využívají entropii dat v nějaké podobě pro detekci anomálií v grafech.

První z nich je metoda Subdue, které vznikla již v roce 2003 a které byla věnována sekce 2.3.1. Jak bylo zmíněno Subdue je celý systém pro vytěžování strukturálních a vztahových znalostí z dat reprezentovaných jako graf. Detekce anomálií je jen jednou z možných aplikací tohoto systému. V článku [13] je velmi zjednodušeně načrtnut postup jakým Subdue detekuje regularitu podstruktur grafu pomocí podmíněné entropie. Podmíněná entropie je definována následovně.

Definice 5.1. Podmíněná entropie.[13]

$$H(X|Y) = -\sum_{y \in Y} \sum_{x \in X} P(y) * P(x|y) * log(P(x|y))$$

kde X je množina všech možných jevů, Y je množina známých jevů, které již nastaly, $P(y)$ je pravděpodobnost že jev y nastal a $P(x|y)$ je pravděpodobnost, že jev x nastane za předpokladu že jev y již nastal. Předpokládá se že $log(0) = 0$.

Za pomocí takto definované entropie (pouze s menší úpravou tak aby odpovídala jejich potřebám) aplikované na podstrukturu grafu se snaží odhadnou, kolik bytů je nutné k popsání okolí dané podstruktury. Okolím podstruktury je myšlena množina rozšíření podstruktury o jeden uzel společně se všemi hranami spojujícími uzel s podstrukturou nebo přidání jedné hany v rámci podstruktury.

Množina Y tedy obsahuje všechny podstruktury grafu obsahující n uzlů a X obsahuje všechny rozšíření podstruktur v Y. Pro $y \in Y$ je $P(y)$ definováno jako počet instancí y v celém grafu děleno počtem všech n uzlových podstruktur. Pro konkrétní podstruktury $x \in X$ a $y \in Y$ je $P(x|y)$ definováno jako procento instancí y, které rozšiřují x.

V článku bohužel neuvádějí výpočetní složitost této metody, ale vzhledem k nutnosti extrakce všech podstruktur, nalezení všech jejich rozšíření a následného porovnávání je výpočetní komplexita pravděpodobně vysoká.

Druhou metodou která využívá entropii pro detekci anomálií v grafech je popsána v článku [37]. Tato metoda je zaměřena na analýzu informačních toků v organizaci a detekci klíčových entit. Metoda na graf klade několik požadavků. Graf musí být orientovaný multigraf (obsahující rovnoběžné hrany), každý uzel reprezentuje entitu v organizaci a hrany definují určitý druh výměny informace společně s časovou značkou kdy k výměně došlo. Například se v grafu mohou vyskytovat hrany typu telefonní hovor vyjadřující, že entita A telefonovala s entitou B, analogicky pro emailovou komunikace, schůzky atd.

Metoda je založena na analýze jednotlivých časových oken a detekci řetězů komunikace, které v organizaci nastaly. Například pokud entita A odeslala email entitě B a v krátkém časovém odstupu entita B odeslal email entitě C, metoda předpokládá, že došlo k přeposlání jednoho emailu, který doputoval od entity A k entitě C. Takto mohou vnikat libovolné dlouhé řetězce výměny informací. Algoritmus následně odebíráním jednotlivých řetězců a přepočítáváním změn entropie odhaduje jejich významnost v grafu a umožňuje detekci významných uzlů v grafu.

5.1.2 Popis navržené metody

Navržená metoda je aplikovatelná na obyčejný vážený graf a využívá entropii pro detekci anomálních uzlů v grafu. Podobně jako algoritmus OddBall 2.3.5, metoda zkoumá blízké okolí každého z uzlu. Tato myšlenka je založena na předpokladu, že pokud jo uzel anomální, projeví se tato vlastnost na způsobu jakým je uzel propojen s okolními uzly.

Entropie je maximální pokud mají všechny jevy stejnou pravděpodobnost a v takovém případě je regularita zdroje jako celku minimální. Tím že se některý z jevů začne vyskytovat častěji, jeho pravděpodobnost se zvýší a pravděpodobnost ostatních se sníží. Tím dojde k rozvážení uniformního rozdělení jevů a celková entropie se začne snižovat.

Metoda je založena na intuitivní myšlence, porovnaní entropie grafu jako celku a entropie grafu po odebrání nějakého uzlu společně s jeho blízkým okolím. Uzly jejichž okolí je něčím významné od zbytku grafu by se tedy svým odebráním měli ve změně entropie projevit výrazněji.

Bohužel neexistuje žádná obecně uznávaná definice entropie grafu. Jednou pravděpodobně z nejznámějších entropií grafu je *Körner Entropy*[38], která je definována jako

$$H(G,P) = \min_{x \in StableSet(G)} \sum_{v \in V(G)} p(v) log(p(v))$$

kde *StableSet(G)* představuje výsledné skupiny z tzv. *stable set problem*[39] neboli problému obarvování grafu. Problém spočívá v rozdělení uzlů do skupin tak, že žádné dva uzly ze stejné skupiny spolu nesousedí. To je však známý NP-těžký problém a z toho důvodu má tato entropie spíše teoretický účel, ale v praxi je nespočítatelná v rozumném čase.

Co se týče entropie vážených grafů jednou z nejznámějších definicí je entropie popsána v článku[40], která je definována následovně.

Definice 5.2. Entropie váženého grafu.

$$H(G) = -\sum_{e \in E} p(e) log(p(e))$$

kde G je graf $G = (V, E)$ a

$$p(e) = \frac{w(e)}{\sum_{e' \in E} w(e')}$$

, kde $w(e) : e \to \mathbb{R}$ je váha hrany e

Tato entropie je sice poměrně jednoduchá na výpočet, ale na druhou stranu nebere v potaz strukturu grafu a zaměřuje se primárně na váhy jednotlivých hran.

Další metodou pro výpočet entropie ve váženém grafu je metoda v článku [38] nazvána jako *Non-parametric Graph entropy*, která pro výpočet využívá vlastní čísla grafu.

Definice 5.3. Non-parametric Graph Entropy.

$$H(G) = -\sum_{i=1}^{k} \frac{|\lambda_i|}{\sum_{j=1}^{|V|} |\lambda_j|} log(\frac{|\lambda_i|}{\sum_{j=1}^{|V|} |\lambda_j|})$$

kde G je graf $G = (V, E)$ a $\lambda_1, \lambda_2, ..., \lambda_k$ jsou nenulové vlastní hodnoty charakteristického polynomu G.

Tato entropie využívá i skrytých topologických uspořádání v grafu a je tedy citlivá i na změnu ve struktuře grafu. Navíc díky symetrii matice sousednosti neorientovaného grafu mají výsledná vlastní čísla jen reálnou složku. Její výpočetní složitost je však řádově vyšší než v případě vážené entropie. Složitost výpočtu vlastních čísel matice $n \times n$ je v praxi odhadována na $O(n^3)$, pokud je toto nutné provést n-krát po odebrání okolí každého z uzlu dostáváme se na $O(n^4)$.

5.1.3 Vstupní/výstupní parametry

Metoda je použitelná na obyčejný graf s váženými hranami. Vyžaduje dva vstupní parametry, velikost okolí odebíraného uzlu a druh entropie. Výstupem je anomální skóre ohodnocující každý z uzlů grafu.

5.1.4 Postup algoritmu

Algoritmus lze rozdělit do několika následujících kroků vycházejících z popisu výše.

1. Výpočet referenční entropie celého grafu.

2. Pro každý uzel v grafu:
    a) Odebrání okolí uzlu společně s uzlem samotným z původního grafu.
    b) Výpočet entropie nového grafu.
    c) Přiřazení uzlu anomálního skóre rovnající se rozdílu referenční entropie a entropie po odebrání.

5.1.5 Realizace

Při implementaci byl dodržen stejný postup jako při implementaci dvou zvolených metod v kapitole 4. Byl vytvořen nový statistický Gephi modul implementující rozhraní *Statistics* a *LongTask*. Konkrétně byli vytvořeny čtyři nové třídy

AnomalousEntropyStatisticsBuilder

Builder třída, které se stará o vytvoření instance statistiky, tak jak bylo popsáno v sekci 4.2.

AnomalousEntrolyStatistics

Třída obsahující logiku algoritmu. Přijímá uživatelské parametry, inicializuje běh algoritmu a zapisuje výsledky. Postup algoritmu vychází přesně z popisu v sekci 5.1.4.

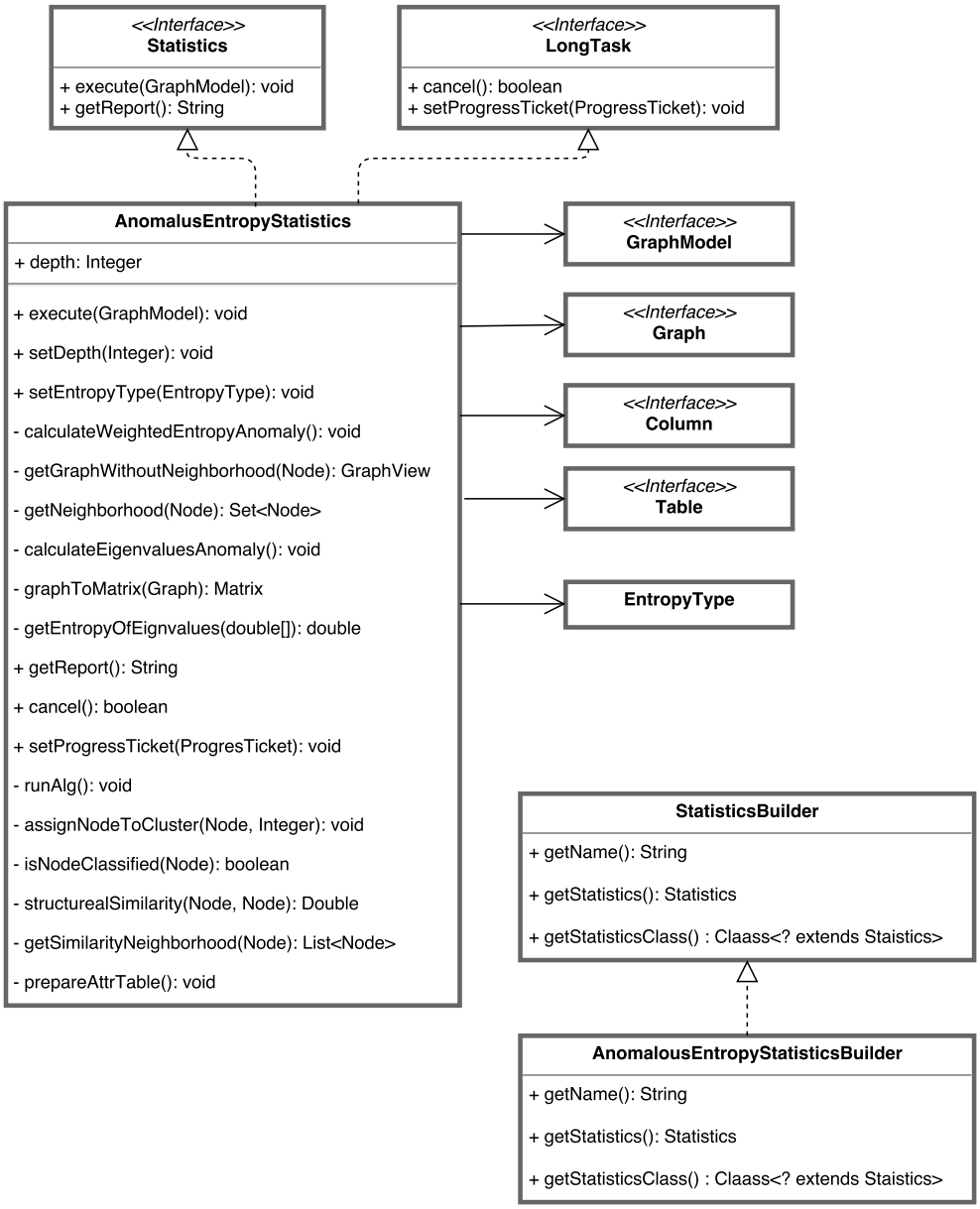
Nejprve je vytvořen nový sloupec pro přiřazení atributů uzlům pro zápis výsledného anomálního skóre uzlu. Je vypočtena referenční hodnota entropie celého grafu. Následně jsou procházeny jednotlivé uzly, pro každý z nich je vytvořeno `GraphView`, které umožňuje definovat nový podhled na graf a vytvořit tak jeho podgraf, bez nutnosti editace původního grafu. Ve vytvořeném `GraphView`, jsou odebrány sousedi aktuálního uzlu a je přepočítána entropie modifikovaného grafu. Výsledná hodnota anomálního skóre je zapsána do tabulky atributů grafu.

EntropyType

`EntropyType` je Java *enum* obsahující dva druhy entropií, které je možné v algoritmu aplikovat a používá se jako vstupní parametr při inicializaci algoritmu. Konkrétně lze zvolit mezi hodnotami `WEIGHTED_ENTROPY` odpovídající *Entropii váženého grafu* a `NON_PARAM_ENTROPY` odpovídající *Non-parametric Graph Entropy*.

AnomalousEntrolyConfig

Tato třída je použita jako vstupní konfigurace pro spuštění algoritmu v knihovně GAD. Obsahuje všechny potřebné informace a parametry pro spuštění algoritmu.



Obrázek 5.1: Class diagram modulu AnomalousEntropy

Jak již bylo zmíněno výsledky jsou zapisovány do tabulky atributů uzlů, konkrétně do sloupce `ANOMALOUS_ENTROPY_outlierness`.

5.2 Modifikace metody SCAN

Druhá navržená metoda je založena na metodě SCAN, ale teoreticky bylo by možné ji zkombinovat i s dalšími metodami pro detekci anomálií v grafech založených na shlukování, fungujícími na podobném principu.

Jelikož SCAN je popsán podrobněji v sekci 4.3, pouze zde shrnu jeho hlavní vlastnosti. Algoritmus SCAN detekuje anomálie v grafech za pomoci detekce shluků uzlů a dal by se rozdělit do dvou fází. V první fázi postupně prochází jednotlivé uzly grafu a posuzuje u nich zda by mohli být tzv. *core* (jádrem) shluku. Pokud se podaří některé z *core* nalézt přejde do fáze dvě, ve které se snaží postupně připojovat dostatečně podobné uzly v okolí *core* a postupně tak zvětšovat jeden ze shluků dokud to lze. Pokud v grafu existuje ještě nějaký doposud nenavštívený uzel, přejde algoritmus zpět do první fáze.

Výsledkem jsou uzly rozřazeny do dvou skupin: uzly které jsou součástí nějakého shluku a uzly které nelze do žádného ze shluků zařadit. Nezařazené uzly jsou dále rozděleny na tzv. *hub* uzly, které tvoří propojení mezi dvěma a více shluky a *outlier* uzly, které nejsou ani součástí shluku a nejsou ani *hub*.

Uzlům které se podařilo zařadit do nějakého ze shluků se algoritmus však již nevěnuje. Ačkoliv se uzel dostal do nějakého ze shluků neznamená to že v rámci jeho shluku nemůže být něčím výjimečný, znamená to pouze, že je propojen se svým okolím natolik, že je považován za součást shluku.

Modifikace metody SCAN popsané v této sekci se zabývá doplněním tohoto možného nedostatku a zkoumá vlastnosti uzlů v rámci jednotlivých shluků a snaží se detekovat další druh anomálií v grafech.

5.2.1 Popis metody

Vhledem k tomu, že algoritmu SCAN je aplikovatelný pouze na obyčejné neorientované grafy, měla by modifikace toto omezení zachovat.

Hlavní otázkou, kterou navržená metoda řeší je: Jaké metriky okolí uzlu zvolit pro detekci anomálií v rámci shluku grafu?

Jelikož se jedná pouze o obyčejný graf bez atributů a vah je nutné se omezit pouze na strukturální vlastnosti grafu. Tyto vlastnosti by měli být z grafu snadno extrahovatelné a zároveň poskytnou informace pro detekci anomálních uzlů. Při rozhodování zda si jsou dva uzly podobné SCAN využívá takzvanou strukturální podobnost, pro připomenutí definovanou následovně.

Definice 5.4. Strukturální podobnost.

$$\sigma(u,v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{\sqrt{|\Gamma u||\Gamma v|}}$$

Kde u a v jsou uzly grafu $G = (V, E)$ a $\Gamma(u)$ je definována jako množina sousedních uzlů uzlu u, formálně

$$\Gamma(u) = \{w|v, w \in E\} \cup u$$

Je tedy zřejmé že pokud jsou dva sousední uzly v jednom shluku, sdílejí spolu velkou část svých sousedních uzlů. Přesto ve shlucích mohou existovat uzly s různou provázaností se svým okolím, například pokud se v grafu bude vyskytovat klika budou mít její uzly velmi vysokou strukturální podobnost, pravděpodobně tak vysokou že většinu z nich algoritmus označí za *core* uzly. Kliky lze však v mnoha aplikačních doménách považovat za anomálie, které by měli být identifikovány.

Další možnou anomálií, kterou je dobré brát v potaz je stupeň uzlu. Stupeň uzlu který je součástí shluku jistě nebude příliš nízký, pokud by byl, je velká pravděpodobnost, že by ho SCAN sám o sobě označil za *outlier* uzel. Na druhou stranu je možné že stupeň uzlu bude natolik vysoký, že bude převyšovat všechny ostatní ve shluku, to je jistě možné považovat za podezřelé.

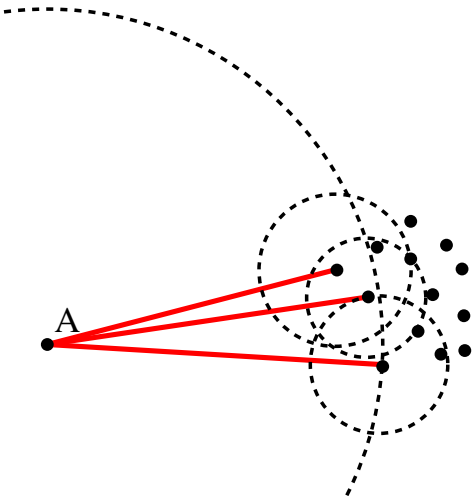
Jistě lze definovat řadu dalších druhů anomálií, které se v grafu mohou vyskytovat, v následujícím textu se však primárně zaměřím na tyto dvě.

Nyní je nutné zvolit metriky okolí uzlů, na jejichž hodnotu budou mít vybrané anomálie velký dopad. Silně provázané okolí uzlů, které tvoří podgrafy blízké klice je možné detekovat například pomocí aproximace počtu trojúhelníků, které se v okolí uzlu nacházejí. Trojúhelníkem je zde myšlena klika stupně 3. Je zřejmé že čím více se bude podoba okolí uzlu blížit klice tím více trojúhelníků bude obsahovat a naopak pokud podgraf bude tvořit tzv. hvězdu nebude trojúhelníky obsahovat žádné. Druhou metriku pro detekci různého počtu sousedních uzlů lze zvolit poměrně přímočaře jako stupeň uzlu.

Dle definice anomálie jakožto jevu, který je odlišný od zbytku pozorovaných dat, lze intuitivně očekávat, že hodnoty zvolených metrik budou pro jednotlivé uzly v rámci shluku podobné a uzly pro které toto neplatí by měli být označeny za anomálii. Toho je možné docílit například vynesením naměřených hodnot do dvou rozměrného prostoru a aplikovat na ně některou z metod pro detekci outlierů v prostoru. Takovou metodou může být například **LOF**[41], ale bylo by možné využít i libovolnou jinou s podobnými vlastnostmi.

LOF - Local outlier factor

Metoda je založena na lokální hustotě bodů v prostoru. Lokální hustota je odhadnuta na základě vzdálenosti ke k nejbližším sousedům. Při porovnaná



Obrázek 5.2: Základní myšlenka algoritmu LOF[4]

lokálních hustot jednotlivých bodů s jejich sousedy se uzly rozdělí do dvou kategorií, uzly které mají podobnou hustotu jako jejich sousedi a uzly které mají podstatně nižší hustotu. Druhá skupina jsou právě hledané odlehlé hodnoty.

Na obrázku 5.2 je znázorněno porovnání hustot sousedních bodů s bodem A při $k = 3$. Je zřejmé že hustota bodu A je podstatně nižší (vzdálenost k jeho k nejbližším sousedům je vyšší) v porovnání s jeho sousedy a proto by bod A byl pravděpodobně považován za *outlier*.

5.2.2 Vstupní/výstupní parametry

Algoritmus je omezen na neorientovaný graf bez vah. Vyžaduje stejné vstupní parametry jako algoritmus SCAN tedy:

- ε - určuje minimální hranici podobnosti takovou aby dva uzly mohli být považovány za podobné.
- μ - určuje minimální velikost shluku.

Dále je nutné zadat parametr k - počet nejbližších sousedů pro algoritmus LOF.

Výstupem je ohodnocení všech uzlů. Ohodnocení může nabývat 3 hodnot: *hub*, *outlier* nebo uzel obsažen ve shluku. Každý uzel který je obsažen v grafu je navíc ohodnocen anomálním skóre $[0, \infty)$.

5.2.3 Postup algoritmu

Postup algoritmu je následující:

1. Rozdělení uzlů do 3 kategorií pomocí algoritmu SCAN 4.3: *hub*, *outlier*, uzel ve shluku
2. Rozdělení uzlů do skupin podle shluku do kterých patří
3. Pro každou skupinu:
   a) Extrakce počtu trojúhelníků a stupňů jednotlivých uzlů ve skupině a sestavení dvou dimenzionálního prostoru
   b) Pomocí LOF přiřazení anomálního skóre všem uzlům

5.2.4 Realizace

Při implementaci byl vytvořen nový modul s názvem `ScanExtStatistics` (ze SCAN extension), který z části využívá již hotové implementace algoritmu SCAN, která je obsažena v knihovně GAD.

ScanExtStatisticsBuilder

Builder třída starající se o vytvoření nové instance statistiky. Díky anotaci `ServiceProvider` je dohledatelná pomocí `Lookup API`.

ScanExtStatistics

Třída obsahující samotný algoritmus. Umožňuje nastavení uživatelských parametrů `minSizeOfCluster` a `minSimilarityTreshold` nutných pro inicializace běhu algoritmu SCAN a dále parametr `kNeighbors` využit jako parametr pro LOF.

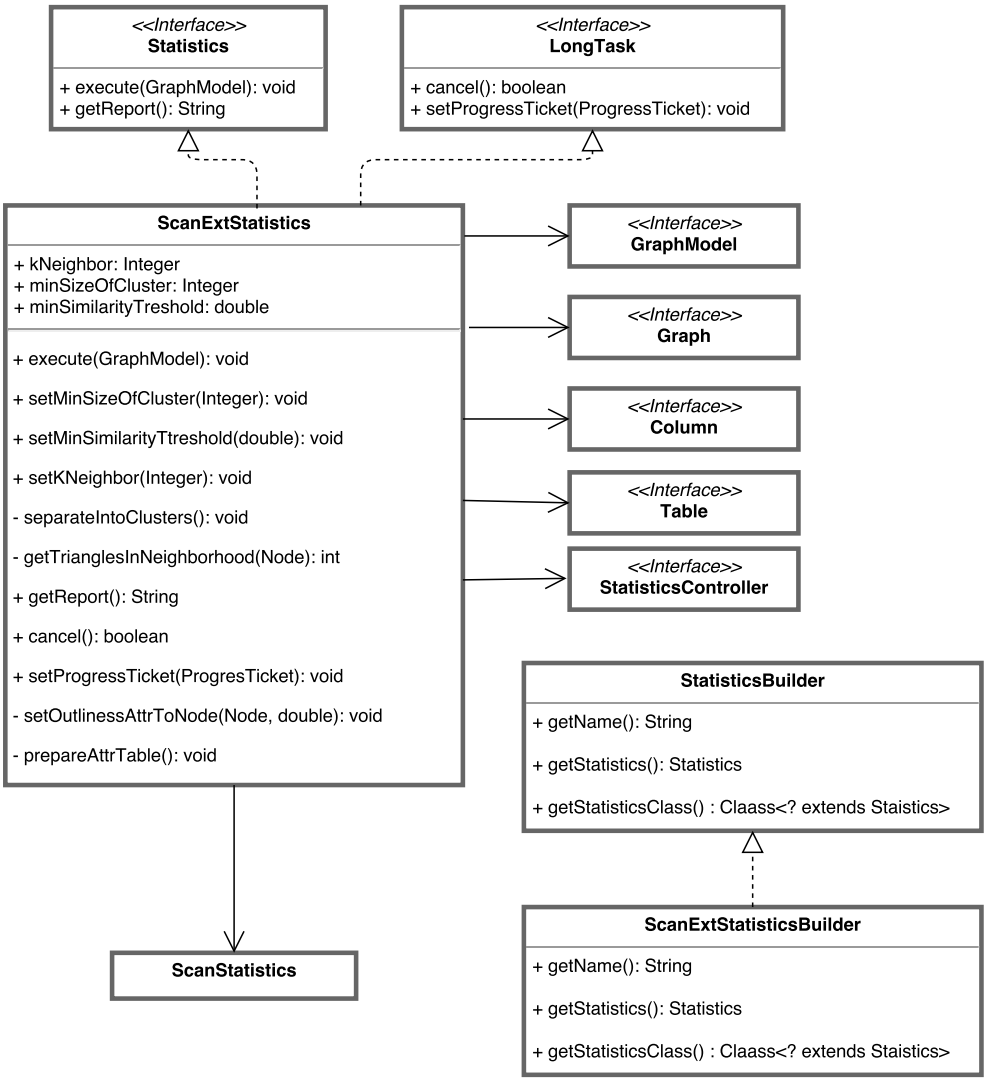
Po spuštění je nejprve pomocí `StatisticsController` kontrolerem pro správu modulů statistik vyhledán a spuštěn algoritmus ScanStatistics nad předaným grafem. Jedná se o algoritmus implementovaný v předchozí kapitole 4.3. Jedním z výstupů tohoto algoritmu jsou atributy s identifikačními čísly shluků přiřazených jednotlivým uzlům podle toho jak byly rozděleny. Tento údaj je použit pro rozdělení uzlů do skupin.

Z každého uzlu ve skupině jsou extrahovány vybrané metriky (počet trojúhelníků v jeho okolí a stupeň uzlu). Z extrahovaných metrik je vytvořen dvoudimenzionální prostor a pomocí LOF jsou jednotlivé uzly ohodnoceny svými *outlier* skóre.

Jelikož je LOF poměrně známý algoritmus existuje pro něj řada hotových implementací. V realizaci modulu využívám jednu z nich [42].

ScanExtConfig

Stejně jako všechny ostatní moduly pro detekci anomálií v knihovně GAD i tento modul má svou konfigurační třídu použitou pro předání parametrů algoritmu, při jeho inicializaci. Tato konfigurační třída je velmi podobná té pro



Obrázek 5.3: Class diagram SCAN Ext

algoritmus SCAN, protože vyžaduje všechny potřebné parametry pro spuštění `ScanStatistics`. Navíc však obsahuje parametr `kNeighbor`, nutný pro LOF.

Výsledky algoritmu jsou jako v případě ostatních modulů uloženy v tabulce atributů přiřazených uzlů konkrétně pod sloupcem označeným jako `SCAN_EXT_outlierness` s hodnotou anomálnímu skóre. Ostatní výstupní hodnoty jako je kategorizace uzlů *hub* a *outlier* jsou již přiřazeny z předchozího běhu algoritmu `ScanStatistics` a není je tedy nutné nastavovat znovu.

Kapitola 6

Vyhodnocení

Tato kapitola se věnuje vyhodnocení všech implementovaných metod pro detekci anomálií v grafech, které jsou součástí výsledné knihovny GAD. Konkrétně jsou to dvě obecně známé metody OddBall a SCAN společně se dvěmi vlastními metodami první založené na změnách entropie v grafu a druhé metodě rozšiřující algoritmu SCAN.

Kapitola se nejprve věnuje možným postupům vyhodnocení kvality jednotlivých metod a výběrem vhodných z nich pro naši konkrétní situaci. Dále následuje samotné vyhodnocení kvality metod na základě zvolených postupů. Část kapitoly je také věnována vyhodnocení výpočetní náročnosti implementovaných algoritmů. Na závěr jsou výsledky vyhodnocení shrnuty v přehledové sekci.

6.1 Způsoby vyhodnocení kvality metod

Tato sekce je zaměřena na výčet možných způsobů jakým lze vyhodnotit kvalitu postupů pro detekci anomálií v grafech, které byli popsány v literatuře a nebo použity v některých z vědeckých prací zabývajících se návrhem metod pro řešení tohoto problému.

- **Datová sada s vyznačenými anomáliemi** - Způsob evaluace kvality metody za použití datové sady, která v sobě má označené anomální elementy a slouží jako zdroj pravdy. Bohužel definovat co přesně pojem anomálie v grafu znamená není jednoduché a grafové datové sady které by obsahovali označené anomálie momentálně v podstatě neexistují. Většina metod si definuje anomálii vlastním způsobem a je dost pravděpodobné, že element jednou metodou označený jako jistá anomálie, může být pro jinou označen jako běžný element.
- ***Sence-making*** - Metoda kterou jsem nazval *sence-making* neboli hledání smyslu spočívá ve snaze vysvětlení významu detekovaných anomálií na základě nějaké hlubší znalosti o datech. Například pokud budeme

analyzovat graf představující komunikaci v rámci nějaké velké organizace a metoda označí některého z členů vrcholového vedení je snadné si tuto anomálii obhájit, protože je intuitivně očekáváno, že jedinec s takto ojedinělou pozicí v organizaci bude odlišnější než většina běžných pracovníků. Problémem této metody je, že většina veřejně dostupných dat je anonymizována a tedy ne vždy máme tyto potřebné informace k dispozici.

Do této kategorie patří i postupy při kterých nemáme žádnou hlubší znalost o datech, ale přesto se můžeme snažit vysvětlit nalezené anomálie na základě známých pravd z reálného světa. Příkladem může být detekovaná hvězda v grafu, který představuje komunikaci v rámci sítě. Tuto anomálii si pravděpodobně můžeme vysvětlit jako pravděpodobný DDoS útok, protože dochází ke komunikace mnoha různých uzlů směrem k jednomu.

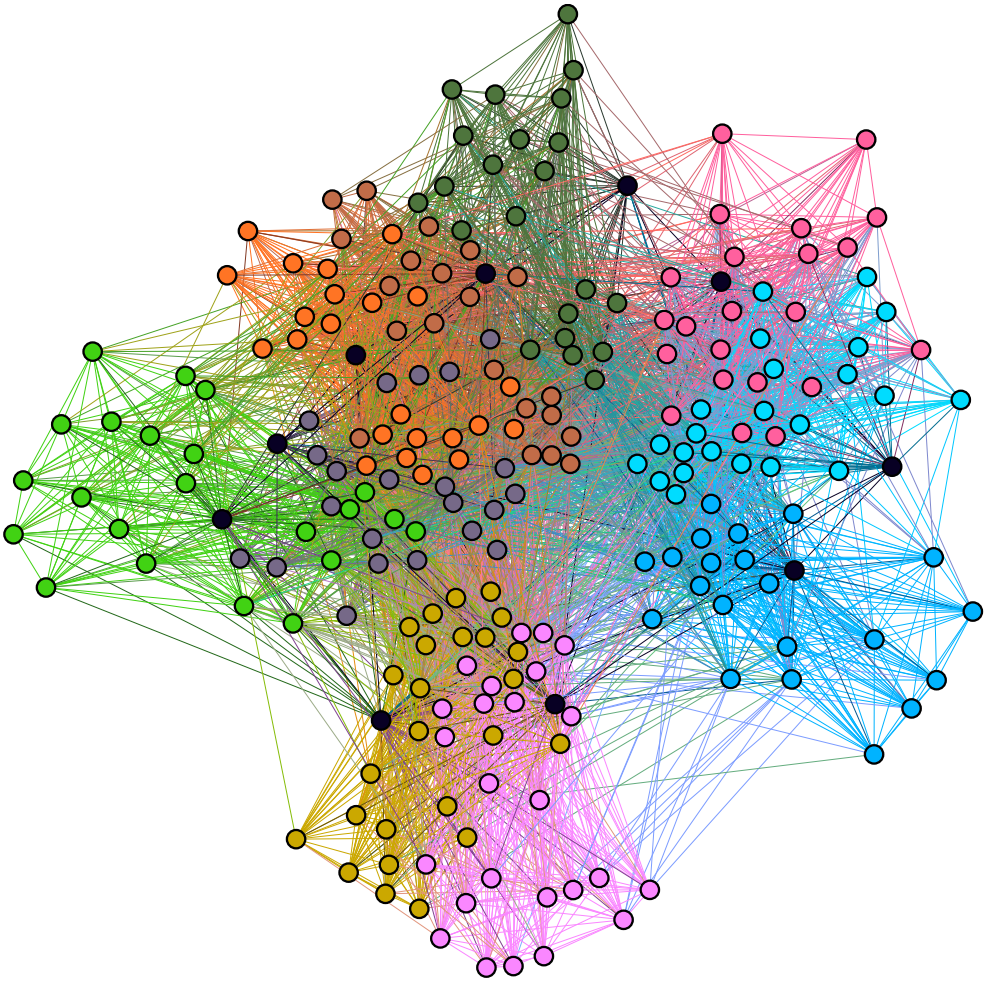
- **Generování syntetického grafu** - Existuje mnoho mechanizmů pro generování syntetických grafů, které se snaží napodobit realistický graf. Některé modely sami o sobě do grafu vkládají anomálie, u jiných je nutné anomálie vytvořit například náhodným dogenerováním hran a uzlů nebo jejich přeuspořádáním. Pro vyhodnocované metody je následně měřeno *precision* a *recall* při detekci vytvořených anomálií.

- **Vložení anomálie** - Funguje na podobném principu jako generování anomálií do syntetického grafu, pouze s rozdílem, že v tomto případě jsou anomálie generovány do reálného grafu. Tento postup může být dost problémový, protože daný graf už sám o sobě pravděpodobně bude obsahovat anomálie podobného typu jako jsou ty vkládané, což může ovlivnit výsledky.

V následujících sekcích se pokusím demonstrovat kvalitu všemi reálně použitelnými metodami. Validace vůči datové sadě s vyznačenými anomáliemi není možné aplikovat jelikož takové datové sady nejsou k dispozici. *Sence-making* je možné aplikovat na některé reálné grafy ze známých problémových domén, tento postup je rozepsán v sekci 6.2. Generování syntetického grafu a vložení anomálie jsou pravděpodobně nejčastěji používané postupy, na které je možné narazit v článcích zabývající se metodami pro detekci anomálií v grafech. Těmito postupy se budu zabývat v sekci 6.3.

6.2 Vizualizace detekovaných anomálií v grafech

Vizualizace detekovaných anomálií je provedena na několika grafech z reálného světa. Grafy v tomto případě byly záměrně voleny menší, aby je bylo možné je vložit jako obrázek do textu bez ztráty přehlednosti. Pro vygenerování náhledu grafů byl použit nástroj Gephi.



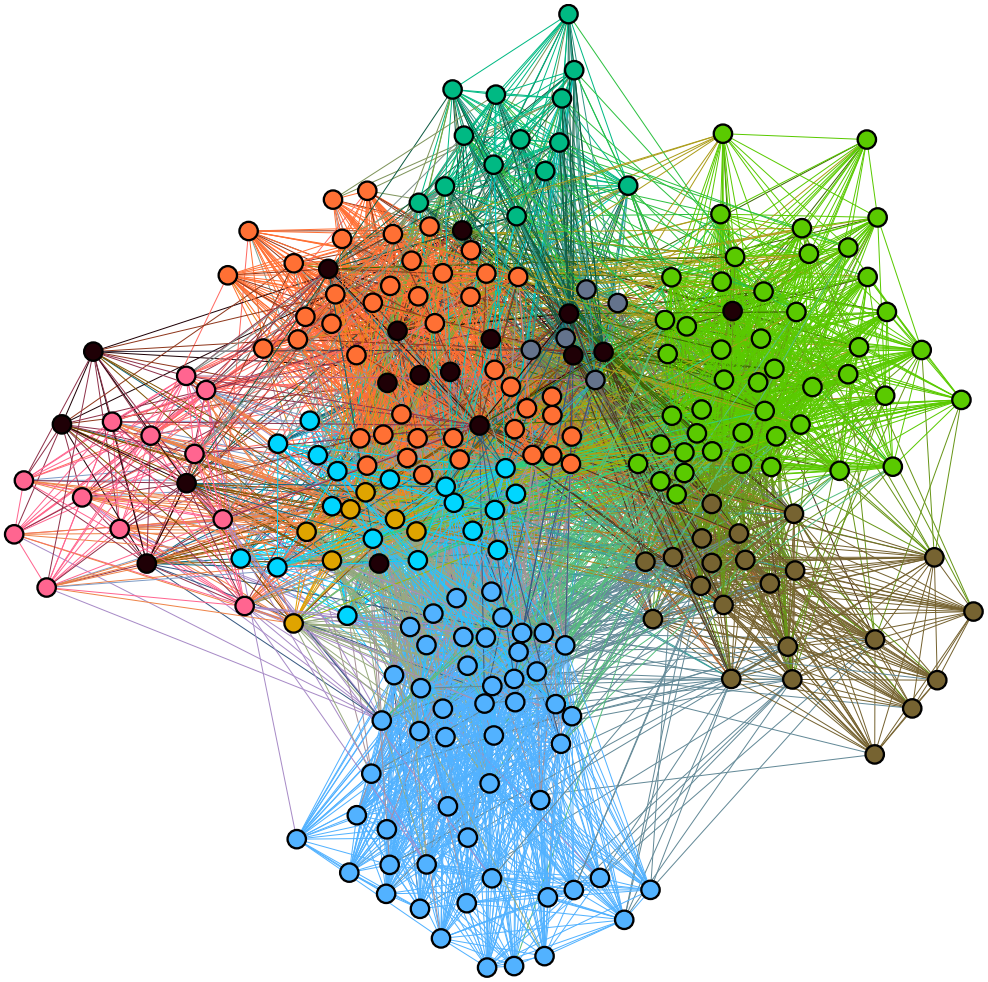
Obrázek 6.1: Primary school dataset - zobrazení tříd

6.2.1 SCAN a metoda rozšiřující SCAN

Pro vyhodnocení metod SCAN a jeho navrženým rozšířením byl vybrán graf [43]. Jedná se o graf zobrazující interakci tváří v tvář mezi učiteli a žáky na základní škole v průběhu jednoho. Uzel představuje žáka nebo učitele a hrana interakci mezi dvěma elementy. Uzly datové sady navíc obsahují atribut označující o žáka které třídy se jedná.

Na obrázku 6.1 je zobrazen celý graf a uzly jsou obarveny podle své příslušnosti do jednotlivých tříd, černě jsou znázorněni učitelé. V ideálním případě by SCAN měl detekovat jednotlivé třídy jako shluky a učitelé by pravděpodobně měli být označeny jako speciální uzly. Již na první pohled je však vidět, že některé třídy jsou vzájemně poměrně silně propojeny a pro algoritmus bude obtížné je od sebe odlišit.

Obrázek 6.2 znázorňuje rozdělení uzlů do shluků za použití algoritmu

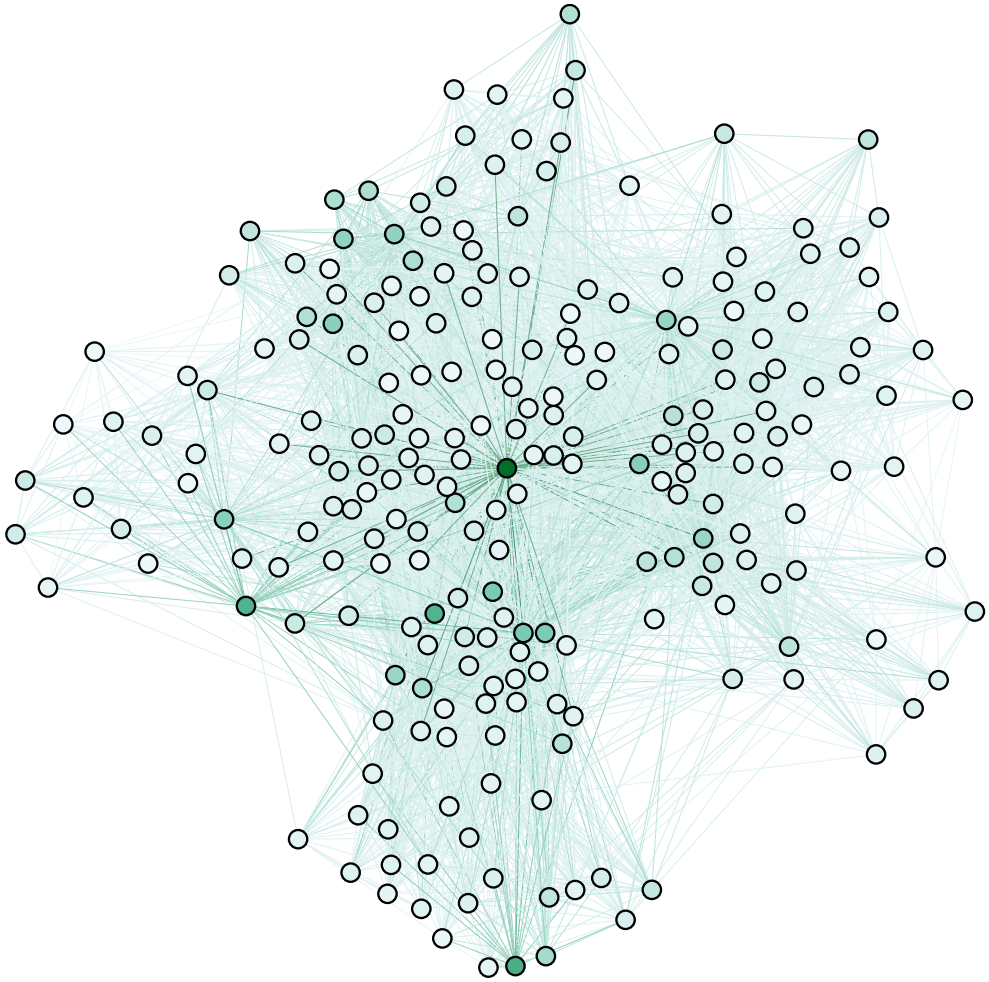


Obrázek 6.2: Primary school dataset - SCAN detekované shluky a *hub* uzly

SCAN s nastavenými parametry $\mu = 5$ $\varepsilon = 0.6$. Je zřejmé že předpoklady s obtížnou detekcí silně propojených tříd se potvrdily. Některé dvojice tříd jsou ve výsledném rozložení shluknuty do jedné, například dvojice tříd, která na původním obrázku 6.1 byly zobrazeny okrovou a světle růžovou (spodní část grafu) jsou nyní slity do jednoho shluku tmavější modré barvy. Jiné třídy však byly detekovány téměř bez chyby což je pozitivní.

Algoritmu se nepodařilo nalézt žádné speciální uzly typu *outlier*, což se dalo očekávat. *Outlier* je takový uzel, který není součástí žádného shluku a zároveň má propojení pouze na jeden ze shluků. Takové chování je u žáků na základní škole celkem nepravděpodobné.

Detekce učitelů jako speciálních uzlů se bohužel nepodařila pro žádný ze zkoušených vstupních parametrů. Pokud bychom se podívali podrobněji na to jakým způsobem jsou učitelské uzly propojeny v rámci sítě, můžeme pozorovat,



Obrázek 6.3: Primary school dataset - SCAN rozšíření, detekované anomálie

že se ve skutečnosti nijak zásadně neliší od běžných žáků své třídy. Za to se však podařilo detekovat několik žáku jako *hub* uzlů (konkrétně 17), které jsou v grafu znázorněny černě. Jsou to především ti, který mají vazby na více spolužáku z různých tříd současně, což je očekávatelné.

Na obrázku 6.3 je znázorněno vyhodnocení grafu pomocí navržené metody rozšiřující algoritmus SCAN. Pro připomenutí se jedná o detekci anomálních uzlů mezi uzly, které SCAN zařadil do některého ze shluků a označil je tak za normální. Nastavené parametry s shodují s těmi pro SCAN tedy $\mu = 5$ $\varepsilon = 0.6$, počet k nejbližších sousedů pro algoritmus LOF byl nastaven na 5.

Uzly na obrázku jsou obarveny v tónech zelené barvy, kde ty nejtmavější mají přiřazené nejvyšší anomální skóre. Rozšířená varianta detekuje úplně nový druh anomálie v grafu. Po bližším prozkoumání se jedná převážně o uzly, které jsou sice v rámci nějakého shluku, ale zároveň mají vazby na mnoho uzlů

z ostatních shluků.

Algoritmus našel zhruba 15 nových uzlů jejichž anomální skóre je vyšší než 1.5, což lze považovat za poměrně jasné anomálie a 17 dalších s anomálním skóre v intervalu $(1.25, 1.5)$, což jsou stále uzly, které stojí za hlubší prozkoumání. Zbylé uzly mají skóre nižší.

6.2.2 OddBall a metoda založená na změnách entropie

Ačkoli tyto dvě metody nemají žádný výrazný základ jako předchozí dvojice, jsou obě použitelné na obyčejný vážený graf, proto budou obě vizualizovány na stejném grafu.

Pro vizualizaci výsledků metod je použit graf sociální sítě z říše zvířat [44]. Konkrétně se jedná o graf dominance bisonů na národním bizoním ranči v Moiese (Montanta, USA). Uzly grafu představují 26 bisonů a hrana mezi uzly představuje pozorovaná dominantní chování dvou jedinců. Každá hrana má svou váhu, která vyjadřuje kolikrát toto chování bylo pozorováno.

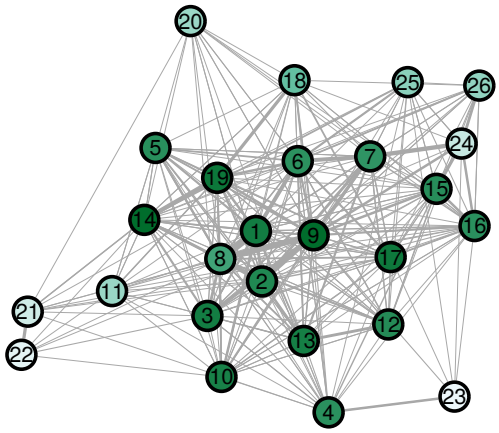
6.2.2.1 OddBall

OddBall dokáže detekovat několik různých typů anomálií a na každý z nich využívá různé metriky grafu. Výsledky bohužel nelze přehledně zobrazit v jednom grafu a proto jsou rozděleny do tří. Anomální skóre je opět znázorněno odstínem zelené barvy uzlu, čím tmavší tím anomálnější.

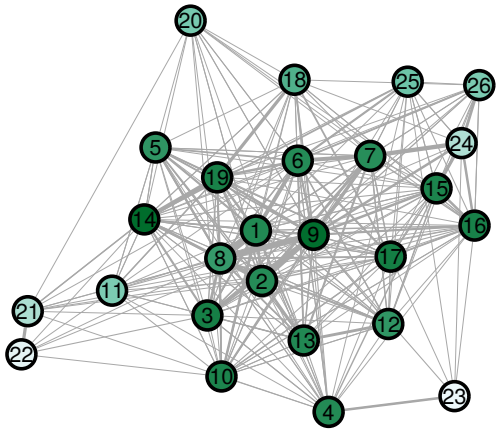
Clique-star je typ anomálie zobrazený na obrázku 6.4. Projevuje se jako podgraf, který je velmi blízko klice (silně provázané podgrafy) a nebo hvězdě. V grafu je možné pozorovat hned několik uzlů, které se vymykají očekávanému chování. Konkrétně se jedná o uzly 14, 9, 19, 17, 1 a několik dalších. Dá se usuzovat, že se jedná o skupinu samců kteří často terorizují své okolí, což se v grafu projevuje jako silně provázaný podgraf.

Dominant edge je způsobeno jednou nebo několika málo hranami s vyšší váhou než všechny ostatní v okolí zkoumaného uzlu. Anomálie je znázorněna na obrázku 6.5. Skupina uzlů s nejvyšší anomálností opět obsahuje stejné jedince jako v předchozím případě pouze v trochu jiném pořadí 9, 17, 15, 14, 7, 1... Graf je ve skutečnosti orientovaný a orientace reprezentuje, kdo je iniciátorem tohoto „dominantního chování“, algoritmus však bohužel orientaci nebere v potaz. V opačném případě by se do výsledků možná promítl i fakt, že vysoká anomálnost uzlů 9, 14 a 19 je ve skutečnosti způsobená častou dominancí ze strany uzlu 8, ten však tím že své okolí dominuje přibližně stejně do všech směrů vychází jako méně anomální.

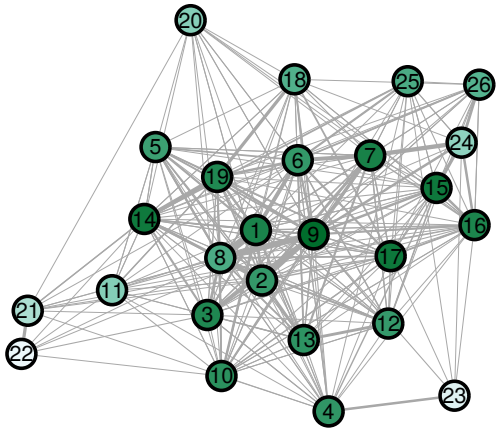
Heavy viciniy neboli těžké okolí grafu se projeví pokud kolem sebe má graf několik spojení na ostatní uzly s vysokou vahou v porovnání k jejich celkovému počtu. Tato anomálie je vyobrazena na 6.6. Na první pohled se může zdát, že se jedná o graf totožný s předchozím protože výsledky v tomto případě, alespoň co do pořadí uzlů vyšli téměř totožné.



Obrázek 6.4: Bison network dataset - OddBall detekce klik a hvězd



Obrázek 6.5: Bison network dataset - OddBall detekce *dominant edge*



Obrázek 6.6: Bison network dataset - OddBall detekce *heavy vicinity*

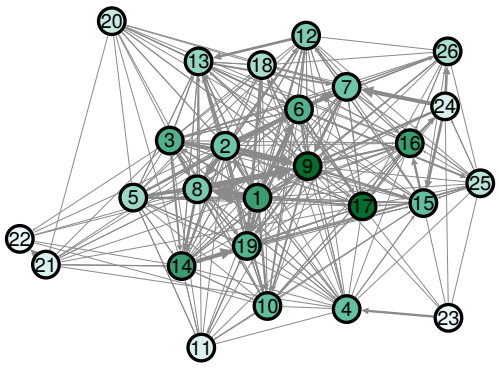
6.2.2.2 Metoda založená na změnách entropie

Jak bylo popsáno v předchozí kapitole, metoda využívá změn entropie při odebírání okolí jednotlivých uzlů a hledá ty, které do entropie přispívají nejvíce. Metoda může využít jeden ze dvou typů entropií, které jsou v textu nazvány jako vážená entropie a *non-parametric entropy*.

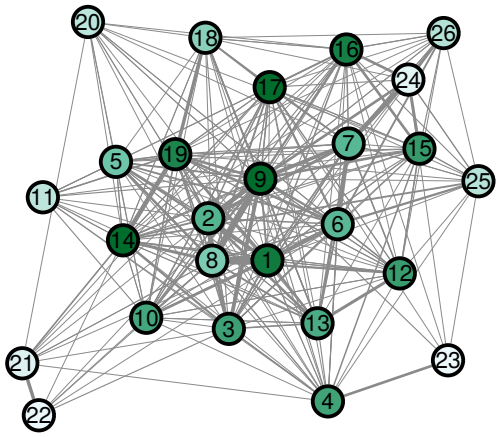
Hlavní výhodou této metody v porovnaní například s OddBall je to, že se na graf dívá v globálním měřítku. OddBall a mnoho dalších metod zkoumá jen velmi blízké okolí jednotlivých uzlů a v případě že nevyhovuje konkrétním očekávaným vlastnostem je uzel automaticky označen za anomální. To může mít za důsledek, že síť která inherentně neodpovídá očekávaným pozorováním bude z větší části považována za anomální což se může zdát jako nesmysl. K tomuto jevu přesně došlo v předchozí sekci věnované algoritmu OddBall. Zkoumaný graf pravděpodobně ne zcela odpovídal očekávaným vlastnostem a proto bylo poměrně velkému množství uzlů přiřazeno relativně vysoké anomální skóre, na druhou stranu alespoň OddBall přiřazuje skóre z nějakého spojitého intervalu, takže je stále možné vyfiltrovat ty uzly, které jsou vážně anomálnější.

Výhodou metod které přistupují ke grafu globálně je, že se dokáží lépe přizpůsobit jeho globálním vlastnostem a extrahovat tak jeho anomálie. Toto chování je možné pozorovat na obrázcích 6.7 a 6.8. Uzly s nejvyšším anomálním skóre jsou stále ty stejné jako při použití OddBall, s rozdílem že v tomto případě výrazně vystupují ty nejanomálnější uzly a proto je možné je snadněji jednoznačně separovat.

Je zde však nutné podotknout, že za tento globální náhled na graf je obvykle placeno vyšší výpočetní náročností.



Obrázek 6.7: Bison network dataset - metoda založená na změnách entropie - vážená entropie



Obrázek 6.8: Bison network dataset - metoda založená na změnách entropie - *non-parametric entropy*

6.3 Vyhodnocení na velkých grafech

Nyní se dostáváme k evaluaci metod na běžných grafech a vyhodnocení jejich schopnosti detekce anomálií. Stejně jako tomu bylo v předchozí sekci i zde budou algoritmy rozděleny do dvou skupin na metody aplikované na obyčejný graf (SCAN a jeho rozšíření) a aplikované na vážený graf (OddBall a metoda založená na změnách entropie).

Každá metoda bude vyhodnocována na dvou grafech, jeden z nich bude graf z reálného světa a jeden synteticky vygenerovaný. Do obou grafů budou vloženy anomálií několika různými způsoby které budou popsány dále. Metody budou posuzovány v tom jak jsou schopné detekovat vložené anomální elementy.

Grafy samy o sobě, ještě před vygenerováním vloženích anomálií, pravděpodobně budou obsahovat anomálie. Metody proto vždy budou nejprve spuštěny na originálním grafu a detekované anomálie budou poznamenány. Po vložení anomálií by algoritmus měl stále označovat původní anomálie, pokud je před vložení vyhodnotil jako anomálii a poté ne, je to bráno jako chyba.

Vyhodnocení bude zaměřeno na evaluaci *precision* a *recall* metod. Před popsáním těchto pojmů je nutné definovat jakým způsobem se detekované elementy mohou projevit ve výsledku:

- **TP** - *true positive* - Element byl vložen jako anomálie a byl správně označen za anomální.
- **FP** - *false positive* - Element nebyl vložen a přesto byl označen jako anomálie.
- **FN** - *false negative* - Element byl vložen jako anomálie, ale nebyl označen jako anomální.

- **TN** - *true negative* - Element nebyl vložen a nebyl označen za anomální.

Precision zjednodušeně vyjadřuje kolik z detekovaných anomálií jsou opravdu anomálie a vypočte se jako

$$precision = \frac{TP}{TP + FP}$$

Recall vyjadřuje kolik z vložených anomálií bylo opravdu označeno jako anomálie, vypočte se jako:

$$recall = \frac{TP}{FN + TP}$$

6.3.1 Generování syntetického grafu

Graf by neměl být generován zcela náhodně, ale měl by v sobě odrážet vlastnosti, které se v běžných grafech reálného světa vyskytují. Modelů pro generování takových grafů existuje celá řada, mezi ty nejznámější patří Erdős-Rényi, Barabási–Albert, Kleinberg, Watts–Strogatz a mnoho dalších.

Pro účely této práce byl vybrán model Watts–Strogatz [45]. Generovaný graf obsahuje vlastnosti malého světa [46] včetně krátkých průměrných cest a vysokého shlukovacího faktoru, což jsou běžné vlastnosti který se vyskytují například v grafech sociálních sítí. Navíc ze všech testovaných modelů si s tímto poradily implementované metody nejlépe. V jednoduchosti shrnuty princip Watts–Strogatz modelu je následující:

1. Pro zadané n počet uzlů a k průměrný stupeň uzlu se nejprve vygeneruje prstenec n uzlů kde každý uzel je spojen s $k/2$ následujícími uzly a $k/2$ předchozími uzly.

2. Následné je procházena každá z hran v grafu a se zadanou pravděpodobností p je jeden z jejích konců připojen k jinému uzlů. Výběr jiného uzlu je proveden s uniformní pravděpodobností ze všech uzlů grafu.

Při generování grafů byla použita již hotová implementace v knihovně NetworkX [47] implementovaná v jazyce Python.

6.3.2 Způsoby vkládání anomálií

Pro účely generování anomálií do grafu bylo vytvořena trojice algoritmů a jejich implementace je součástí výsledné knihovny GAD.

Random edge generator

Tento algoritmus generuje pouze nové propojení mezi uzly. Generátor vyžaduje 3 vstupní parametry:

- n - počet generovaných hran
- d - maximální vzdálenost nově propojených uzlů
- p - pravděpodobnost vícenásobného vygenerování ze zdrojového uzlů

Algoritmus náhodně vybírá zdrojové uzly pro generování hrany. Pro každý zdrojový uzel vybere náhodně uzel v jeho okolí maximálně vzdálený d. S pravděpodobností p toto opakuje, jinak vybere jiný zdrojový uzel dokud nevygeneruje n hran.

Cílem tohoto postupu je vygenerovat anomálie v blízkém okolí uzlů a nepropojovat je například napříč celém grafem nebo s uzly v jiném shluku a zároveň v grafu vytvořit hvězdy.

Egonet node generator

Na rozdíl od předchozího, tento postup generuje i nové uzly a v tomto případě je zaměřen na velmi blízké okolí uzlů, konkrétně jejich *egonet*. Algoritmus přijímá 2 vstupní parametry:

- n - počet generovaných uzlů
- p - pravděpodobnost propojení

Algoritmus náhodně vybírá uzly grafu a pro vybraný uzel získá seznam jeho sousedů. Vytvoří nový uzel spojí ho s každým ze sousedu s pravděpodobností p. Nakonec jej spojí i s vybraným uzlem. Pokračuje výběrem dalšího uzlu dokud jich nevygeneruje n.

Barabási-Albert generator

Jedná se o generování uzlů na základě známého modelu Barabási–Albert [48]. Algoritmus bere 2 parametry:

- n - počet generovaných uzlů
- m_0 - počet propojení nového uzlu

Algoritmus vygeneruje nový uzel, v náhodném pořadí prochází uzly grafu a ke každému nový uzel připojí hranou s pravděpodobností

$$p_i = \frac{k_i}{\sum_j k_j}$$

kde k_i je stupeň uzlu i a j představuje počet aktuálních uzlů v grafu, dokud nevytvoří m_0 spojení k novému uzlu. Toto opakuje dokud nevygeneruje n uzlů.

Tato metoda upřednostňuje silněji propojené uzly a ještě více zvětšuje jejich propojení.

Implicitně všechny generátory generují hrany s váhou 1. Je možné jim předat interval jako vstupní parametr ze kterého budou náhodně uniformě vybírány hodnoty vah nově vygenerovaných hran.

6.3.3 SCAN a jeho rozšíření

Oba algoritmy byly testovány na dvou grafech. Jeden z nich syntetický generovaný dle popisu výše, tedy na základě Watts–Strogatz grafového modelu a druhý je graf [49] z reálného prostředí konkrétně se jedná o část grafu ze sociální sítě Facebook.

Facebook graf

Anonymizovaný podgraf sociální sítě Facebook, obsahující 4039 uzlů a 88234 hran. Uzly představují uživatele a neorientované hrany přátelství mezi nimi. Graf je poměrně dobře shlukovatelný, jeho průměrný clusterovací koeficient je 0.6055.

Algoritmus SCAN v grafu detekoval zhruba 40 shluků, z toho největší zabírající 30% grafu, další dva největší detekované shluky zabírají 16% a 12%. Velikost ostatních shluků se pohybuje v řádech jednotek procent uzlů.

Zadané vstupní algoritmy byly $\mu = 8$, $\varepsilon = 0.35$ a $k = 8$.

Syntetický graf

Vygenerovaný graf obsahoval 4000 uzlů a 38437 hran, průměrný stupeň uzlů 19. Vhledem k požitému modelu, graf disponuje poměrně nízkou maximální vzdáleností uzlů (diametr grafu) a vysokým shlukovacím faktore. To se projevilo i na velikosti a rozložení detekovaných shluků. Největší z nich zabírá necelých 50% uzlů, další 18%, 13% a několik které zabírají méně než 10% grafu.

Zadané vstupní algoritmy byly $\mu = 8$, $\varepsilon = 0.55$ a $k = 8$.

6.3.3.1 Výsledky měření

Anomálie by v datech měly být něco výjimečného a pravděpodobně by se jejich počet měl držet v jednotkách procent velikosti grafu. Z toho důvodu byl počet anomálií vložených do grafu roven 150, což je pro grafy se 4000 uzly 3.75% jejich velikosti.

Tabulka 6.1: SCAN a SCAN Ext výsledky vyhodnocení na velkých grafech

		SCAN		SCAN Ext	
Gaf	**Typ anomáie**	**Prec**	**Rec**	**Prec**	**Rec**
Facebook	**Náhodná hrana**	96%	18%	96%	23%
	Uzel egonetu	100%	9%	100%	14%
	Barbási-Albert	100%	40%	100%	55%
Syntetický	**Náhodná hrana**	98%	24%	98%	31%
	Uzel egonetu	100%	7%	100%	8%
	Barbási-Albert	100%	37%	100%	55%

Do originálního grafu byly postupně vloženy anomálie třemi různými způsoby popsánými výše. Pro každý ze způsobů vložení anomálie byl aplikován třikrát a uvedené výsledku jsou průměrem třech měření.

Z tabulky 6.1 je zřejmé, že výsledků které byly uvedeny v článku popisující SCAN algoritmus [19] se dosáhnout nepodařilo. Důvodů proč tomu tak je pravděpodobně existuje více. Například to může být způsobeno rozdílnými vlastnostmi grafu, které byly použity nebo typem generovaných anomálií, které pro algoritmus byly obtížně detekovatelné.

Ve výsledcích je však možné pozorovat, že nejlépe si algoritmy vedly při detekci anomálie generovaných Barabási–Albert modelem. To se dalo očekávat, protože tento model dokáže generovat propojení mezi uzly napříč celým grafem a tedy je velmi pravděpodobné, že dojde k propojení vloženého uzlu s více shluky a jejich označení jako *hub*. Pro SCAN Ext jsou v tomto případě ještě o něco lepší.

Náhodné hrany byly generovány s maximální vzdáleností mezi propojenými uzly rovné 5, což se může zdát poměrně málo, ale vzhledem k poměrně malému diametru, kterými grafy disponují je to přiměřená hodnota. Tento typ anomálie má tendenci generovat hvězdy kolem náhodných uzlů z grafu a algoritmům se podařilo detekovat zhruba čtvrtinu z nich. Důvod je podobný jako v předchozím případě, generované uzly se svými vlastnostmi liší od ostatních a mají vysokou šanci na provázání více různých shluků

Nejhůře detekovatelné byly uzly vložené do egonetů ostatních uzlů. Tento typ anomálie je jen velmi lokální a pravděpodobnost vzniku *hub* uzlu tímto způsobem je velmi malá a vlastnosti které identifikuje SCAN Ext nejsou dostatečně rozdílné od ostatních uzlů v shluku.

6.3.4 OddBall a metoda založená na změnách entropie

Stejně jako v předchozím případě jsou i zde použity dvě sítě pro vyhodnocení kvality výsledků jedna z reálného světa konkrétně síť představující vzájemnou důvěru uživatel v rámci platformy zvané Bitcoin Alpha [50] a druhá synteticky generovaná síť vytvořená obdobným způsobem jako v předchozím případě.

6.3.4.1 Bitcion graf

Graf představuje vzájemnou důvěru u uživatel v rámci platformy zvané Bitcoin Alpha. Uzly reprezentují jednotlivé uživatele a spojení mezi nimi vyjadřuje důvěru jejíž míra je reprezentovaná váhou hrany -10 až 10. OddBall je aplikovatelný pouze na grafy s pozitivně ohodnocenými hranami, což zvolený graf nesplňuje proto byla ke všem hranám přičtena stejná konstanta, tak aby byly všechny hrany pozitivně ohodnoceny.

6.3.4.2 Výsledky měření

Do originálních grafů byly opět vloženy anomálie třemi různými způsoby v počtu rovnajícímu se 3.75% velikosti grafu. Měření pro každý z uvedených druhů způsobů vložení anomálie do grafu bylo provedeno třikrát a ve výsledcích je uveden průměr těchto měření. Výsledky pro metody OddBall jsou agregací všech typů nalezených anomálií, které dokáže detekovat, konkrétně *clique-star*, *heavy vicinity*, *dominant edge*. Kvůli výpočetní náročnosti metody založené na změnách entropie v grafu při použití *non-paremtric entropy*, která je více rozvedena v následující sekci, nebylo možné provést měření na grafu o velikosti 4000 uzlů. Z toho důvodu bylo měření provedeno na podgrafech o velikosti 500 uzlů, kde uzly byly vybírány náhodně s přihlédnutím k tomu aby graf zůstal spojitý.

V tabulce 6.2 jsou uvedeny výsledky měření. *Recall* jednotlivých metod je o poznání vyšší než u dvou předchozích metod, což je pravděpodobně způsobeno tím, že tyto metody jsou citlivější na zvolený druh generovaných anomálií. Pro OddBall byla vždy detekována alespoň polovina vložených anomálií. Co do typů anomálií, které OddBall dokáže detekovat se nejvíce projevovali *Dominant edge* což je pravděpodobně způsobeno tím, že v originálním grafu je většina hran ohodnocených poměrně vysokými hodnotami vah, zatímco hrany spojující generované uzly měli uniformní rozložení z intervalu $[0 - 20]$.

V porovnání s ostatními si s anomáliemi nejhůře poradila metoda s váženou entropií, jejíž *Recall* je poměrně nízký. To je pravděpodobně způsobeno příliš velkou jednoduchostí zvolené entropie a její necitlivostí na strukturální změny.

Nejlépe si stojí Metoda založená ne změnách entropie za použití *non-parametric entropy*. Bohužel však na úkor vysoké výpočetní složitosti.

Tabulka 6.2: OddBall a metoda založená na změnách entropie, výsledky měření na velkých grafech

		OddBall		**Entorpy weight**		**Entropy non-param**	
Gaf	**Typ anomáie**	**Prec**	**Rec**	**Prec**	**Rec**	**Prec**	**Rec**
Bitcoin	**Náhodná hrana**	100%	50%	100%	38%	100%	62%
	Uzel egonetu	100%	59%	100%	39%	100%	66%
	Barbási-Albert	100%	77%	100%	38%	100%	80%
Syntetický	**Náhodná hrana**	100%	55%	98%	41%	100%	61%
	Uzel egonetu	100%	48%	100%	35%	100%	72%
	Barbási-Albert	100%	61%	100%	35%	100%	77%

6.4 Výpočetní náročnost metod

Tato sekce se věnuje vyhodnocení výpočetní komplexity jednotlivých metod. Měření bylo prováděno na počítači s CPU Intel i5-3470 3.2 Ghz a 8 GB RAM.

6.4.1 Postup měření

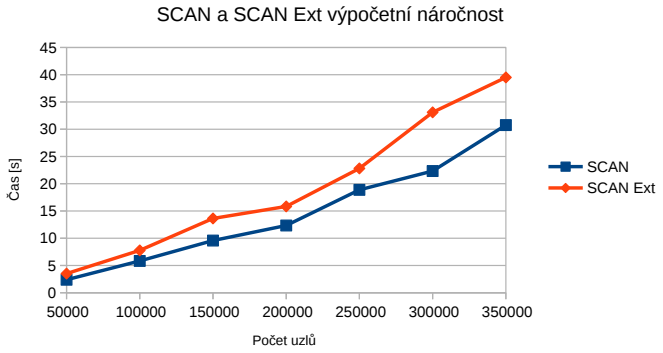
Měření bylo prováděno na grafu do kterého byly postupně náhodně dogenerovávány uzly a hrany aby dosáhl potřebné velikosti. Váhy hran do grafů generovaných pro OddBall a metodu založenou na změnách entropie byly generovány náhodně. Měření byla vždy prováděna třikrát na každém grafu a časy uvedeny v tabulkách jsou průměru těchto měření. Měření bylo prováděno pouze během výpočtu samotného algoritmu poté co již byla všechna data načtena ze souboru.

6.4.2 SCAN a jeho rozšíření

Dle odhadů v článku [19] by výpočetní složitost SCAN měla být omezena $O(n)$ kde n je počet uzlů. První graf byl generován poměrně řídký s průměrným stupněm uzlu pouze 7 a analyzovat jej i s počtem uzlů 350000 zabralo do minuty. Naměřené hodnoty jsou v tabulce 6.3 a grafu 6.9.

Tabulka 6.3: SCAN a SCAN Ext výpočetní složitost, řídký graf

Počet uzlů	**Počet hran**	**SCAN čas [s]**	**SCAN Ext čas [s]**
50000	107630	2.39	3.511
100000	238382	5.836	7.769
150000	383678	9.575	13.624
200000	536464	12.338	15.831
250000	692091	18.879	22.811
300000	841651	22.334	33.098
350000	925872	30.763	39.499



Obrázek 6.9: SCAN a SCAN Ext - Výpočetní složitost, řídký graf

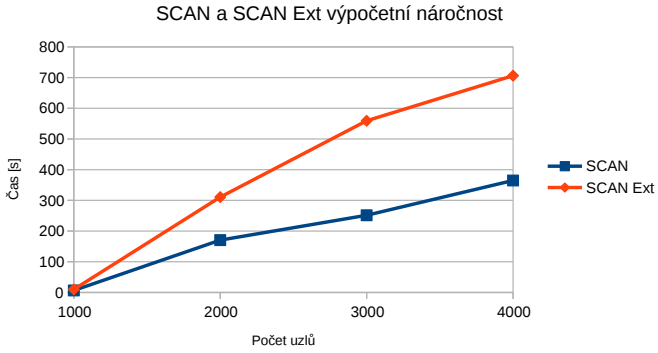
Při měření na podstatněji hustším grafu je výpočetní složitost o poznání vyšší. Při měření provedeném na grafu s průměrným stupněm 40 které je popsané v tabulce 6.4 a grafu 6.10 se již pro poměrně malý graf dostáváme do řádů několika minut.

Při profilování algoritmu se ukázalo, že nejvíce času je tráveno výpočtem podobnosti uzlů, které je v průběhu algoritmu vypočteno pro většinu dvojic sousedních uzlů a při jejíž výpočtu je nutné provést průnik množiny sousedů obou měřených uzlů.

Odstup času výpočtu SCAN Ext od SCAN je v tomto případě také o poznaní vyšší, to je způsobenu tím, že v hustě propojeném grafu je vyšší pravděpodobnost detekce velkých shluků, jejichž následná analýza poté zabere daleko více času. Trend vývoje potřebného času pro výpočet se však z měření zdá být stále lineární.

Tabulka 6.4: SCAN a SCAN Ext výpočetní složitost, hustý graf

Počet uzlů	Počet hran	SCAN čas [s]	SCAN Ext čas [s]
1000	9894	6.464	9.84
2000	37655	170.475	310.659
3000	71318	251.339	559.301
4000	88234	364.687	706.107



Obrázek 6.10: SCAN a SCAN Ext - Výpočetní složitost, hustý graf

6.4.3 OddBall a metoda založená na změnách entropie

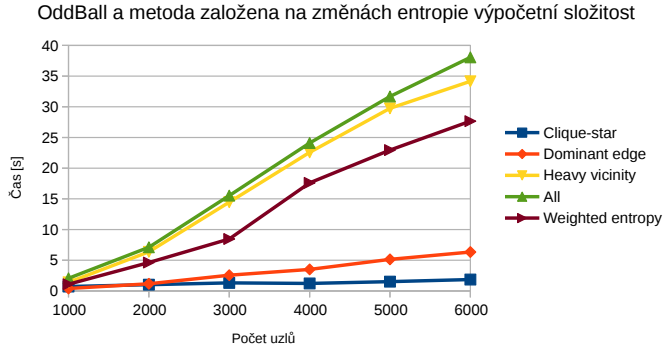
OddBall a metoda založená na změnách entropie byly měřeny na podstatě menších grafech čítajících maximálně 6000 uzlů. U OddBall je důvodem poměrně vysoká náročnost při výpočtu vlastních hodnot podgrafů a pro druhou metodu je to především kvůli její globální povaze. V tabulce 6.5 a grafu 6.11 jsou znázorněny měření detekcí jednotlivých druhů anomálií za použití algoritmu OddBall a metody založené na změnách entropie za použití vážené entropie.

V grafu je možné pozorovat, že detekce *Cluqie-start* anomálie a *Dominant edge* jsou o poznání méně výpočetně náročné protože vyžadují pouze snadno získatelné metriky podgrafů jako jsou stupně uzlů, počty hran a součet vah hran. Detekce *Heavy vicinity* je na tom o poznání hůře, protože vyžaduje výpočet vlastní hodnoty matice sousednosti podgrafu.

Porovnáme-li entropickou metodu s OddBall je zřejmé, že řádově se jejich složitosti nijak zásadně neliší.

Tabulka 6.5: OddBall a metoda založená na změnách entropie výpočetní složitost

Počet uzlů	Počet hran	Clique-star čas [s]	Dominant edge čas [s]	Heavy vicinity čas [s]	All čas [s]	Weight entropy čas [s]
1000	12054	0.719	0.397	1.399	2.056	1.091
2000	27636	1.02	1.166	6.369	7.114	4.62
3000	37691	1.309	2.565	14.455	15.53	8.463
4000	43747	1.221	3.503	22.582	24.101	17.606
5000	49543	1.515	5.144	29.751	31.678	22.951
6000	60074	1.856	6.334	34.155	38.054	27.662

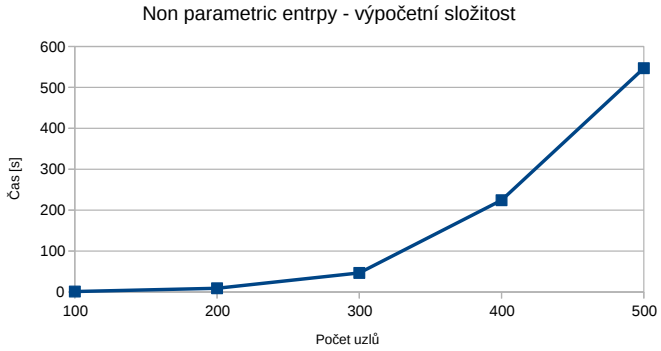


Obrázek 6.11: OddBall a metoda založená na změnách entropie výpočetní složitost

Jak se dalo očekávat při použití *non-parametric entropy* je výpočetní složitost podstatně vyšší a už pro poměrně malé grafy je velmi obtížně upočítatelná jak je zřejmé z tabulky 6.6 a grafu 6.12. Pravděpodobně by bylo možné dosáhnout i lepších výsledků, hlavním problémem je především omezená Gephi Toolkit sady nástrojů, který sám o sobě neumožňuje výpočet vlastních čísel matice sousednosti, nebo vůbec nějakým rychlím způsobem matici sousednosti získat. Problém by se dal vyřešit použitím vhodnějšího nástroje.

Tabulka 6.6: Metoda založená na změnách entropie (*non-parametric entropy*) výpočetní složitost

Počet uzlů	**Počet hran**	**Čas [s]**
100	498	1.058
200	1098	9.099
300	1875	46.586
400	2900	224.181
500	3811	546.873



Obrázek 6.12: Metoda založená na změnách entropie výpočetní složitost

6.5 Shrnutí vyhodnocení

Vyhodnocení kvality metod bylo prováděno jak na grafech z reálného světa tak na synteticky generovaných grafech na základě zvoleného modelu. Při měření nebyl pozorován žádný zásadní rozdíl ve výsledcích pro tyto dva druhy grafů. Bohužel kvality výsledků prezentovaných v článcích autorů jednotlivých metod se nepodařilo dosáhnout především kvůli nedostatku informací o použitých datech.

Přesto se měření dá považovat za úspěšné. Algoritmy jen málo kdy označili neanomální uzel za anomálii a z anomálních uzlů dokázali detekovat zhruba polovinu.

Při porovnání SCAN a navrženým rozšířením SCAN vyšlo rozšíření o několik procent lépe a podařilo se jím detekovat další druh anomálií mezi uzly, které SCAN označil jako normální.

Porovnáme-li OddBall s navrženou metodou založenou na změnách entropie v obou případech dochází k detekci podobných typů anomálních uzlů. Z výsledků se dá usoudit, že je entropická metoda schopná se lépe adaptovat na grafy, jejichž vlastnosti za normálního stavu zcela neodpovídají vlastnostem které OddBall očekává. Při měření na velkých datech si entropická metoda při

použití vážené entropie vedla poměrně špatně, pravděpodobně z důvodu příliš vysoké jednoduchosti metody a necitlivosti na strukturální změny v grafu. *Non-parametric entropy* se však co do kvality projevila jako velmi kvalitní, dávající dobré výsledky. Pro další experimentování by bylo vhodné navrhnout metodu, které by se podařilo spojit výhody obou přístupů pro výpočet entropie, tedy tak aby byla citlivá na strukturální změny a zároveň měla nízkou výpočetní komplexitu.

Co se výpočetní náročnosti týče, metody vykazovali lineární růst času výpočtu vzhledem v rostoucí velikosti grafu. Jedinou výjimkou byla entropická metoda s využitím *non-parametric entropy*, která byla již pro poměrně malé grafy neupočítatelná.

Závěr

Práce se zabývala problémem detekce anomálií v grafech a grafově orientovaných datech. Pro relativně novou a doposud málo prozkoumanou problematiku bylo představeno několik již existujících řešení, které k problému přistupují různými způsoby. Každé řešení bylo vyhodnoceno v několika definovaných kritériích, která mohou být směrodatná při výběru vhodné metody pro konkrétní problémovou doménu a dále uvedeno srovnání všech představených metod. Kvůli poměrně velkému rozsahu byla práce zaměřena především na detekci anomálií ve statických grafech.

Byly uvedeny dva zcela nové přístupy pro detekci anomálií v grafech, první z nich je založen na změnách entropie grafu při jeho editacích a druhý je rozšířením jednoho z existujících řešení, konkrétně algoritmu SCAN.

Výsledkem práce je knihovna implementovaná pomocí frameworku Gephi. Knihovna obsahuje implementaci dvou existujících algoritmů OddBall a SCAN, společně s vlastními navrženými metodami. Knihovna je zabalená a distribuovatelná jako jeden .jar soubor a připravena k použití v dalších projektech.

Na závěr práce je uvedeno vyhodnocení všech implementovaných metod a porovnání jejich kvality. Jelikož pro zkoumanou problematiku nejsou dostupné žádné použitelné datové sady, které by obsahovali označené anomálie, na základě kterých by se dali metody vyhodnotit, byla použita technika vkládání umělých anomálií do grafů a technika generování syntetických grafů. Vyhodnocení kvality navržených metod vyšlo velmi pozitivně. Metoda rozšiřující SCAN dokázala detekovat několik nových druhů anomálií, které se originálnímu SCAN detekovat nepodařilo. Při vyhodnocení metody založené na změnách entropie grafu, bylo experimentováno se dvěma druhy entropie, váženou entropií a entropií vycházející z vlastních čísel grafu. První z nich se ukázala jako příliš triviální a její výsledky nebyli tak dobré, naopak druhá z entropií se ve vyhodnocení projevila jako velice kvalitní v porovnání s ostatními metodami. Jejím problémem však je vysoká výpočetní náročnost, která ji omezuje jen na poměrné malé grafy.

Literatura

[1] Gephi. https://gephi.org/.

[2] Gephi : dynamic features. https://www.slideshare.net/Eldarion/gephi-dynamic-features.

[3] Akoglu, L.; McGlohon, M.; Faloutsos, C.: OddBall: Spotting Anomalies in Weighted Graphs. *Advances in Knowledge Discovery and Data Mining*, ročník 6119, 2010: s. 410–421.

[4] Local outlier factor. https://en.wikipedia.org/wiki/Local_outlier_factor.

[5] Chandola, V.; Banerjee, A.; Kumar, V.: Anomaly Detection : A Survey. *ACM Computing Surveys*, 2009: s. 1–72.

[6] Eberle, W.; Holder, L.: Anomaly detection in data represented as graphs. *Intelligent Data Analysis*, ročník 11, č. 1, 2007: s. 663–689.

[7] Kaur, R.; Singh, S.: A survey of data mining and social network analysis based anomaly detection techniques. *Egyptian Informatics Journal*, ročník 17, č. 2, 2016: s. 199–216.

[8] Ranshous, S.; Shen, S.; Koutra, D.; aj.: Graph based anomaly detection and description: a survey. *Data Mining and Knowledge Discovery*, ročník 29, 2015: s. 626–688.

[9] Sensarma, D.; Sarma, S. S.: A Survey on Different Graph Based Anomaly Detection Techniques. *Indian Journal of Science and Technology*, ročník 8, č. 31, 2015.

[10] Muller, E.; Sanchez, P. I.; Mulle, Y.; aj.: Ranking Outlier Nodes in Subspaces of Attributed Graphs. *Proceedings of the 4th International Workshop on Graph Data Management: Techniques and Applications*, 2013.

[11] Ranshous, S.; Shen, S.; Koutra, D.; aj.: Anomaly detection in dynamic networks: a survey. *WIREs Computational Statistics*, ročník 7, 2015: s. 223–247.

[12] Noble, C. C.; Cook, D. J.: Graph-Based Anomaly Detection.

[13] SUBDUE system. `http://ailab.wsu.edu/subdue/`.

[14] KDD Cup 1999 Data. `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`.

[15] Eberle, W.; Holder, L.: Discovering Structural Anomalies in Graph-Based Data. *IEEE International Conference on Data Mining*, ročník 7, 2007: s. 393–398.

[16] Beyer, K.; Goldstein, J.; Ramakrishnan, R.; aj.: When is nearest neighbors meaningful. *IDBT*, 1999: s. 217–235.

[17] Perozzi, B.; Akoglu, L.: Focused Clustering and Outlier Detection in Large Attributed Graphs.

[18] Moonesinghe, H. D. K.; ning Tan, P.: OutRank: A GRAPH-BASED OUTLIER DETECTION FRAMEWORK USING RANDOM WALK. *International Journal on Artificial Intelligence Tools*, ročník 20, č. 10, 2007: s. 1–18.

[19] Xu, X.; Nurcan Yuruk, Z. F.; Schweiger, T. A. J.: SCAN: A Structural Clustering Algorithm for Networks. *Proceedings of the 13th ACM International Conference on Knowledge Discovery and Data Mining*, 2007: s. 824–833.

[20] Chakrabarti, D.: AutoPart: Parameter-Free Graph Partitioning and Outlier Detection. *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2004: s. 112–124.

[21] Sun, J.; Qu, H.; Chakrabarti, D.; aj.: Neighborhood Formation and Anomaly Detection in Bipartite Graphs.

[22] Gephi - The Open Graph Viz Platform. `https://gephi.org/`.

[23] Gephi Toolkit. `https://gephi.org/toolkit/`.

[24] Gephi - Graph API. `https://github.com/gephi/gephi/wiki/Graph-API`.

[25] Gephi - Layout API. `https://github.com/gephi/gephi/wiki/Layout-API`.

[26] Gephi - Attributes API. https://github.com/gephi/gephi/wiki/Attributes-API.

[27] Gephi - Statistics API. https://github.com/gephi/gephi/wiki/Statistics-API.

[28] Gephi - Import API. https://github.com/gephi/gephi/wiki/Import-API.

[29] Gephi - Export API. https://github.com/gephi/gephi/wiki/Export-API.

[30] Gephi - Tools API. https://github.com/gephi/gephi/wiki/Tools-API.

[31] Gephi - Filters API. https://github.com/gephi/gephi/wiki/Filters-API.

[32] Gephi - Generator API. https://github.com/gephi/gephi/wiki/Generator-API.

[33] Gephi - Project API. https://github.com/gephi/gephi/wiki/Project-API.

[34] Gephi - LongTask API. https://github.com/gephi/gephi/wiki/LongTask-API.

[35] Triadic closure. https://en.wikipedia.org/wiki/Triadic_closure.

[36] JAMA : A Java Matrix Package. https://math.nist.gov/javanumerics/jama/.

[37] Shetty, J.; Adibi, J.: Discovering Important Nodes through Graph Entropy The Case of Enron Email Database.

[38] Mowshowitz, A.; Dehmer, M.: Entropy and the Complexity of Graphs Revisited. *Entropy — Open Access Journal*, ročník 14, 2012: s. 559–570.

[39] Independent set (graph theory). https://en.wikipedia.org/wiki/Independent_set_(graph_theory).

[40] Eagle, N.; Macy, M.; Claxton, R.: Network diversity and economic development. *Science*, ročník 328, 2010: s. 1029–1031.

[41] Breunig†, M. M.; Kriegel, H.-P.; Ng, R. T.; aj.: LOF: Identifying Density-Based Local Outliers. 2000.

[42] jLOF. https://github.com/bkaluza/jlof.

[43] DATASET: Primary school – cumulative networks. http://www.sociopatterns.org/datasets/primary-school-cumulative-networks/.

[44] DATASET: Bison. http://konect.uni-koblenz.de/networks/moreno_bison.

[45] Watts, D. J.; Strogatz, S. H.: Collective dynamics of 'small-world' networks. *Nature*, 1998: s. 440–442.

[46] Easley, D.; Kleinberg, J.: *Networks, Crowds, and Markets: Reasoning about a Highly Connected World*. 2010, 611–644 s.

[47] NetworkX - Software for complex networks. https://networkx.github.io/.

[48] laszlo Barabasi, A.: *Network science - The Barábasi-Alber model*. 2014.

[49] DATASEST - Social circles: Facebook. https://snap.stanford.edu/data/egonets-Facebook.html.

[50] Kumar, S.; Spezzano, F.; Subrahmanian, V.; aj.: Edge weight prediction in weighted signed networks. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, IEEE, 2016, s. 221–230.

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

CSV Comma seaprated value

RBF Radial basic function

MDL Minimal description length

GAD Graph anomaly detection

SNA Social network analysis

LOF Local outlier factor

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `jar` ..... adresář knihovnou GAD zabalenou jako .jar soubor
- `manuals` ..... adresář s uživatelskou příručkou GAD a JavaDoc
- `src`
  - `GAD` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Ondrej_Filip_2018.pdf` ..... text práce ve formátu PDF
  - `DP_Ondrej_Filip_2018.ps` ..... text práce ve formátu PS