FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Serverový modul pro hledání podgrafů
Student:	Bc. Vojtěch Bakaj
Vedoucí:	Ing. Marek Sušický
Studijní program:	Informatika
Studijní obor:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Zadáním je analyzovat, navrhnout, implementovat a otestovat nový modul do existující aplikace ClueMaker. Aplikace slouží pro vizualizaci vztahů z dat, kde mohou být informace získávány z různých tabulek či databází. Tento nástroj se používá v oblasti odhalování bankovních a pojistných podvodů, nebo investigativní žurnalistice.
Nový modul má uživateli umožnit označit podezřelý podgraf a přiřadit k němu anotaci. Anotované podgrafy se serializují do databáze a následně se nad daty hledají podobnosti s využitím strojového učení. Entity jsou uživatelsky definované a může jít o osoby, adresy, události, transakce... Každá entita může obsahovat atributy (částka, jméno, ulice, IP adresa, ...), které jsou vytvořeny uživatelem. Součástí učení musí být i pochopení a určení důležitosti jednotlivých atributů. Vše musí být generické. Je totiž možné, že v různých doménách a typech atributů bude třeba využít jiné porovnávací techniky. Porovnávání bude realizováno pouze v češtině.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Karel Klouda, Ph.D.
vedoucí katedry

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

V Praze dne 4. prosince 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Serverový modul pro hledání podgrafů

Bc. Vojtěch Bakaj

Katedra Aplikované matematiky
Vedoucí práce: Ing. Marek Sušický

9. května 2019

Poděkování

Rád bych poděkoval Ing. Marku Sušickému za jeho ochotu, vstřícnost a trpělivé vedení mé práce. Dále bych chtěl poděkovat společnosti Profinit EU s.r.o. za poskytnutí možnosti pracovat na diplomové práci v rámci jejich projektu. Děkuji také své přítelkyni Veronice Šotolové a rodině za podporu během celé doby studia.

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 9. května 2019 .....

Odkaz na tuto práci

Bakaj, Vojtěch. *Serverový modul pro hledání podgrafů*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Tato diplomová práce se věnuje analýze, návrhu, realizaci modulu a částem s ním spojeným. Tento modul by měl sloužit jako rozšíření produktu ClueMaker od firmy Profinit EU s.r.o., který se používá k vizualizaci dat z různých datových zdrojů. Tento nově implementovaný modul napomáhá uživatelům s detekcí podvodných činností na základě případů, které se staly v historii. V práci mimo jiné naleznete srovnání různých metod přístupů k této problematice, přičemž byla nakonec použita shluková analýza.

Klíčová slova ClueMaker, vizualizace dat, detekce podvodů, hledání podgrafů, grafový embeding, shluková analýza, hierarchické shlukování

Abstract

This master thesis deals with analysis, design, implementation of the module and parts associated with it. This module should serve as an extension of ClueMaker from company Profinit EU s.r.o., which is used to visualize data from various data sources. This newly implemented module helps users to detect fraudulent activites based on cases that have happened in history. Among other things, you will find a comparison of different approaches to this issue, where I will use a cluster analysis.

Keywords ClueMaker, data visualisation, fraud detection, subgraph search, graph embedding, clustering analysis, hierarchical clustering

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V posledních letech stále roste používání informačních technologií a s tím je spojená i výšší produkce dat. Bohužel existuje nespočet firem, které tato data stále hromadí a dále už je nezpracovávají, protože jich je nepřeberné množství a mnohdy nikdo neví, co všechno obsahují. Vzniká díky tomu potřeba vizualizace těchto dat z různých datových zdrojů a jejich propojení, které nemusí být na první pohled úplně zřejmé. Jedním z nástrojů, které takovou vizualizaci umožňují, je ClueMaker.

Tato aplikace je komerčním produktem společnosti Profinit EU s.r.o., kde se jako zaměstnanec podílím na její tvorbě v rámci této diplomové práce.

Motivace a cíl práce

Jak už bylo zmíněno v úvodu, ClueMaker je komerční produkt a je pro něj tedy klíčové, aby si udržoval konkurenceschopnost a snažil se stále poskytovat podobné nebo lepší služby jako jiné aplikace.

Za tímto účelem vznikla i tato diplomová práce. Jedná se o návrh a implementaci nové funkcionality, která umožňuje uživatelům vyhledávat podobné struktury s danými vlastnostmi z dat na základě předem označených referenčních vzorů. Použití je následující: uživatel si vytvoří jednotlivé kategorie představující například podvody, do kterých bude postupně přidávat označené struktury. Nakonec na pokyn uživatele nástroj provede vyhledávání všech podobných struktur a ty reportuje zpět uživateli.

Realizace této funkcionality má v reálném světě mnoho využití, například pro pojišťovací společnosti ke snadnému odhalování a prevenci pojistných podvodů, nebo například v investigativní žurnalistice pro vyhledávání souvislostí.

Struktura práce

Práce je rozdělena na šest logických celků. První část je o nástroji ClueMaker a jsou zde shrnuty všechny jeho funkcionality a důležité technologie, které byly při implementaci použity.

Druhá část se zaobírá analýzou celé problematiky, kde je nejdříve provedeno vymezení práce, popis jednotlivých problémů, se kterými jsem se v celé práci potýkal. Nakonec jsou zmíněny různé metody přístupů a z toho všechno je vytvořen závěr.

Třetí kapitola této práce je o sběru dat, které následně byly využity pro budoucí analýzu. Popíši, jak jsem k celému problému přistupoval a na jaké překážky jsem narazil.

Čtvrtá kapitola je o samotném návrhu a implementaci nové funkcionality. Nejdříve je zmíněna architektura aplikace a všech potřebných částí. Na závěr jsou detailně vysvětleny jednotlivé kroky implementace.

Předposlední část je o srovnání a vyhodnocení použitých metod na metrikách, které jsem si sám zadefinoval.

V poslední kapitole je pojednáváno o budoucím rozšíření nově implementované funkcionality, díky kterému by se mohlo dosáhnout lepších výsledků a snazšího a přímočařejšího přístupu k předzpracování dat.

Kapitola 1

ClueMaker

ClueMaker je nástroj, který slouží pro analýzu a vizualizaci dat. Je rovněž známý pod názvem SVAT (Smart Visual Analytic Tool), který vznikl v rámci projektu v jedné z největších českých bank v roce 2012 a sloužil převážně pro odhalování různých podvodů ve finančních transakcích.

Po zjištění potenciálu nástroje SVAT byla v roce 2015 aplikace kompletně přepsána a tím získal ClueMaker dnešní podobu. Od té doby je aplikace stále aktivně rozvíjena a jsou přidávány nové funkcionality.

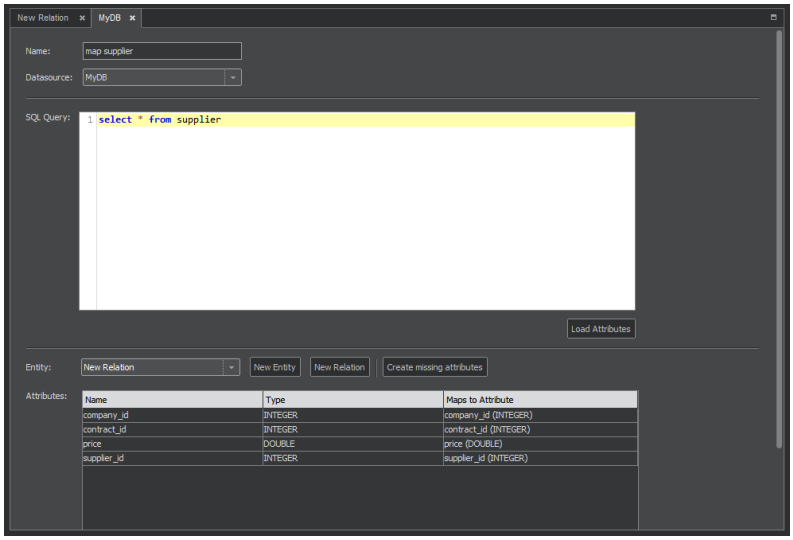
1.1 Přehled funkcionalit

Nástroj ClueMaker je rozdělen do dvou menších aplikací, kdy první z nich má na starosti samotné zobrazování dat a operace nad nimi. Druhá je konfigurátor, který spravuje datové zdroje a mapování jednotlivých atributů entit. K samotné aplikaci je možné přes konfigurátor navíc připojit ClueMaker server, který rozšiřuje aplikaci o další funkcionality. Jednotlivé části a jejich funkce jsou přiblíženy na následujících řádcích.

1.1.1 ClueMaker konfigurátor

Jak již bylo zmíněno výše, konfigurátor je samostatná aplikace, ve které mohou být zadefinované různé datové zdroje pro import dat. Dále se zde vytváří

Obrázek 1.1: Logo aplikace ClueMaker [1]



Obrázek 1.2: ClueMaker konfigurátor [2]

jednotlivé entity a vazby mezi nimi, které se následně mapují na uživatelem vytvořené datové zdroje. Jedna z dalších funkcionalit konfigurátoru je možnost zadefinovat si entitní či všeobecný reporty nebo dotazy potřebné pro jejich vykonání.

Výstupem konfigurátoru je poté soubor ve formátu ".sws", který v sobě nese všechny informace o datových zdrojích. Také obsahuje SQL dotazy pro mapování dat na entity a vztahy mezi nimi a samozřejmě SQL dotazy pro uložené vyhledávání a reporty.

1.1.1.1 Podporované zdroje dat

ClueMaker, respektive konfigurátor, podporuje všechny klasické relační databáze jako jsou PostgreSQL, Oracle, MySQL a Microsoft SQL, a dokonce také méně rozšířené databázové systémy, například Firebird, Impala, Teradata, Apache Hive, Aster Data, IBM DB2, Netezza a Splunk.

Další ze způsobů jak lze nahrát strukturovaná data do ClueMakeru je import z Excelu. V konfigurátoru se nachází speciální průvodce, který uživatele provede celým procesem importu a mapovaním dat na entity, respektive na nové uzly a vazby mezi nimi.

1.2 ClueMaker aplikace

V samotné aplikaci uživatel tráví většinu času. Je to z toho důvodu, že právě ta zobrazuje veškerá nakonfigurovaná data ve formě grafu, kde jsou jednotlivé entity reprezentovány uzly a vazby mezi nimi hranami. Taktéž každý uzel i

hrana má vlastní atributy a jejich výčet je dán typem uzlu či entity. ClueMaker tento graf může zobrazovat hned v několika rozloženích:

- **organické** - je zobrazení, ve kterém jsou uzly rozmístěné do tvaru hvězdic a lze tedy jednoduše z grafu vyčíst důležitost jednotlivých uzlů
- **hierarchické** - je seřazení uzlů, ve kterém hraje hlavní roli stupeň uzlů. ClueMaker podporuje konkrétně dvě verze tohoto zobrazení a to horizontální a vertikální
- **časové rozložení** - každý uzel nebo hrana má nějakou platnost, která se může vázat například na vytvoření uzlu nebo také na libovolný atribut. Díky tomuto zobrazení je možné uzly seřadit podle jejich časové platnosti od nejstaršího po nejnovější

Doteď jsem zmiňoval pouze zobrazování dat ve formě grafů. Další způsob, který tento nástroj umožňuje, je zobrazovat data se všemi atributy formou tabulky.

1.2.1 Import dat

Na základě vytvořené konfigurace, kde jsou definice všech datových zdrojů, mapování entit a atributů na SQL příkazy, je dalším krokem import dat, která se budou zobrazovat na plátně aplikace.

Jelikož výsledný graf může obsahovat statisíce až miliony uzlů, nebylo by z pohledu performance vhodné všechny vykreslit naráz. Proto import dat musí splňovat určitá kritéria atributů jednotlivých entit, podle kterých se budou entity vyhledávat v datových zdrojích. Po vyhledání dat je uživateli zobrazena tabulka s entitami, které splňují všechna kritéria a uživatel si je může následně importovat na plátno, resp. do grafu.

Na uzlech, které se nácházejí v grafu, lze pravým klikem jednotlivé uzly expandovat a tím dodatečně načíct všechny vazby spojené s daným uzlem. Jsou zde možnosti výběru konkrétní vazby, která nás zajímá, ale i kompletní vyhledání všech typů vazeb.

Další možností jak vložit na plátno nové entity a vazby je nástroj paleta. Zde jsou k dispozici zadefinované entity a vazby, případně existují již předem definované obecné entity v ClueMakeru. Poté jednoduchým kliknutím na plátno lze vytvořit nový uzel či hranu. Z palety je také možné vkládat textová pole nebo obrázek za účelem nějaké poznámky, nebo pro lepší pochopení dané problematiky.

1.2.2 Filtrování a vyhledávání

Nástroj ClueMaker umožňuje uživateli také filtrovat v grafu tak, že se vždy zvýrazní ta část grafu, která splňuje všechna kritéria nastaveného filtru. Filtro-

vat je možné na základě atributů a nebo typů hran či entit, jednotlivé filtrace lze libovolně skládat.

Podobně funguje i fulltextové vyhledávání, kde se na základě vyhledávání zvýrazní určitě část grafu. Stejná funkcionalita lze aplikovat i na tabulkové zobrazení, kde se vždy zvýrazní korespondující řádky.

Po dané selekci dat si lze tento mezivýsledek uložit a později se k němu vrátit. Další funkcí, kterou nad těmito uloženými výběry je možné provádět, jsou množinové operace.

1.2.3 Reporty

Pokud byl založen projekt s nějakou konfigurací, která je založena nad databázemi, mohou být spouštěny SQL dotazy s formátovaným výstupem, jak v grafovém formátu, tak ve stylu tabulky. V nástroji ClueMaker existují dva druhy reportů:

- **všeobecný** - je report, který nemá vazby na žádné entity a slouží zejména na spouštění složitějších SQL dotazů s parametry, které zadá uživatel

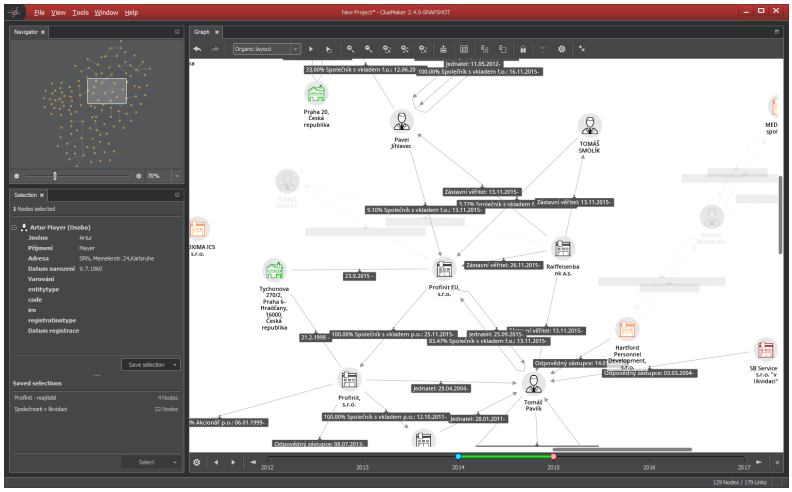
- **entitní** - je druh reportu, který je přímo vázaný na konkrétní entitu. V dotazech se zde používají nejen parametry zadané uživatelem, ale i samotné atributy dané entity

Výsledek reportu může být uložen v několika formátech. Pokud je report reprezentovaný tabulkou, tak existuje možnost výběru mezi formáty CSV, který se hodí pro budoucí zpracování, a nebo formátem XLS. Naopak pokud je výsledek reportu v grafu, tak je možnost využití exportu do obrázku ve formátu PNG.

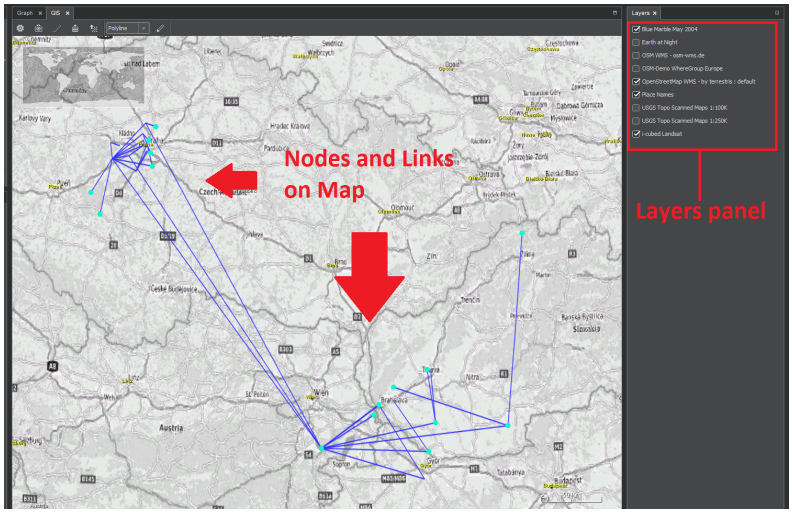
1.2.4 Časová osa

Každý uzel nebo hrana může mít časovou platnost od a do. Tato vlastnost může být doplněna manuálně, případně vytvořena společně s vytvořením uzlu. Také je možnost tato dvě pole namapovat na atributy dané entity. Poté lze využít nástroj časová osa, která se nachází ve spodní části obrazovky, kde si nastavíme začátek a konec intervalu. Uzly nebo hrany, které spadají do vybraného intervalu, budou zvýrazněny.

Jednotky času na ose je možné dynamicky měnit pomocí pohybu kolečka na myši. Také nástroj umožňuje nastavit přesnou délku intervalu, libovolně interval zužovat či rozšiřovat, případně celé časové okno posouvat o určitý krok.



Obrázek 1.3: ClueMaker ukázka časové osy [3]



Obrázek 1.4: ClueMaker ukázka GIS modulu [4]

1.2.5 GIS

Další z hlavních funkcionalit je geografický informační systém (zkráceně GIS). Tento systém se stará o zaznamenávání, ukládání, manipulaci, analýzu, správu a prezentaci prostorových a geografických dat. Respektive pokud se uživatel nachází v klasickém zobrazení grafu na plátně, tak si může zvolit libovolnou podmnožinu, případně celý graf, a zobrazit si tato data na mapě, která je realizována knihovnou WorldWind od NASA.

Tato funkcionalita vkládání entit na mapu ulehčuje uživateli analýzu dat z geografického hlediska, kterou je možné například využít při zkoumání dat

o pohybu uživatelů, údaje o telefonických hovorech, případně zjišťování sousedských vazeb u osob či firem.

Další z vlastností, kterou modul GIS umožňuje, je import entit z datového zdroje na základě nějaké geometrické výseče. Tu lze pomocí kreslítka vytvořit hned v pěti různých provedeních: elipsa, obdelník, úsečka (o nějaké šířce), kruh a nakonec výřez definovaný bodem a vzdáleností.

1.2.6 Export projektu

ClueMaker v současné verzi podporuje export aktuálního projektu do dvou formátů. Prvním z nich je export do vlastního formátu (*.spr), tzv. ClueMaker Project, který slouží k uložení rozpracovaného grafu a všech změn v atributech entit, hran a celkového rozpoložení grafu. Naopak se neukládájí selekce a věci týkající se konfigurace. Ta je uložena zvlášť, jak již bylo popsáno v kapitole výše. Dalším způsobem je export grafu do obrázku ve formátu (*.png), který může sloužit jako rychlý report dat pro obchodní oddělení přes emailovou komunikaci.

1.3 ClueMaker Server

Jak již název napovídá, jedná se o samostatně stojící server založený na technologii Spring Boot1.4.4. Pro plnou funkčnost server potřebuje být propojen se svojí databází. Momentálně podporuje PostgreSQL a Microsoft SQL. Samotný server umožňuje autentifikaci pomocí tokenu, který uživatel získá po přihlášení. To lze provést přes REST službu4.1.1 nebo přes grafické uživatelské prostředí, které server také obsahuje.

V aktuální verzi má server dva moduly, které budou na následujících řádcích blíže specifikovány.

1.3.1 Web crawler

Prvním modulem je Web crawler, který umožňuje vytvářet a zpracovávat dotazy na sociální sítě, fóra a RSS kanály. Dotazování je rozděleno u každého datového zdroje na dva typy dotazů: hledání (u sociálních sítí specificky fulltextové hledání) a dotazování se na podrobnosti již nalezených entit. Dotazy na podrobnosti o entitách lze provádět pouze na sociální sítě.

Podporované sociální sítě:

- Facebook
- Twitter
- Google+

- LinkedIn
- VKontakte

Podporovaná fóra:

- phpBB
- vBulletin

1.3.2 Web monitor

Web monitor dokáže prohledávat web podle hledaného výrazu nebo skupiny klíčových slov. Poté z výsledných dat dokáže vytáhnout entity, jako jsou například měny nebo čísla účtů. V tomto případě ale ClueMaker Server slouží jako aplikační rozhraní na posílaní dotazů a získávání výsledků z fronty, která je realizována technologií ActiveMQ.

1.4 Technologie

Jak klientská, tak i serverová část je napsaná v jazyce Java. Další společná technologie je Apache Maven, který se stará o správu, řízení a automatizaci sestavení aplikace. Oba nástroje ClueMaker a Konfigurátor jsou založeny na technologii NetBeans Platform. Na implementaci serveru byl použit Spring Boot, zejména kvůli své jednoduchosti a efektivitě práce. Tyto čtyři základní kameny ClueMakeru budou podrobněji popsány na následujících řádcích.

1.4.1 Java

Java je objektově orientovaný programovací jazyk. Java aplikace jsou typicky překládány do takzvaného "bytecode"a následně jsou interpretovány Java virtual machine, zkráceně JVM. Díky tomuto principu není nástroj ClueMaker závislý na platformě, tedy není potřeba ho zvlášť sestavovat pro každý typ operačního systému a může být bezproblémově používán na Windows, Linux nebo MacOS.

Tento jazyk byl vybrán pro implementaci hned z několika důvodů. Nejenom, že je multiplatformní, ale i díky jeho velké oblibě a rozšířenosti v programatorské komunitě k němu existuje nespočet knihoven a "best practices"pro tvorbu kvalitního softwaru.

Ve všech modulech a samostatných aplikacích, které souvisí s nástrojem ClueMaker, se používá Java ve verzi 8. Tato verze přináší několik zásadních změn a vlastností [5] do světa Javy. Dovolím si zmínit především ty, které se v aplikaci značně používají, a to sice Stream API a lamda funkce, díky kterým je kód přehlednější, čitelnější a vnáší tím funkcionální nádech.

1.4.2 Apache Maven

Maven je nástroj sloužící k automatizaci sestavení aplikací. Primárně byl vytvořen pro jazyk Java, ale dá se použít také například pro C#, Ruby nebo také Scala projekty. Jeho hlavním úkolem je definovat a zprostředkovávat dvě základní potřeby: jakým způsobem bude software sestaven, a za druhé jaké závislosti je potřeba dotáhnout a jakým způsobem je vkládat do aplikace.

Všechny tyto informace se definují v POM[1] souborech, které jsou vytvořeny pro každý z modulů. Každý z těchto souborů může mít jednoho předka, ve kterým se obvykle definují základní vlastnosti celého projektu, jako jsou například různé konstanty, verze jednotlivých knihoven a také je možnost přidávat a nastavovat různé pluginy a profily sestavení. Typickým příkladem je požadavek, aby se nám aplikace sestavovala jinak pro produkční prostředí a pro lokální ladění.

1.4.3 NetBeans platform

Když se řekne NetBeans, tak se zajisté každému vybaví známé IDE, ve kterém se snad na každé technické střední škole vyučuje programování. Méně známý, ale také poměrně populární produkt, se jmenuje NetBeans Platform. Je to nástroj sloužící pro tvorbu složitějších modulárních aplikací. Jedná se o generický framework pro tvorbu Swing Java aplikací. O pár řádků níže se pokusím bodově vypsat klíčové funkcionality[6], které tento framework nabízí:

- systém pluginů
- servisní infrastruktura
- práce se soubory
- window systém
- standardizovaný UI nástroj
- generická prezentační vrstva založena nad NodeAPI
- pokročilé Swing komponenty
- JavaHelp integrace
- management celého životního cyklu aplikace

Například VisualAPI, které zde také můžeme najít, se stará kompletně o zobrazování grafů v ClueMakeru.

[1] Project Object Model - strukturovaný XML soubor obsahující nezbytné informace o projektu.

1.4.4 Spring Boot

Spring je známý a velice populární framework pro vývoj Java EE aplikací. Spring Boot[7] je jakási nadstavba, která obsahuje klasický Spring framework a nad ním umožňuje vytvořit takzvanou "stand-alone"aplikaci. Tu lze jednoduše spustit bez nahrávání na server a různých konfigurací. To vše je díky už vestavěným serverům jako jsou Tomcat, Jetty nebo Undertow. Dále obrovská výhoda je, že Spring Boot už v sobě obsahuje základní závislosti a konfiguraci, takže není potřeba pro rychlé spuštění žádné speciální konfigurace serverů. Navíc v základní verzi má "production-ready"nástroje, jako jsou například metriky a kontrola stavu serveru.

1.5 Využití v praxi

ClueMaker v dnešní době využívá několik českých bank, pojišťoven, investigativních žurnalistů a také například Letiště Václava Havla. Tyto instituce ho primárně využívájí na odhalování finančních podvodů, respektive pojistných podvodů, různých propojení mezi lidmi a toky peněz.

Pro lepší představu uvedu jeden příklad z praxe. Na tomto konkrétním případu[8] byla pomocí ClueMakeru a jeho schopnosti vyhledávat propojení odhalena síť lidí spojených obchodními vazbami, a následně se díky tomu podařilo odhalit skutečné proudění peněz. Bez ClueMakeru nebo jinému podobnému nástroji by se takové vazby odhalovaly daleko složitěji nebo by to bylo až nemožné.

Kapitola 2

Analýza

V této kapitole si lze přečíst co je cílem této diplomové práce. Dále bude představena problematika, na které bude celá práce demonstrována, a poté shrnu problémy, se kterými jsem se během své práce setkal.

Následuje podkapitola, ve které zmíním určité pojmy, jež na následujících řádcích často užívám.

Na konci této kapitoly bude sekce řešící různé přístupy k této problematice, které nakonev sumarizuji.

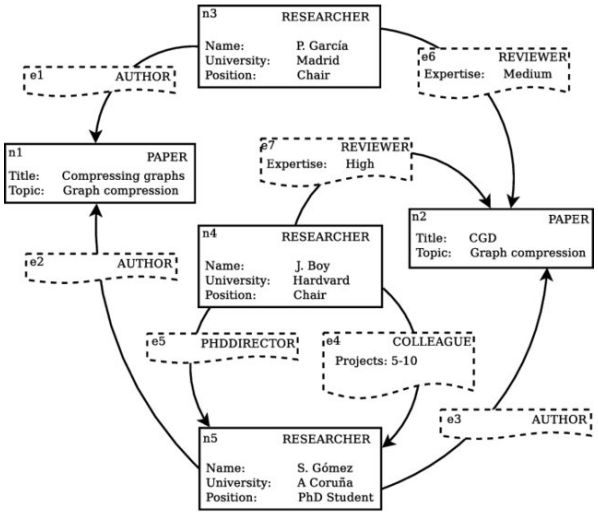
2.1 Cíl práce

Ještě než rozeberu samotný problém, tak nejprve popíši co je cílem této práce. Jedná se o návrh a implementaci nového modulu do nástroje ClueMaker, který by měl sloužit na odhalování podvodů. Jeho funkce by měla být následující: uživatel v grafu postupně označuje libovolné podgrafy, které zároveň i kategorizuje. Po sesbírání určitého množství těchto podgrafů se provede na pokyn uživatele, hledání podobných podgrafů reprezentujících stejný problém a ty následně budou reportovány zpět k uživateli.

2.2 Vymezení práce

Celá práce bude demonstrována na detekci "bílých koňů". Kdo nebo co je bílý kůň, si lze přečíst v následující kapitole 2.2.1.

Tento druh trestné činnosti jsem si vybral z toho důvodu, že již existuje vytvořená konfigurace pro ClueMaker, ve které jsou naagregovaná data z obchodního a insolvenčního rejstříku. Jedná se tedy o obrovskou databázi firem a osob s celou historií všech vzájemných vztahů. Navíc je většina těchto dat dostupná pro veřejnost.



Obrázek 2.1: Ukázka označkovaného grafu s attributy [9]

2.2.1 Bílý kůň

Bílý kůň je slangové označení pro osobu, která je nastrčena k páchání trestné činnosti, aby zakryla skutečného pachatele, který má z této činnosti osobní prospěch. Poté všechna vina padá na bílého koně. Do této role může být osoba donucena násilím nebo různými druhy vydírání, případně přemluvením, pokud se jedná o naivní nebo nevzdělanou osobu, která dokonce ani nemusí tušit, že se dopouští trestné činnosti.

Dalším příkladem bílých koňů jsou osoby, u kterých se předpokládá, že vzhledem k jejich věku či duševnímu zdraví bude soud přihlížet výrazně mírněji k výpočtu trestu nebo nejlépe od potrestání upustí.

Mezi nejznámější formy bílých koňů patří registrace takového koně jako podnikatele, který následně odebere určité množství zboží bez úmyslu za něj zaplatit. Další známý případ je takový, že se firma dostane do finančních potíží, následně se výrazně zadluží a v konečné fázi se firma přepíše na bílého koně se 100% podílem, a ten poté nese veškerou trestní zodpovědnost.

2.3 Popis problému

Jak už je z cíle práce patrné, jedná se o ”patern matching”s tím ztížením, že jde o multigraf, kde každý uzel či hrana má 0 až n atributů a navíc je vždy nějakého typu viz. obrázek2.1. Graf může mít až milióny uzlů a hran a není proto žádoucí, aby všechny výpočty probíhaly na klientské straně aplikace.

Dalším problémem jsou zde již zmiňované atributy a typy entit, které jsou uživatelsky definované a je tedy problém pochopit jejich význam, zda se jedná například o nějaký důležitý číselný údaj nebo jde pouze o nějaký identifikátor.

2.4 Slovník pojmů

Ještě než byl popsal, jak jsem postupoval při analyzování přístupů jednotlivých řešení tohoto problému, tak zmíním některé pojmy, které se budou v následujících odstavcích nekolikrát opakovat.

2.4.1 Embedding

Embedding[10](česky vložení) je obecně mapování, resp. konverze dat na vektory čísel. Jelikož české slovo "vložení"by na čtenáře mohlo působit zmatečně, budu nadále v textu používat slovo embedding. Pokud je řeč o embdedding v kontextu neuronových sítí, tak se jedná o kontinuální vektorovou reprezentaci diskrétních proměnných, která má malou dimenzi. Aby se dosáhlo těchto požadovaných vlastností, tak se na to typicky využívá dvouvrstvá neuronová síť, která se postupně zdokonaluje v této transformaci. Tento konkrétní embedding je v praxi velmi užitečný, protože díky němu můžeme redukovat dimenzi dat a reprezentovat smysluplně data v prostoru. Navíc snižujeme riziko prokletí dimenze.

Embedding se primárně používá k třem hlavním účelům:

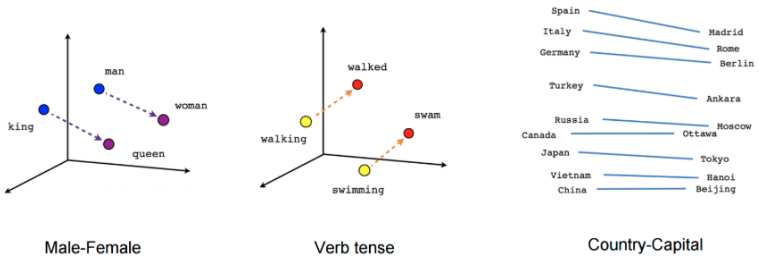
- Hledání nejbližších sousedů v embedding prostoru. Toto se využívá hlavně pro doporučovací systémy.
- Výsledné vektory se používají jako vstupy pro následovné strojové učení s učitelem. Je to z toho důvodu, že se daleko lépe pracuje s vektory, než s nějakými strukturovanými daty.
- Pro různé vizualizace dat a vztahy mezi nimi.

2.4.1.1 Grafový embedding

V této práci jsou data ve formátu grafu. Stálo by tedy za zmíňění, co je to grafový embedding[11]. Jde opět o transformaci grafu do vektoru nebo nějaké množiny vektorů, typicky matice. Embedding by měl zachycovat topoligii grafu, jednotlivé vazby mezi uzly a ostatní relevantní informace o grafu, podgrafech a hranách. Většinou platí, že čím více vlastností je zakódováno do embeddingu, tím se dosahuje lepším výsledků.

Grafový embedding lze zhruba rozdělit do dvou skupin:

- **Uzlový embedding** - Jedná se o reprezentaci každého uzlu jako jednoho vektoru. Tento způsob se nejčastěji využívá pro vizualizace, predikace stupně uzlu nebo také pro predikci spojení mezi dvěma uzly na základě uzlové podobnosti.
- **Grafový embedding** - Zde reprezentujeme celý graf jedním vektorem. Opět se tento druh embdeddingu používá pro predikce úrovně grafu a



Obrázek 2.2: Ukázka word2vec modelu [14]

vizualizace grafu jako celku. Nejčastěji má využití zejména v chemii, kde se využívá na porovnání chemických struktur.

2.4.2 Embedding prostor

Opět stejně jako u embeddingu je pojem embedding prostor poměrně krkolomný výraz, proto od tohoto momentu budu už používat jenom pojem embedding space[12]. Jedná se o prostor, který vznikne projekcí dat do vektorového prostoru. Lze si to snadno ukázat na word2vec[13] modelu, který dokáže konvertovat slova na vektory.

Většinou jsou požadovány od takového prostoru nějaké základní vlastnosti, například, aby podobná slova(mluvíme teď o podobnosti ve smyslu významu slova) byla reprezentována podobným vektorem, resp. nacházející se blízko sebe ve výsledném prostoru.

Další zajímavou vlastností je aritmetika nad těmito embeddovanými slovy. Dovolím si opět uvést příklad z word2vec: mám například vektory král, muž a žena. Pokud se poté vyřeší triviální rovnice, kde se odečte vektor muž od krále, a poté přičte vektor žena, tak výsledek je vektor královna.

Toto byl jeden z mnoha příkladů embeddingu. Vektorový prostor může mít ale i jiné vlastnosti, nejenom ty, co se týkají podobnosti dat. Záleží na tom, jak je celkový embedding proveden.

2.4.3 Laplacianova matice

V grafové teorii Laplacianova matice[15], často nazývána jako vstupní matice, Kirchhoffova matice nebo také disktrétní Laplacian, je matrice reprezentující graf, která obsahuje několik užitečných vlastností grafu. Zmiňuji jí zde z toho důvodu, že se často využívá pro konstrukci různých embdedding metod, které poté mají uplatnění ve strojovém učení.

Pokud máme graf G, který obsahuje n vrcholů, poté Laplacianova matice $L_{n\times n}$ je definována jako:

$$L = D - A$$

kde D je matice stupňů(matice o rozměru $D_{n\times n}$, která má na své diagonále stupeň i-tého uzlu) a A je matice sousednosti grafu G.

Pokud se jedná o orientovaný graf, pak Laplacianova matice je definována následujícín předpisem:

$$L_{i,j} := \begin{cases} deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_i \\ 0 & \text{otherwise} \end{cases}$$

Ještě bych si dovolil zmínit symetrickou normalizovanou Laplacianovu matici, která je definována následovně:

$$L^{sym} := D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I \ - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

Jednotlivé složky matice L^{sym} jsou dány předpisem:

$$L^{sym}_{i,j} := \begin{cases} 1 & \text{if } i = j \text{ and } deg(v_i) \neq 0 \\ -\frac{1}{\sqrt{deg(v_i)deg(v_j)}} & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_i \\ 0 & \text{otherwise} \end{cases}$$

2.4.4 Editační vzdálenost grafů

Editační vzdálenost grafů(v angličtině: graph edit distance)[16], je metrika pro měření podobnosti, resp. rozdílnosti dvou grafů, která se využívá v matematice a počítačové vědě. Tato metrika je založena na šesti základních operacích nad grafem:

- **Přidání uzlu** - přidání jednoho uzlu se značkou do grafu
- **Odebrání uzlu** - odebrání jednoho uzlu z grafu
- **Nahrazení uzlu** - změna značky/barvy jednoho uzlu
- **Přidání hrany** - přidání jedné obarvené/označkované hrany mezi dva uzly
- **Odebrání hrany** - odstranění hrany mezi dvěma uzly
- **Nahrazení hrany** - změna značky/barvy jedné hrany

Každá operace může mít určitou váhu, respektive cenu. Pokud je nalezen nejmenší počet operací, kterými z grafu g_1 dostaneme graf g_2, pak je jejich součet nazýván editační vzdálenost grafů g_1 a g_2 a značí se z anglické zkratky(graph edit distance) jako $GED(g_1, g_2)$.

Formálně definováno jako:

$$GED(g_1, g_2) = \min_{(e_1, ..., e_k) \in \mathcal{P}(g_1, g_2)} \sum_{i=1}^{k} c(e_i)$$

kde $\mathcal{P}(g_1, g_2)$ značí množinu editačních transformací z grafu g_1 do g_2 a $c(e) \geq 0$ je cena každé operace e.

2.4.5 Metody učení

Algoritmy strojového učení lze podle způsobu učení rozdělit do třech kategorií:

- **Učení s učitelem** - je nejznámější metoda strojového učení pro naučení funkce z trénovacích dat. Tato data se skládájí z dvojic vstupních objektů(nejčastěji se jedná o vektor atributů) a požadovaného výstupu. Výstupem funkce může být spojitá hodnota(při regresi) nebo může predikovat označení třídy vstupního objektu(při klasifikaci). Úlohou algoritmu je predikovat výstupní hodnotu funkce pro každý platný vstup, na základě učení z trenovacích dat. Aby tuto nelehkou úlohu dokázal splnit, musí se naučit zobecnit reprezentovaná data na nové vstupy nějakým smysluplným způsobem.

- **Učení bez učitele** - má na rozdíl od učení s učitelem pouze vstupní data bez výsledné množiny. Zejména se tedy používá k hledání struktur v datech a ke shlukování. Tento druh učení se tedy provádí pouze na vstupních datech. Jsou zde ale určitá omezení, například, že algoritmus nemá žádnou zpětnou vazbu díky absenci výstupní množiny dat. Proto způsob učení bez učitele funguje na principu identifikace společných rysů. Algoritmy poté reagují na přítomnost nebo absenci takovýchto rysů v nových datech.

- **Posilované učení** - je učení, které je založeno na zpětné vazbě. Většinou se jedná o agenta, který se nachází v nějakém prostředí a má sadu akcí, které může vykonávat, aby se přemístil z jednoho stavu do druhého. Za každou pozitivní akci, kterou agent provede, dostane odměnu a naopak za negativní akce ho čeká trest. V tomto učení se tedy nic nepredikuje a ani nehledají společně rysy, ale jde o nalezení vhodného modelu akcí, který by maximalizoval celkovou odměnu agenta.

2.4.6 Shluková analýza

Shluková analýza(anglicky cluster analysis)[17] je vícerozměrná statistická metoda, která se používá ke klasifikaci objektů. Jejím úkolem je třídit jednotky do skupin tzv. shluků, tak aby si jednotky náležící do stejné skupiny byly podobnější, než obejkty z ostatních skupin.

Shlukovou analýzu je možné provádět na množině objektů, kde každý z nich musí být popsán prostřednictvím stejného souboru znaků, které má smysl v dané množině sledovat.

Shlukovací metody lze členit do základních dvou metod podle cíle:

- **Hierarchické shlukování** - vytváří systém podmnožin, kde průnikem dvou podmnožin resp. shluků je buď prázdná množina nebo jeden z nich. Pokud nastane alespoň jednou druhý případ, jedná se o systém hierarchický. Lze si to tedy připodobnit k jakémusi větvení, zjemňování klasifikace. K tomuto typu shlukování se může přistupovat ze dvou stran. Rozlišujeme tyto přístupy na divizní(vychází se z celku jednoho shluku, která se poté dělí) a aglomerativní(vychází se z jednotlivých objektů resp. shluků o jednom prvku a ty poté spojují). Hierarchické shlukování nabízí více alternativních řešení, které mohou být reprezentovány například takzvaným dendrogramem.

- **Nehierarchické shlukování** - nevytvářejí žádnou hierarchickou strukturu, rozkládají množinu do podmnožin dle předem daného kritéria. První rozklad na podmnožiny se dále nedělí. Probíhá zde upravení tak, aby se optimalizovala vzájemná vzdálenost a odlišnost shluků, a aby objekty v nich byly rovnoměrně rozložené. Nalezení nejlepšího rozkladu vyzkoušením všech možných uspořádání shluků je většinou neproveditelné. Proto většina metod začíná s nějakým předem daným rozkladem, který dále upravuje a hledá optimální rozklad. Optimálního rozkladu dosáhneme, pokud nabude funkcionál kvality rozkladu své extrémní hodnoty. Nevýhodou je, že metody většinou končí pouze s lokálně optimalizovaným rozkladem.

2.5 Metody přístupu

Při analýze výše zmiňovaného problémů, jsem dospěl k několika různým postupům, jak lze tomuto problému čelit. Proto je v následujících podsekcích vyjmenuji a okomentuji, proč byla či nebyla zvolena právě daná metoda.

2.5.1 Generalizace podgrafů

První metoda je založena na generalizaci, když uživatel označí například deset podgrafů reprezentující stejný podvod, tak bych nejspíše očekával, že všechny tyto podgrafy budou mít nějaké vlastnosti společné. Princip, jakým způsobem bych dané grafy generalizoval do jednoho, je následující: mám deset grafů, přičemž každý může mít jiný počet uzlů a hran. Mým úkolem by bylo najít největší společný podgraf(samozřejmě bych bral do úvahy typy jednotlivých uzlů a hran), čímž bych získal deset stejných podgrafů. Nakonec by už zbývalo

vybrat společné atributy, které mají stejné nebo alespoň blízce podobné hodnoty, což už by však nebylo úplně jednoduché. Výsledkem by byl jeden obecný graf popisující daný podvod, který by potom posloužil jako reference pro vyhledávání stejného podgrafu ve zbytku dat.

Nakonec jsem od této myšlenky upustil, protože jednotlivé vzory podvodů by nešlo takto lehce generalizovat. Mohlo by se stát, že po takovém zobecnění by výsledek mohl nabýt i jiného významu, než bylo zamýšleno. Dovolím si uvést příklad z problematiky bílých koňů v kontextu přepisu firem na bílé koně. Berme v potaz pouze generalizaci dvou grafů, kdy majitelé firem A a B se dostanou do finančních problémů. Majitel firmy A přepíše firmu na fyzickou osobu, která má trvalé bydliště v cizině se 100% podílem ve firmě. Majitel firmy B ji přepíše na občana České republiky, který má trvalé bydliště na úřadě (bezdomovec). V tomto případě by generalizace vypadala následovně. Jednalo by se o přepis firmy se 100% podílem na fyzickou osobu, tedy všechny podezřelé faktory, jako je například trvalé bydliště na úřadu, by se nebraly vůbec v potaz. Mohlo by se tedy jednat o běžné přepsaní firmy kvůli prodeji nebo přepsání rodinného podniku z otce na syna.

2.5.2 Klasifikace pomocí neuronové sítě

Další metoda, nad kterou jsem se zamýšlel, bylo využití neuronových sítí ke klasifikaci podgrafů. Prvním krokem je tedy sestavení množiny dat na kterých by bylo prováděno učení. Množinu reprezentující podvodnou činnost bychom už měl, tu mi postupně vytvořil uživatel. Jsou zde ale dva zásadní problémy. První problém je sestavení druhé množiny, která by měla reprezentovat tu samou činnost, ale bez jakékoliv kriminální aktivity. Druhou otázkou je, jak určit hranici, kdy už mám dostatek vzorků dat, abych mohl začít s učením neuronové sítě.

Vrátím se tedy k hledání druhé množiny trénovacích dat. Existují dvě možnosti, jak bych tato data mohl získat. Najít podgrafy stejné struktury ve zbytku dat a brát v potaz ty, které jsou diametrálně jiné ve smyslu hodnot atributů nebo tuto množinu generovat na základě nějakých pravidel. Bohužel ani v jednom případě bych nikdy nemohl prohlásit, že se jedná o vzorek, který nereprezentuje podvod, protože zkrátka o něm nic nevím.

Tento způsob přístupu jsem už nadále nerozváděl, protože nemám k dispozici oanotovaná data, tedy žádné metody učení s učitelem mi v tomto případě nepomohou.

2.5.3 Využití shlukování

Poslední metoda je založena na shlukové analýze2.4.6 podobných podgrafů. Cílem by tedy bylo vytvořit embdedding space, kde bych se snažil aby podobné podgrafy byly blízko u sebe. Na tento vytvořený prostor bych poté aplikoval vhodnou metodu pro shlukování a vybral ty shluky, které by ob-

sahovaly označené podgrafy od uživatele. Ty bych následně prošel a vybral všechny nalezené nové podgrafy, které bych označil za podezřelé a reportoval je zpět na uživatele.

Mám zde ale opět několik problémů a otázek, které musím vyřešit. První otázkou je, jaký zvolit počet shluků. Na první pohled se nabízí zvolit pouze dva shluky, protože úkolem je klasifikovat prvky do dvou tříd. Jak jsem ale už popisoval výše, jeden typ kriminální činnosti může mít více tváří, tedy pouze dva shluky mi stačit nebudou. Dalším problémem je, jak vytvořit zmiňovaný embedding space, ale o tom více v následujících kapitolách.

2.6 Analýza embedding metod

Při analýze různých metod, které jsou zmíněny výše, jsem často opakoval, že bude potřeba provést nějaký embdedding. Proto jsem se v této sekci snažil zaměřit na různé "State of the art"embedding2.4.1 metody, protože v jakémkoli budoucím přístupu bude potřeba reprezentovat graf v nějaké vhodné matematické podobě, se kterou se dá v budoucnu pracovat. Hledal jsem tedy embdedding metody/frameworky, kde by figuroval multigraf s atributy a navíc k tomu všemu je každý uzel a hrana nějakého typu, resp. má nějakou značku.

Po několikadenním průzkumu jsem zjistil, že žádný univerzální embdedding na tento problém neexistuje, vždy tam jsou nějaká omezení. Některé algoritmy se zaměřují zejména na strukturu grafu, některé zase na typ uzlů a klasifikaci typu uzlů a v neposlední řadně existuje pár embdedding metod řešící i atributy uzlů.

Asi největším problémem se zdají být hrany. Většina algoritmů pracuje pouze s maticí sousednosti, která není pro můj problém úplně dostačující. Protože právě hrany pro detekci bílých koní hrají obrovskou roli, jelikož popisují vznik obchodních vazeb, jako je například přepis firmy s určitým podílem.

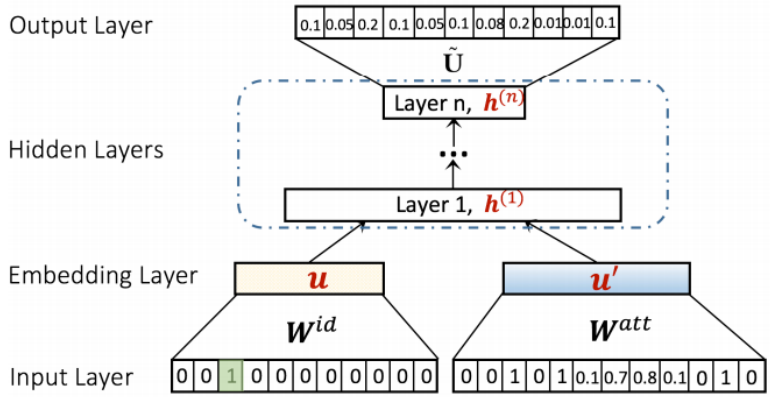
2.7 Konkrétní embedding metody

V této podsekci budou popsány embdedding metody, kterým jsem věnoval nejvíce pozornosti, případně jsem se jimi inspiroval.

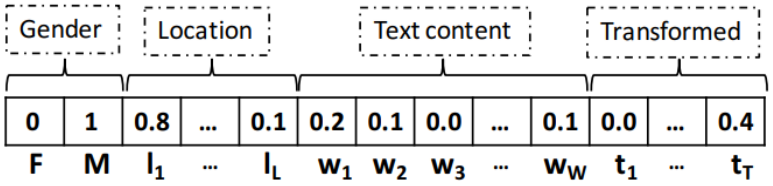
2.7.1 Graph2vec, node2vec a sub2vec

V první sekci zmíním hned tři metody najednou. Je to z toho důvodu, že se jedná o typické "State of the art"metody pro grafový embdedding. Všechny tyto zmíněné metody těží informace hlavně ze struktury grafů a podgrafů s výjimkou node2vec[18], která navíc přidává informaci o typu uzlu.

Graph2vec[19] se skládá ze dvou hlavních komponent. První procedůra vygeneruje kořenové podgrafy z každého uzlu daného grafu a druhá procedůra se učí provádět embdedding na těmito vygenerovanými podgrafy.



Obrázek 2.3: ASNE framework[21]



Obrázek 2.4: Ukázka vektoru atributů v ASNE síti [21]

Zbylé dvě node2vec[18] a sub2vec[20] používají k zjištění vlastností grafu náhodné procházky.

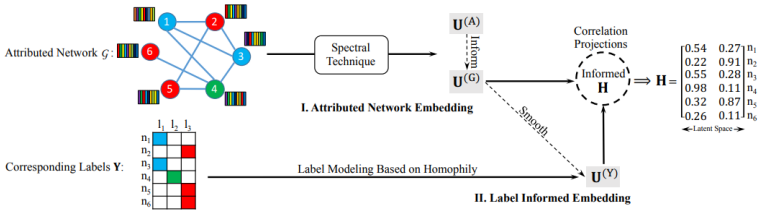
Všechny tyto metody se zejména využívají pro detekci komponent a predikci spojení, pro můj problém nejsou tedy úplně vhodné.

2.7.2 ASNE

Zkratka ASNE[21] znamená embdedding sociální sítě s atributy. Tedy jedná se o transformaci dat sítě do vektorového prostoru o malé dimenzi. Tento framework ke svému učení využívá jak strukturu sítě, tak i podobnosti jednotlivých atributů.

Jak vypadá struktura celého ASNE frameworku se můžete podívat zde2.3. Ta se skládá ze vstupní vrstvy, embedding vrstvy, **n** skrytých vrstev a nakonec výstupní vrstvy.

Vstupní vrstvu asi nemusím představovat, takže začnu s popisem embedding vrstvy, ta obsahuje dvě komponenty. První je "one-hot"projekce vektoru identifikátorů do vektoru **u**, který také zároveň obsahuje inforamci o struktuře grafu. Druhá komponenta jsou agregované atributy jednotlivých uzlů. Na obrázku2.3 značena vektorem **u'**. Zakódování atributů je prováděno tak, že se všechny konvertují do generického vektoru viz. příklad2.4 takového vektoru.



Obrázek 2.5: Struktura LANE frameworku [22]

Následuje skyrtá vrstvua což je vlastně vícevrstvý perceptron. Na vstupu první vrstvy bude spojen vektor **u** s vektorem **u'**, který je navíc vynásoben konstantou λ, reprezentující důležitost atributů. Všechny skryté vrstvy jsou plně propojeny.

Jak probíhá učení není pro tuto analýzu až tolik důležité, protože se jedná o učení s učitelem. Proto tento způsob nemůžu použít, z důvodů které jsem zmiňoval výše.

2.7.3 LANE

LANE(Label Informed Attributed Network Embedding) volně přeloženo jako embdedding sítě s atributy obohacen o typ uzlů, jak je vidět na schématu struktury2.5 celého frameworku.

Jak lze vidět z nákresu, celý framework se skládá ze dvou hlavních modulů, které se ke konci spojí do výsledného embdeddingu.

První modul se stará o embeding atributů a struktury sítě. Prvním krokem je tedy vytvoření affiní matice, která obsahuje informace o struktuře grafu(vychází z matice sousednosti s váhami). To samé vytvořím pro atributy, kdy sloupce matice budou jednotlivé atributy uzlů a řádky jsou již zmiňované uzly sítě. Obě matice jsou založeny na Laplacianově normalizované matici2.4.3. Jako spektrální techniku pro optimalizaci je použita dekompozice na vlastní vektory.

Druhý modul je jednoduchý embdedding značek uzlů. Je k dispozici ke každému uzlu kategorická proměnná nesoucí informaci o typy uzlu, ze které se vytvoří matice pomocí techniky "one-hot"dekompozice, díky které je výsledná matice $Y_{n\times m}$, kde n je počet vrcholů v grafu a m je počet unikátních značek. Nakonec se tato matice vynásobí tou samou transponovanou maticí a převede se na normalizovanou Laplacianovu matici.

Na závěr je potřeba výstupy obou modulů dát dohromady. Na vstupu jsou tedy tři matice- první z nich je A nesoucí informace o atributech jednotlivých uzlů, matici G, která reprezentuje strukturu grafu a v nesposlední řadě matici Y, která má v sobě informaci o typech uzlů, resp. značkách. Protože všechny tyto latentní reprezentace jsou omezeny odpovídajícím Laplacianem, jsou všechny tyto matice promítnuty do nového prostoru H, čímž si získají

větší míru svobody a flexibility. Pro uchování informací v matici G je využíván rozptyl projektované matice jako metrika jejich korelací, které jsou definováné následovně:

$$p_1 = Tr(G^T H H^T G)$$

$$p_2 = Tr(A^T H H^T A)$$

$$p_3 = Tr(Y^T H H^T Y)$$

Poté taková "loss funkce"pro všechny tři projekce je definována takto:

$$\underset{A,G,Y,H}{\text{maximalize}} \quad \jmath_{corr} = p_1 + p_2 + p_3$$

Díky tomu, že se maximalizuje p_1, p_2 a p_3 současně, je framework schopen se naučit kolerace mezi všemi maticemi.

2.7.4 PPNE

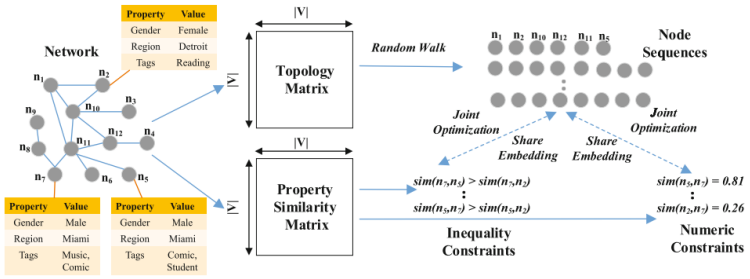
Framework PPNE(Propety Preserving Network Embedding)[23] je opět embdedding sítě, přičemž každý uzel má nějakou množinu atributů. Prvním krokem v PPNE je konstrukce dvou matic ze vstupní sítě(jak lze vidět na schématu2.6): první matice zachycuje topologii grafu, což je obyčejná matice sousednosti, druhá je matice atributů, která je rozdílná oproti matici, která je zmíněna například v algoritmu LANE. Zde má rozměr $A_{n\times n}$, kde n je počet uzlů a jednotlivé prvky matice jsou podobnosti uzlů. Metriku pro měření podobnosti uzlů autor ve své práci neuváděl, je to na uvážení každého, kdo by tento framework využil a zejména za jakým účelem.

Dalším krokem je aplikace algoritmu náhodné procházky na matici topologie k vygenerování množiny, která obsahuje sekvence uzlů. Z matice atributů se extrahuje množina podmínek, které zajišťují, že výsledné embedding vektory uzlů s podobnými atributy budou rozloženy ve výsledném, naučném prostoru blízko u sebe. Tyto zmíněné podmínky se dělí na dva typy: nerovnostní a číselné.

Nakonec se použije navržená objektivní funkce, která je odvozena z atributů pro každou z podmínek, a poté se spojí s další objektivní funkcí odvozenou z topologie sítě a společně jsou optimalizovány pro sdílení stejných parametrů. Tyto funkce už zde nebudu více rozebírat, lze si o nich přečíst více přímo v publikaci[23] autora práce.

2.8 Závěr analýzy

Ze všech analyzovaných metod ani jedna neobsahovala embdedding typů hran a ani jejich atributů, které jsou pro můj problém klíčové, protože popisují



Obrázek 2.6: PPNE framework [23]

vznik, resp. zánik vazeb, což lze přeložit do problematiky bílých koňů jako přepis firem nebo naopak zbavení se zodpovědnosti. S tímto problémem jsem ale počítal už od prvotního průzkumu, proto jsem volil algoritmy takové, které bych mohl rozšířit například o nový modul, díky kterému bych tam tuto novou funkčnost zakomponoval.

Na následujících řádcích bude popsáno, na jaké problémy jsem narážel při implementaci či aplikaci výše zmiňovaných metod. První trojici algoritmů: graph2vec, node2vec a sub2vec jsem zavrhl už po prvotním seznámením z důvodů, které byly zmíněny už v kapitole 2.7.1. Všechny tyto metody se příliš soustředily na strukturu grafu a atributy jednotlivých uzlů vůbec nebraly v potaz.

Další zkoumanou metodou byl framework s názvem ASNE. Zde jsem narazil na překážku, která vyžadovala pro své učení mít oanotované data, resp. probíhalo zde učení s učitelem2.4.5. V mém případě jsem měl oanotoavné pouze bílé koně od uživatele a druhá množina pro mě byla neznámá. Neuronová síť byla postavena a učena hlavně pro predikci spojení a klasifikaci typu uzlů. Po další výzkumu jsem objevil práci, kde autor využíval tuto síť pro měření podobností grafů, ale jednalo se pouze o aproximaci editační vzdálenosti grafů2.4.4, samozřejmě opět nad oanotovanými grafy.

Dáší zkoumanou metodou byl PPNE framework. Zde se už jednalo o klasický grafový embedding založený nad náhodnou procházkou. Narážel jsem zde ale na problémy s atributy. Graf, který byl na vstupu, mohl být složen z jednotek až desítek různých typů entit. Každá z entit má většinou disjunktní množinu atributů. Z toho plyne, že matice atributů, která obsahuje podobnosti jednotlivých uzlů či hran, byla velmi řídká. Což ve výsledku mělo na celý embedding negativní vliv.

Poslední metoda byla LANE. Do této metody jsem vkládal nejvíce důvěry. Jedná se o metodu založenou na faktorizaci matic, která dokáže zachytit globální strukturu grafu a všech atributů za cenu vysoké časové a paměťové náročnosti. Tyto dvě nevýhody nehrály pro mou práci zase takovou roli, protože v mém případě se jednalo o poměrně malé podgrafy. Bohužel právě

velikost grafů byla v pokračujícím zkoumání velikým kamenem úrazu. Kvůli malému počtu entit, výsledné matice po embeddingu byly skoro totožné. Zkoušel jsem různě experimentovat s váhou atributů oproti struktuře grafu, měnit výstupní dimenzi embdeddingu, která je samozřejmě ze shora omezená počtem uzlů, kvůli dekompozici na vlastní vektory a stále to nepomohlo natolik, abych tuto metodu mohl dále použít.

Kapitola 3

Sběr dat

Dalším důležitým krokem bylo sesbírání dat. Samozřejmě nikde není veřejný seznam bílých koní, ze kterého bych mohl čerpat, proto můj první impuls byl, že si přes ClueMaker načtu uzel, který představuje například adresu městského úřadu Prahy 1 a přes tento uzel najdu všechny osoby, které sídlí na této adrese. K těmto osobám si poté dotáhnu všechny vazby, který reprezentují vztah fyzické osoby k podnikatelské, díky kterým také dostanu ke každé vazbě entitu představující firmu. Poté bych mohl expertním pohledem zkoumat vývoj firmy v čase a zjišťovat, zda se tam jedná o bílého koně a nebo ne. Má to bohužel jeden háček- pokud to nedokážu nijak podložit důkazy, tak daného člověka nemůžu považovat za bílého koně ani přes jeho nestandardní chování.

Proto jsem se uchýlil k možnosti, že budu sbírat bílé koně na základě známých a medializovaných kauz. Vyhledávání kauz přes různé zpravodajské stránky by bylo neefektivní. Jako nejlepší způsob se mi osvědčilo na základě doporučení mého vedoucího diplomové práce, vyhledávání skrze přepisy, které poskytuje Česká televize. Konkrétně se jedná o přesné přepisy z pořadu Reportéři ČT a dostalo se mi tak obsáhlé databáze různých článků. Bylo tedy mým dalším úkolem najít relevantní články s tématikou bílých koní a extrahovat z nich jména firem a lidí.

V následujících sekcích detailně popíši, jak jsem tyto data získával.

3.1 Vyhledávání přepisů

Jako první věc, kterou jsem potřeboval, byly jednotlivé odkazy na články přepisů. K tomu jsem použil vyhledávač google a jeho vyhledávání s operátory a klíčovými slovy. Konkrétně jsem vyhledával všechny soubory s příponou ***.doc** na stránkách `http://www.ceskatelevize.cz` v sekci reportéři, výsledný výraz vypadal následovně **"filetype:doc site:ceskatelevize.cz reporteri"**.

Zkoušel jsem i jiné přípony souborů, jako jsou například ***.txt** či novější formát produktu Microsoft Word ***.docx**, ale zde jsem dostal vždy maximálně jednotky výsledků, které jsem prošel manuálně a všechny byly irelevantní.

Výsledkem vyhledávání bylo přibližně tři sta reportáží. Poté jsem vytvořil web crawler[2], který získal seznam všech absolutních odkazů na články ze všech nalezených stránek.

3.2 Stažení obsahu

Další částí bylo stažení jednotlivých dokumentů pro budoucí zpracování. Stahování jsem prováděl po jednom souboru, který jsem hned přečetl a zjistil, zda je relevantní pro moje potřeby. Zde jsem ale narazil na problém se čtením zastaralého world formátu ***.doc**. Proto jsem musel každý soubor nejdříve zkonvertovat do novějšího formátu ***.docx** přes nativní rozhraní, které poskytuje windows a pro python má vytvořenou klientskou knihovnu. V zápětí jsem soubor ihned přečetl a zkontroloval, jestli obsahuje tématiku bílých koní. To jsem zjišťoval na základě klíčových slov ve všech různých tvarech. Nakonec jsem si všechny relevantní články uložil do paměti pro další zpracování.

3.3 Vyhledávání a indentifikace entit

Po tom, co jsem měl k dispozici už všechny přepisy rozhovorů připravené, tak následovala extrakce a identifikace entit. Na tuto problematiku jsem zkusil použít dvě nejznámější knihovny obsahující už předtrénované modely, které teď budou krátce představeny.

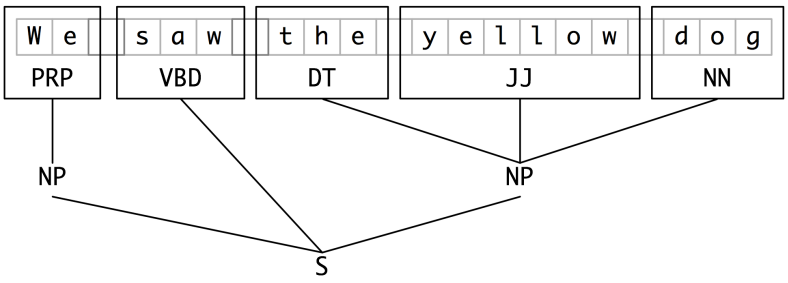
3.3.1 NLTK

NLTK[24] je přední platforma pro vytváření Python programů pro práci s daty lidského jazyka. Poskytuje několik snadno použitelných rozhraní k více než padesáti korpusům a lexikálním zdrojům.

Prvním krokem byla tokenizace jednotlivých článků a převedení slov do základního tvarů. Poté jsem využil předučený model, který NLTK knihovna poskytuje ke klasifikaci slovních druhů. Výsledkem je, že z každého slova se stane dvojice, jako je například tato *('kůň', 'NN')*. Tento zápis znamená, že slovo 'kůň' je podstatné jméno. Následně se tyto jednotlivé tokeny zaškatulkují do větších seskupení, kterým se říká chunky(česky kusy). Tyto seskupení se většinou vytváří na základě regulárních výrazů, které popisují různě po sobě jdoucí sekvence slovních druhů. Další zajímavou vlastností chunků je, že chunk může být tvořen opět z chunku, né jenom z tokenů. Typický příklad je chunk reprezentující větu, díky tomu vzniká nad námi analyzovaným textem stromová struktura, jak lze vidět například na ilustraci3.1.

Při vytváření chunků, NLTK dokáže k jednotlivým chunkům představující entitu přiřadit na základně opět předučeného modelu typ, jako je například

[2]Web crawler je specializovaný internetový bot, který prochází internet za nějakým účelem, typicky kvůli indexování nebo za účelem sběru konkrétních dat.



Obrázek 3.1: Stromová reprezentace jednotlivých chunků [25]

osoba, organizace nebo geografické uzemí. Poté mi stačilo už jenom projít tento strom a všechny entity, které jsou typu ”organizace”vybrat a tím jsem získal názvy firem, ve kterých figuroval nějaký bílý kůň.

3.3.2 Spacy

Spacy[26] je jedna z dalších knihoven poskytující rozhraní pro zpracování přirozeného jazyka. Nabízí podporu pro více než 49 jazyků, bohužel český jazyk neobsahuje. Spacy nabízí ale vícejazyčný univerzální modul, který jsem použil. Na rozdíl od NLTK je založen na hlubokém učení[3], architektura modelu řešící rozpoznávání entit zatím nebyl zveřejněn, ale autor knihovny naznačil, že se jedná o kombinaci konvolučních a rekurentních neuronových sítí.

Použití Spacy je opravdu jednoduché, zavolá se na vstupní text jedna hlavní funkce, která zprocesuje celý dokument a vrátí všechny potřebné inforamce včetně pojmenovaných entit.

3.4 Závěr

Obě knihovny dokázaly najít snad všechny entity představující organizace ve všech přepisech. O trochu hůře si vedla knihovna Spacy, protože včetně organizací vracela i plno nesmyslných slov, jako jsou například: městská, zatímco, toho a tak dále. Je asi pochopitelné, že to je zapříčiněno absencí předem naučeného modelu nad českými daty.

I přes to všechno bylo nakonec potřeba jednotlivé reportáže projít, vyhledat dané organizace a ujistit se z kontextu reportáže, že se jedná opravdu o podvodnou firmu. Tímto způsobem jsem získal první množinu dat, která reprezentuje bílé koně. Získaných vzorků nebylo mnoho, můžeme se bavit o jednotkách. To ale není na škodu, protože se tímto přibližuji reálné situaci. Většinou se bude jednat o jednotky vzorků, ke kterým se budou hledat podobné podgrafy ve stovkách až tisících grafů.

[3]Jedná se o pomerně novou disciplnínu v rámci strojového učení, která se zabývá využitím algoritmů(především neuronových sítí) s velkým počtem vrstev reprezentující data.

Kapitola 4

Návrh a implementace

Tato kapitola se zabývá návrhem a implementací celého pluginu a je rozdělena do dvou hlavních podsekcí, podle toho, jestli daná funkce zapadá do klientské části, resp. do pluginu aplikace ClueMaker nebo se bude jednat o rozšíření CluerMaker serveru.

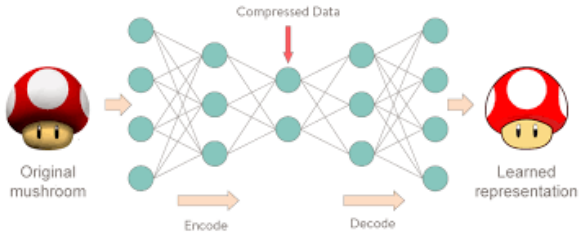
4.1 Slovník pojmů

Na následujících řádcích jsou vyjmenovány různé pojmy s vysvětlením, pro lepší pochopení textu.

4.1.1 RESTful webové služby

REST(Representational State Transfer)[27] je styl architektury pro vytváření webových služeb. Komunikace mezi klientem a serverem je zajištěna protokolem HTTP, který nabízí několik standardních operací:

- **GET** - je základní operace, která provádí čtení z nějakého zdroje. Tato operace by neměla mít žádné postranní efekty viditelné mimo aplikaci a musí být idempotentní.
- **HEAD** - vrátí pouze hlavičky obsažené v reprezentaci tohoto zdroje.
- **POST** - se používá pro vytvoření nového zdroje.
- **PUT** - se používá pro nahrazení stávajícího zdroje novým na základě zaslaného stavu. Operace musí být idemponentní.
- **DELETE** - smaže zadaný zdroj. Operace musí být idemponentní.
- **OPTIONS** - dala by se označit za nápovědu pro klienta. Vrací metody, které může klient na tento zdroj použít.



Obrázek 4.1: Příklad autoenkodéru[29]

4.1.2 Autoenkodéry

Jedná se o typ umělé neuronové sítě[28], která se skládá ze dvou částí. První část kóduje vstupy a druhá je dekóduje zpět ve snaze o co nejmenší chybu. Počet neuronů vstupní vrstvy je tedy totožný s počtem vstupů a cílem je produkovat na výstup stejné data jako na vstupu.

Na první pohled se může zdát, že taková síť nemá moc využití, protože se vlastně snaží o identitu a mohla by jednoduše překopírovat vstup na výstupy. Pokud je ale zvolena výstupní dimenzi nižší než vstupní, tak neuronová síť bude nucena se chovat úsporněji a snažit se tak o generalizaci. Naopak pokud výstupní dimenze bude stejná jako vstupní, tak se autoenkodéry využívají k odstranění šumu v datech. Případně se může využít prostřední vsrtva ke komprimaci dat, jak lze vidět na ilustraci4.1.

4.1.3 NP-úplné problémy

Jedná se o třídu problémů, jejichž statut není zatím známý. Nebyl objeven žádný polynomiální algoritmus, který by je dokázal vyřešit. Ale zároveň se nepodařilo dokázat, že by takový algoritmus nemohl existovat.

Tedy jediné aktuální možné řešení je použití hrubé síly, to znamená že se postupně zkouší všechna možná řešení, dokud se nenajde to správné.

4.1.4 OneHot kódování

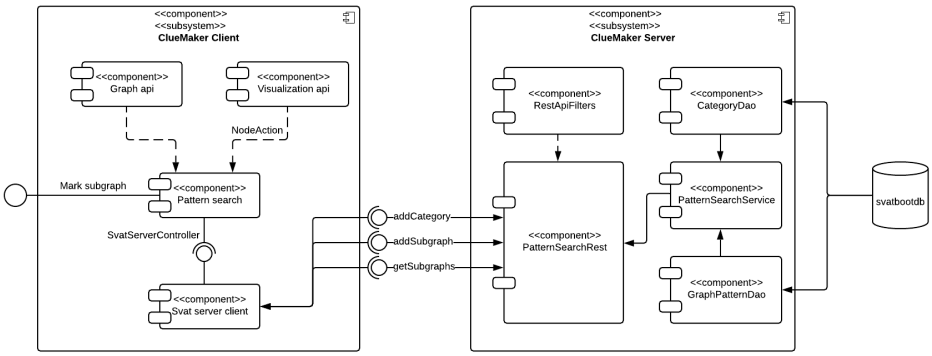
OneHot kódování[30] by se dalo považovat za standardní přístup ke kategorickým datům. Jedna se o zakódování kategorické proměnné tak, že pro každou různou hodnotu se vytvoří nový sloupec boolovských hodnot. Hodnoty v řádcích jsou poté tvořeny tak, že hodnota je rovna 1 pokud řádek obsahuje hodnotu sloupce, v ostatních případech 0. Takto vypadá například implemetace4.2 kódování v pythonu:

OneHot kodování má dobrý vliv na různé modely, ale počet nových sloupců se rovná počtu jediněčných hodnot, to je jeho hlavní nevýhoda. Pokud data obsahují velký počet kategorických proměnných, pak rozšířit předzpracování o toto kodování, může způsobit velké problémy s pamětí.

```
ce_one_hot = ce.OneHotEncoder(cols = ['color'])
ce_one_hot.fit_transform(X, y)
```

	color_1	color_2	color_3	color_-1
0	1	0	0	0
1	0	1	0	0
2	1	0	0	0
3	1	0	0	0
4	0	0	1	0
5	0	0	1	0

Obrázek 4.2: Ukázka ”OneHot”kódování[30]



Obrázek 4.3: Komponentová architektura

4.2 Architektura

Než přejdu k popisu jednotlivých součástí, tak bych nejdříve představil můj návrh komponentové architektury4.3 a popsal jednotlivé komponenty.

4.2.1 ClueMaker

První komponenta je ”Graph Api”, která se stará o veškeré operace a transformace s grafem. Jedním z hlavních úkolů této komponenty je zajistit, aby všechny akce, které se budou provádět nad grafem, byly prováděny transakčně. Díky tomu všechny operace, které ClueMaker poskytuje a lze je povádět asynchroně, nemohou graf dostat do nějakého nekonzistentního stavu.

Další z komponent je ”Visualization api”. Jak už název napovídá, jedná se o modul zobrazující graf na plátno aplikace. Hlavním úkolem této komponenty je zaregistrovat a odchytávat všechny události, jako jsou například označení nebo zaměření daných uzlů a hran. Dalším z důležitých úkolů je zvýraznění určité části grafů. Tato funkcionalita se například používá ve fulltextovém hledání atd.

Třetí komponentou je ”SvatServerClient”. Jedná se o komponentu poskytující rozhraní pro práci se serverem. Jejím úkolem je provést prvotní přihlášení k serveru, tím si získá autentizační token, který je poté využíván pro všechny ostatní služby k autentizaci.

Poslední komponentou je nový modul pro hledání podgrafů se jménem ”PatternSearch”. Zmiňuji ho záměrně jako poslední modul v klientské části, protože bude závislý na výše zmiňovaných komponentách. Jeho úkolem bude poskytování uživatelům grafické rozhraní pro vytváření nových kategorií podvodů a následné označování podezřelých podgrafů pro budoucí vyhledávání.

4.2.2 ClueMaker server

První komponenta na serveru, kterou představím, je ”RestApiFilters”. Tato komponenta se stará o příchozí dotazy na server a kontroluje, zda uživatel zaslal společně s dotazem i autentizační token. Pokud ano, tak zkontroluje úroveň oprávnění, zda má uživatel právo provádět tento druh dotazů. V ostatních případech je dotaz zamítnut.

Druhá komponenta se jmenuje ”PatternSearchRest”. Jedná se o novou komponentu, která bude poskytovat restové rozhraní4.1.1 pro klienta. Její služby budou: vytvoření nové kategorie, uložení oanotovaného podgrafu a navrácení výsledků vztahujících se k příslušné kategorii. Jak už bylo zmíněno v odstavci výše i tyto služby nebudou veřejné, ale bude je zastřešovat výše zmíněná komponenta řešící filtraci dotazů.

Další dvě komponenty ”CategoryDao”a ”GraphPatternDao”shrnu v jednom odstavci, protože jsou velice podobné. Obě poskytují abstraktní datové rozhraní servisní vrstvě. To znamená, že nezáleží na tom, v jaké databázi jsou data fyzicky uložena, protože díky těmto komponentám se k nim může libovolně přistupovat jako k Java objektům.

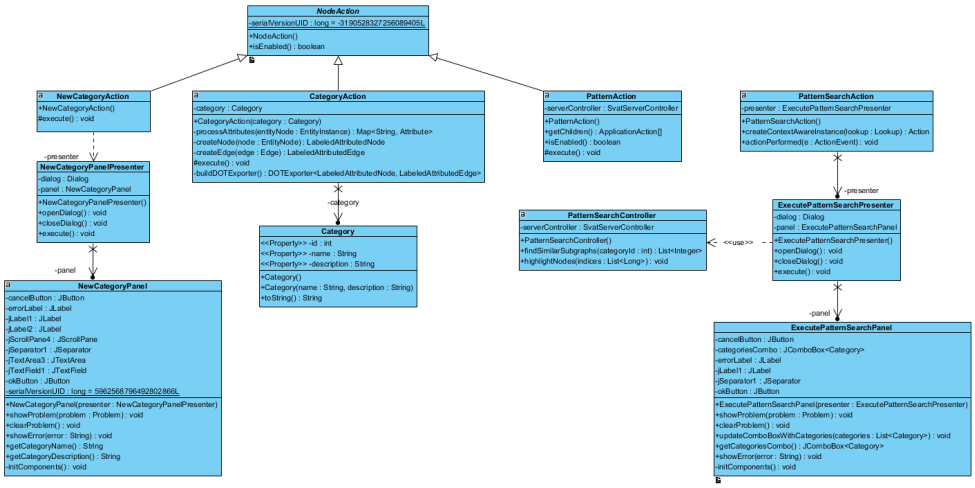
Poslední komponentou je ”PatternSearchService”. Jedná se o komponentu poskytující služby restovému rozhraní. Budou zde prováděny všechny datové operace a agregace výsledků tak, aby už komponenta řešící restové služby dostala zapouzdřená data, která bude propagovat dále.

4.3 Klientská část

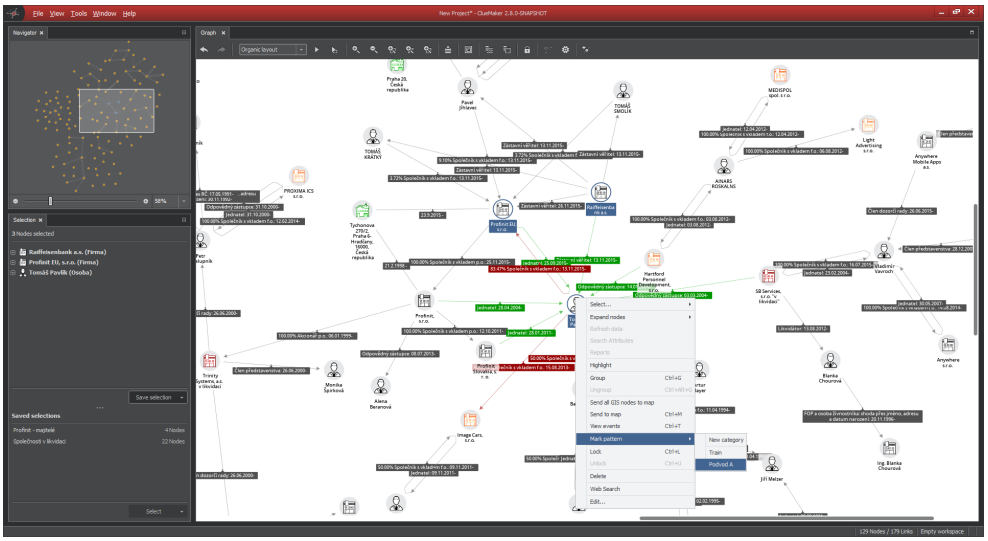
Klientská část je část, kterou vidí uživatel nástroje ClueMaker a pracuje s ní. Na klientské straně aplikace jsem vytvořil čtyři akce, jak lze vidět v diagramu4.4 tříd. Jednotlivé akce budou představeny v následujících podsekcích.

4.3.1 Rozšíření kontextové nabídky

První akce rozšiřuje kontextovou nabídku nástroje, kterou může uživatel vyvolat pravým kliknutím na označený podgraf. Jak lze vidět na obrázku4.5, přidal jsem sekci ”Označit pattern”, která umožňuje uživateli vyvolat zbylé



Obrázek 4.4: ClueMaker - diagram tříd

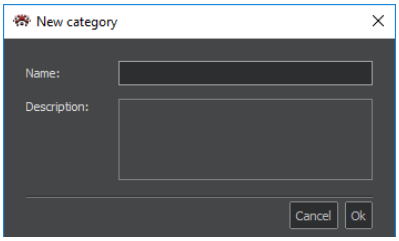


Obrázek 4.5: ClueMaker - kontextová nabídka

dvě akce(vytvoření nové kategorie, uložení kategorizovaného podgrafu). Jednotlivé kategorie jsou dynamicky získávány REST službou ze serveru.

4.3.2 Vytvoření nové kategorie

Jak už název napovídá, jedná se jednoduchou akci, která vytvoří modální okno4.6, do kterého uživatel zadá jméno kategorie podvodu a krátký popisek,



Obrázek 4.6: ClueMaker - vytvoření nové kategorie

který se zobrazuje jako nápověda, po najetí myší na kategorii v kontextovém menu. Všechny tyto informace se po potvrzení formuláře odešlou na server.

4.3.3 Uložení označeného podgrafu

Tato akce slouží k ukládání jednotlivých podgrafů na server. Uložení podgrafu probíhá tak, že uživateli stačí označit pouze konkrétní uzly bez hran, aplikace všechny existující hrany náležící daným uzlům přidá, tak aby byl graf spojitý.

Následuje serializace grafů do JSON formátu. Bohužel zde jsem nemohl použít standardních metod, které serializují objekty na JSON a zpátky, kvůli možným cyklickým závislostem v grafu. Proto jsem musel nejprve graf serializovat do vlastního formátu. Při zkoumání formátů jsem musel brát v potaz, že mám atributovaný multigraf a musím zachovat všechny informace o grafu. Také jsem bral ohled na budoucí přepoužitelnost, tak aby tato funkcionalita nebyla jednoúčelová. Proto jsem se nakonec rozhodl pro otevřený formát DOT.

Tento formát DOT je podporován celou řadou knihoven v různých jazycích od Javy, JavaScriptu až po C++, také ho například podporuje proslulý nástroj k manipulaci a vykreslování grafu Graphviz. Zde jsem přiložil krátkou ukázku syntaxe tohoto popisovacího jazyku.

```
digraph G {
A~[id=1 label="Person" firstname="Jan" surname="Novak"];
B [id=2 label="Company" name="Foo"];
C [id=3 label="Company" name="Bar"];
D [id=4 label="Company" name="FooBar"];

A~-> B [label="FO-PO" param1="123"];
A~-> B [label="FO-PO" param1="456"];
A~-> C [label="FO-PO" param1="7825" param2="45"];
A~-> D [label="FO-PO" param1="27" param2="78"];
B -> C [label="PO-PO" param1="452" param2="12"];
D -- C [label="PO-PO" param1="452" param2="12"];
}
```

Díky tomu, že jsem měl už takto reprezentován graf řetězcem, mohl jsem všechny data odeslat na server.

4.3.4 Spuštění vyhledávání podobných podgrafů

Poslední akci lze vyvolat z hlavního menu aplikace. Vytvoří se modální okno, kde si uživatel může vybrat příslušnou kategorii, na základě které se odešle požadavek na server a dojde tak k vyhledání podobných podgrafů. Ty jsou následně v aplikaci zvýrazněny.

4.4 Serverová část

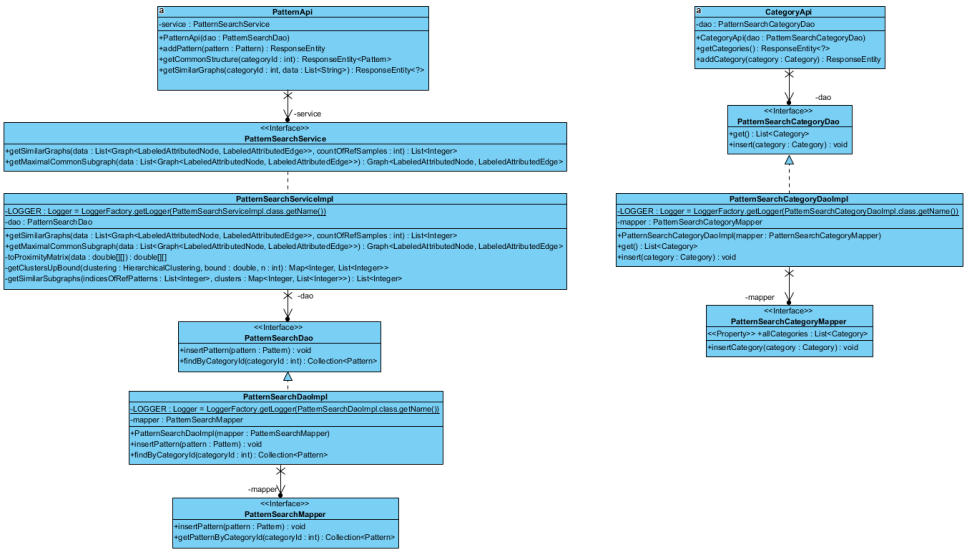
Na straně serveru, jsem musel data obdržená z CluerMakeru uložit a následně je dále zpracovávat. Zde už přichází na řadu již zmiňovaný embdedding, protože na straně serveru už je potřeba s grafy pracovat. Na základě analyzovaných metod a frameworků jsem se rozhodl, že nezvolím ani jednu z nich, protože jsem vždy narazil na nějaké omezení či problém, díky kterému jsem nemohl danou metodu použít.

Nakonec jsem se tedy zvolil metodu statického embeddingu. Ještě chvílemi jsem přemýšlel nad využitím nějaké sofistikovanější metody, například využít autoenkodéry4.1.2 s rekurentní sítí. Ale opět bych potřeboval data ve formátu nějaké sekvence, například rozložena na časové ose. Tento způsob embeddingu bych pravděpodobně mohl využít na porovnávání bílých koní, protože jednotlivé uzly a vazby vznikají a zanikají v čase, tedy nebyl by problém graf reprezentovat jako sekevenci dat. V ClueMakeru mohou být ale libovolná data, která né vždy půjdou takto reprezentovat. Proto opět tuto metodu není možné použít jako "univerzální embedding", protože je příliš závislá na povaze vstupních dat.

Vrátil jsem se zase k metodě statického embdeddingu, pro ten potřebuji, aby každý podgraf měl stejnou strukturu a byl seřazen. Můžu předpokládat, že pokud mám množinu grafů představující konkrétní podvod, tak každý graf bude podobný co se týče struktury a účastníků. Respektive, pokud mám graf reprezentující činnost bílého koně, tak bych mohl očekávat, že v grafu bude figurovat nějaká osoba, firma a vztah mezi nimi.

Proto další činností, kterou server vykonává, je hledání těchto společných rysů ze všech uložených grafů k dané kategorii. Poté musí seřadit jednotlivé entity a převést na vektory čísel, které nakonec spojí do jednoho velkého vektoru reprezentující celý graf.

Posledním úkolem serveru je provést shlukovou analýzu nad daty, které označí uživatel jako podezřelé a zbytkem importovaných dat. Na základě této analýzy vyhodnotí potencionální podvodné podgrafy a ty odešle zpět na klientskou stranu aplikace.



Obrázek 4.7: ClueMaker server - diagram tříd

Jednotlivé kroky, jak jsem postupoval při implementaci všech funkcionalit serveru, jsou rozepsány v následujících podsekcích. Pro lepší představu, přikládám diagram4.7 tříd.

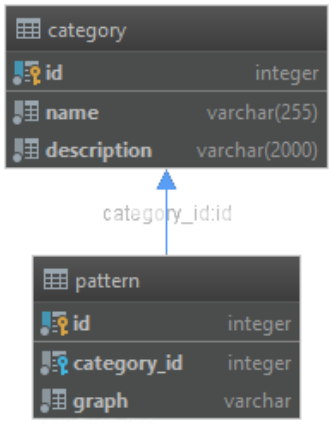
4.4.1 Databázový model

První věc, kterou jsem potřeboval, byla data poslaná z aplikace někam uložit. Zvolil jsem pro tento úkol databázi, na kterou je server napojen viz. komponentová architektura4.2. Aktuální verze serveru podporuje dva typy databázových systémů PostgreSQL a Microsoft SQL, proto je potřeba využít nějaké abstraktní datové vrstvy, kterou v mém případě tvoří framework MyBatis. Konfigurace mapování SQL dotazů a jejich výsledků na objekty, je prováděno pomocí XML souborů.

V této práci potřebujeme evidovat kategorie podvodů a k nim jednotlivé vzorky. Z toho důvodu postačí dvě tabulky viz. schéma4.8.

4.4.2 Vytvoření webových služeb

Databáze už je připravená, proto dalším krokem bude vytvořit webové služby pro nahrávání a získávání dat. K implementaci REST služeb jsem využil framework Spring a vystavil jsem těchto pár základních služeb, které splňují všechny potřeby:



Obrázek 4.8: ClueMaker server - databázové schéma

- **/Category - POST** - tělo této služby obsahuje všechny informace o kategorii(název, popis). Výsledkem je nový záznam v databázi.
- **/Category - GET** - vrací seznam všech kategorií.
- **/Pattern - POST** - dotaz musí obsahovat serializovaný podgraf a cizí klíč kategorie. Výsledkem je nový záznam v databázi.
- **/Pattern/category/*categoryId* - GET** - parametr služby je jednoznačný identifikátor kategorie. Vrací společný podgraf, který slouží jako referenční prvek pro vyhledávání podobných podgrafů ve smyslu struktury.
- **/Pattern/category/*categoryId* - POST** - tělo služby obsahuje vzory stejné struktury, které jsou vyhledány na základě referenčního grafu z předchozí služby. Výsledkem služby je množina identifikátorů uzlů podezřelých grafů, které jsou získány ze shlukové analýzy.

Nakonec jsem na tyto služby napojil nově vytvořený plugin z klientské části.

4.4.3 Hledání společné struktury grafů

Když už všechna data jsou uložená v databázi, nezbývá nic jiného, než implementovat algoritmus hledající největší společný podgraf, který se použije k extrahování společné struktury z každého podgrafu a nakonec se na každý podgraf použije navržený embedding. Než představím implementační detaily, tak blíže popíši problekatiku hledání největšího společného podgrafu.

4.4.3.1 Největší společný podgraf

Jedná se o NP-úplný4.1.3 problém, jako je většina úloh v teorii grafů. Největší společný podgraf[31] má několik definicí, odvíjející se od vlastností grafů. Záleží například, jestli máme graf orientovaný, multigraf a nebo jestli bereme v potaz značky vrcholů a hran. Proto zde zmíním takovou definici, která koresponduje s tímto problémem.

Definice 1

Nechť je graf G definován jako čtveřice $G = (V, E, \alpha, \beta)$, kde

- V je konečná množina vrcholů
- $E \subseteq V \times V$ je množina hran
- $\alpha : V \to L$ je funkce přiřazující značku k vrcholům
- $\beta : V \to L$ je funkce přiřazující značku k hranám

L je konečná neprázdná množina značek. Poté hrana (u, v, l) vede z vrcholu u do vrcholu v a má značku l. Dva vrcholy u a v v grafu G jsou si sousední, pokud hrana (u, v, l) nebo (v, u, l) existuje v G pro alespoň jedno $l \in L$. Máme cestu z vrcholu u do v, pokud existuje sekvence vrcholů začínající v u a končící v v taková, že každý vrchol v sekvenci je sousedem následujícího vrcholu.

Definice 2

Dalším z pojmů, který je potřeba definovat je indukovaný podgraf. Nechť $G = (V, E, \alpha, \beta)$ a $G' = (V', E', \alpha', \beta')$ jsou grafy, G' je indukovaný podgraf G, $G' \subseteq G$, pokud

- $V' \subseteq V$
- $\alpha(v) = \alpha'(v)$ pro každý $v \in V'$
- $E' = E \cap (V' \times V')$
- $\beta(e) = \beta'(e)$ pro každý $e \in E'$

Z této definice vyplývá, že daný graf $G = (V, E, \alpha, \beta)$ a jakákoli podmnožina $V' \subseteq V$ jeho vrcholů, definuje unikátní podgraf a tento podgraf je nazýván podgrafem indukovaným V'.

Tabulka 4.1: Složitosti algoritmu pro hledání největšího společného podgrafu[31]

Algoritmus	Paměťová složitost	Časová složitost
McGregor	$O(N_1)$	$\frac{(N_2+1)!}{(N_2-N_1+1)!}$
Durand-Pasari	$O(N_1 \cdot N_2)$	$\frac{(N_2+1)!}{(N_2-N_1+1)!}$
Balas-Yu	$O(N_1 \cdot N_2)$	$\frac{(N_2+1)!}{(N_2-N_1+1)!}$

Definice 3

Předposlední pojem, který zadefinuji je grafový isomorfismus. Nechť G a G' jsou dva grafy, pak grafový isomorfismus mezi G a G' je bijekce $f : V \rightarrow V'$ taková, že

- $\alpha(v) = \alpha'(f(v))$ pro každý $v \in V$
- pro každou hranu $e = (u, v) \in E$ existuje hrana $e' = (f(u), f(v)) \in E'$ taková, že platí $\beta(e) = \beta'(e')$, a dále pro každou z hran $e' = (u', v')$ existuje hrana $e = (f^{-1}(u), f^{-1}(v)) \in E$ pro kterou platí $\beta(e) = \beta'(e')$.

Pokud $f : V \rightarrow V'$ je grafový isomorfismus mezi grafy G a G', a G' je indukovaný podgraf jiného grafu G'', to je $G' \subseteq G''$, poté funkce f je nazývána "podgrafovým isomorfismem"z G to G''.

Definice 4

S využitím předchozích definic, definuji největší společný podgraf. Nechť tedy $G_1 = (V_1, E_1, \alpha_1, \beta_1)$ a $G_2 = (V_2, E_2, \alpha_2, \beta_2)$ jsou grafy. Pak společný podgraf G_1 a G_2 je $G = (V, E, \alpha, \beta) = cs(G_1, G_2)$ takový, že existuje podgrafový isomorfismus z G do G_1 a z G do G_2. Graf G je nazýván největším společným podgrafem grafů G_1 a G_2, $\mathrm{mcs}(G_1, G_2)$, za předpokladu, že neexistuje žádný společný podgraf G_1 a G_2, který by obsahoval více uzlů jak podgraf G.

Známé algoritmy

Na problém hledání největšího společného podgrafu existuje mnoho algoritmů, protože tato diplomová práce není jenom o této problematice, tak jsem se rozhodl použít srovnaní z této publikace[31].

Autor ve své práci srovnává tyto tři algoritmy McGregor, Durand-Pasari a Balas-Yu. Porovnání provádí nad různými typy grafů a velikostech. Ve většině testů drtivě vítězí algoritmus McGregor, a také jako jediný má paměťovou složitost rovnou velikosti vstupu, jak lze vidět v tabulce složitostí4.1.

Na základě všech těchto informací z výše zmiňované publikace, jsem se rozhodl pro algoritmus McGregor. Samotnou implementaci jsem provedl na

základě publikace[32] autora algoritmu Jamese J. McGregora, kde prezentuje svůj algoritmus a diskutuje nad jeho rozšířením o značky vrcholů a hran.

Proto zde zmíním pseudokód1 mé implementace, která už je obohacena o jednotlivé značky.

Použité symboly v pseudokódu

V_1: množina vrcholů grafu G_1
V_2: množina vrcholů grafu G_2
E_1: množina hran grafu G_1
E_2: množina hran grafu G_2
medges: boolovská matice, kde hodnota *medges[u][v]* je pravdivá, když hrana u v G_1 koresponduje s hranou v v G_2
medgesCopies[i]: pole, kde jsou uloženy kopie *medges*, tyto kopie jsou obnoveny, když algoritmus provádí backtracking
T[i]: je množina vrcholů grafu G_2, které už byly zkoušeny pro vrchol i z G_1
noLabelMatch[i]: boolovské pole příznaků, které odpovídá vrcholům i z G_1, ze začátku je inicializováno na `False` hodnoty

Rozšíření algoritmu na více grafů

Jelikož budu pracovat s více jak dvěma grafy, tak jsem algoritmus rozšířil, aby našel největší společný podgraf z n grafů. Nejdříve se snažím najít největší společný podgraf na nejmenších vzorcích(ty co mají nejméně vrcholů) v dané kategorii. Ten poté iterativně hledám v dalších grafech, pokud mi algoritmus vrátí menší podgraf než mám, tak celý proces opakuji.

Nakonec jsem implementoval ještě ošetření pro případ, kdyby některé grafy byly strukturou úplně disjunktní než většina, tak se tyto grafy přeskočí a nebudou se brát vůbec v potaz.

Výsledkem celého algoritmu je k, stejných podgrafů z hlediska struktury a značek vrcholů a hran, kde $k \leq n$.

4.4.4 Embedding

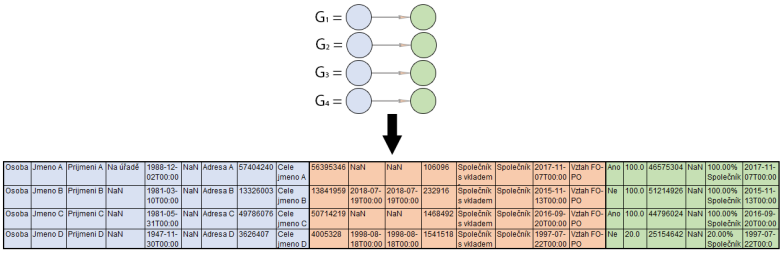
Jsem ve fázi, kdy mám n stejných podgrafů a chtěl bych je reprezentovat jako množinu vektorů, kdy každý vektor bude reprezentovat jeden konkrétní podgraf. Díky vlastnostem McGregor algoritmu, který se vždy chová deterministicky, je zaručeno, že výstupem budou vždy seřazené podgrafy. Pokud si tedy vezmm množinu vrcholů z každého podgrafu, tak vrchol z prvního podgrafu na indexu i, bude vždy odpovídat(z hlediska typu a role v grafu) všem ostatním vrcholům, které se nacházejí na tom samém indexu i.

Následujícím krokem byla extrakce vektorů z atributů entit. Tyto vektory se poté poskladají za sebe do jednoho velkého vektoru reprezentující celý graf. Pro lepší představu jsem vytvořil ilustraci4.9, kde demonstruji tento statický embedding na čtyřech grafech o dvou uzlech a jedné hrany.

Algorithm 1 McGregor(G1, G2)

Všechny níže použité symboly byly vysvětleny v samostatném odstavci4.4.3.1 z velikostních důvodů.

```
1: let a = (v_a, u_a, l_a) and b = (v_b, u_b, l_b), set medges[a][b] to contain l_a = l_b for all a ∈ E_1 and b ∈ E_2
2: i ← 0
3: bestEdgesLeft ← 0
4: T[i] ← ∅
5: while i ≥ 0 do
6:     if |T[i]| < |V_2| then
7:         xi ← getUntriedVertex(i)
8:         T[i] ← T[i] ∪ {xi}
9:         if α_1(i) ≠ α_2(xi) then
10:            noLabelMatch[i] ← True
11:        else
12:            medgesCopies[i] ← medges
13:            refineMedges(i, xi)
14:            edgesLeft ← getEdgesLeft()
15:            if edgesLeft > bestEdgesLeft then
16:                if allPossibleVerticesPaired() then
17:                    if medges.getConnectedEdgesLeft() > bestEdgesLeft then
18:                        bestMedges ← medges
19:                        bestEdgesLeft ← edgesLeft
20:                else
21:                    i ← i + 1
22:                    medgesCopies[i] ← medges
23:                    T[i] ← T[i] ∪ {xi}
24:            else
25:                medges ← medgesCopies[i]
26:    else if noLabelMatch[i] and i ≠ |V_1| − 1 then
27:        noLabelMatch[i] ← False
28:        i ← i + 1
29:        medgesCopies[i] ← medges
30:        T[i] ← T[i] ∪ {xi}
31:    else
32:        i ← i − 1
33:        medges ← medgesCopies[i]
34: return maximal common subgraph
```



Osoba	Jmeno A	Prijmeni A	Na úřadě	1988-12-02T00:00	NaN	Adresa A	57404240	Cele jmeno A	56395346	NaN	NaN	106096	Společník s vkladem	Společník	2017-11-07T00:00	Vztah FO-PO	Ano	100.0	46575304	NaN	100.00% Společník	2017-11-07T00:00
Osoba	Jmeno B	Prijmeni B	NaN	1981-03-10T00:00	NaN	Adresa B	13326003	Cele jmeno B	13841959	2018-07-19T00:00	2018-07-19T00:00	232916	Společník s vkladem	Společník	2015-11-13T00:00	Vztah FO-PO	Ne	100.0	51214926	NaN	100.00% Společník	2015-11-13T00:00
Osoba	Jmeno C	Prijmeni C	NaN	1981-05-31T00:00	NaN	Adresa C	49786076	Cele jmeno C	50714219	NaN	NaN	1468492	Společník s vkladem	Společník	2016-09-20T00:00	Vztah FO-PO	Ano	100.0	44796024	NaN	100.00% Společník	2016-09-20T00:00
Osoba	Jmeno D	Prijmeni D	NaN	1947-11-30T00:00	NaN	Adresa D	3626407	Cele jmeno D	4005328	1998-08-18T00:00	1998-08-18T00:00	1541518	Společník s vkladem	Společník	1997-07-22T00:00	Vztah FO-PO	Ne	20.0	25154642	NaN	20.00% Společník	1997-07-22T00:0

Obrázek 4.9: ClueMaker server - embedding

4.4.5 Předzpracování dat

Vstupem pro předzpracování je výsledná matice po embeddingu. Sloupce matice obsahují atributy jednotlivých entit a co řádek matice, to jeden podgraf. Jelikož atributy jsou uživatelsky definované, tak je potřeba vycházet jenom z hodnot atributů, případě je k dispozici název atributu. Mezi výjimky patří dva atributy, které nesou informaci o vzniku a zániku entit a pro geografické entity jsou předdefinované názvy atributů pro zeměpisnou šířku a výšku. Všechny ostatní atributy jsou neznámé a je potřeba k nim tak přistupovat.

Rozdělil jsem si tedy jednotlivé atributy do čtyř skupin[33], které jsem se pokusil v datech detekovat a poté s nimi tak nakládat.

Číselné atributy - obsahují číselné hodnoty, které popisují měřitelné veličiny jako čísla. Většinou hodnoty těchto atributů odpovídají na otázku "Kolik?". Dále se rozdělují na atributy:

- **Spojité** - jedná se o číslenou proměnnou, která má většinou hodnotu na množině reálných čísel. Hodnota pro spojitou proměnou může zahrnovat hodnoty tak malé, jaké umožňuje měřící přístroj. Příklad spojitých proměnných je například výška, čas, věk a teplota.
- **Diskrétní** - je číselná proměnná, která má hodnotu založenou na počtu celé řady odlišných hodnot. Diskrétní proměnná nemůže mít hodnotu zlomku mezi jednou hodnotou a jí nebližší. Příkladem diskrétní proměnné jsou počet dětí v rodině nebo počet registrovaných aut.

Data, které obsahují číselné atributy jsou kvantitativní data.

Kategorické atributy - obsahují hodnoty popisující "kvalitu"nebo určitou "charakteristiku"dat. Odpovídají na otázky jako "Jakého typu?"nebo "Jaké kategorie". Kategorické atributy spadají do vzájemně se vylučujících(v jedné kategorii nebo v jiné) a do vyčerpávajících(včetně všech možných variant) kategorií. Tento druh atributů je tedy kvalitativní a bývá často reprezentován nečíselnou hodnotou.

Kategorické atributy mohou být členěny na:

- **Ordinální** - je proměnná, jejíž hodnotu lze logicky uspořádat nebo zařadit. Tedy kategorie spojené s ordinálními proměnnými mohou být hodnoceny jako vyšší nebo nižší než jiné, ale nemusí být nutně stanoven nějaký číselný rozdíl mezi každou kategorií. Příklady ordinálních kategorických proměnných zahrnují například akademické ohodnocení(A, B, C, ...) nebo velikost oděvů(S, M, L).

- **Nominální** - je kategorická proměnná, jejíž hodnoty nemůžou být organizovány v logickém sledu. Jakou příklady nominálních kategorických proměnných mohou být barva očí, náboženství nebo pohlaví.

Na následujících řádcích je možné nalézt samotný popis implementace předzpracování dat, který jsem rozdělil do několika kroků.

1. **Odstranění konstant** - jeden z prvních kroků, které jsem podnikl, bylo ostranění sloupců, které jsou konstantní resp. všechny jejich hodnoty jsou stejné.

2. **Odfiltrování irelevantních sloupců** - mnoho sloupců v celém datasetu byly identifikátory interních systémů nebo osob, firem či daňových poplatníků. Jelikož by tyto hodnoty zapadaly do diskrétních číselných atributů, ale pro budoucí analýzu by nebyly vůbec přínosné, spíše naopak škodlivé. Musel jsem tyto atributy nějakým způsobem z dat ostranit. Využil jsem pro tento úkol slovník klíčových slov indetifikátorů a vytvořil regulární výraz, který by tato slova popisoval. Poté pokud název atributu odpovídal regularnímu výrazu, tak jsem ho z dat odstranil.

3. **Zpracování kalendářních dat** - ve všech částech aplikace ClueMaker se používá jednotný formát pro práci s kalendářními daty. Proto nebude těžké detekovat sloupce, které obsahují pouze kalendářní data. Navíc znám jména atributů, které obsahují vznik a zánik entit. Všechny tyto hodnoty jsem převedl na unixový čas[4]. Z dat představující interval, jsem vytvořil nový sloupec s dobou trvání.

4. **Zpracování kategorických hodnot** - další na řadu přišla detekce sloupců obsahující kategorické hodnoty. Ty jsem vybral na základě počtu rozdílných hodnot ve sloupci, tedy pokud počet různých hodnot je menší jako konstanta K, pak jsem prohlásil atribut za kategorický. Následně jsem s těmito sloupci nakládal jako s nominálními, protože bych nikdy z hodnot nemohl detekovat, zda se jedná o ordinální kategorickou proměnnou. Následně jsem všechny tyto sloupce převedl na "One-Hot"kódování4.1.4, které je vhodné pro nominální hodnoty, ale funguje dobře i pro ordinální data.

[4]Jedná se o číslo, vyjadřující počet sekund, které uběhly od 1.1.1970 00:00:00 UTC.

5. **Konverze numerických hodnot** - předposlední krok je najít a konvertovat všechny numerické sloupce do správného datového typu. Všechny ostatní sloupce, které jsem do této doby nepoužil nebo jsem nad nimi neprovedl žádnou transformaci jsem odstranil. Jsou to nepodstatné sloupce obsahující například jméno a příjmení osoby nebo název firmy. Tedy jde o atributy s nulovou informační hodnotou.

6. **Doplnění chybějících hodnot** - posledním krokem je doplnění hodnot, které v datech chybí. Já jsem zvolil pro tento problém univerzální metodu a doplnil jsem každou chybějící hodnotu, průměrem daného atributu.

4.4.5.1 Normalizace

Po vytvoření matice předzpracovaných numerických hodnot, je ještě nutné ji normalizovat. To znamená, že všechny numerické hodnoty musí být převedeny do nějakého intervalu škály.

Já jsem v mé práci zvolil algoritmus MinMax, kde jsem všechny hodnoty škáloval do intervalu $< 0, 1 >$. Tuto transformaci jsem počítal jako:

$$scale = (max - min)/(X.max - X.min)$$
$$X_{scaled} = scale * X + min - X.min * scale$$

kde proměnné min a max jsou spodní resp. horní hranicí intervalu do kterého jsou hodnoty škálovány. $X.max$ resp. $X.min$ je maximální resp. minimální hodnota sloupce, na který se aplikuje transformace a X je konkrétní hodnota určená k normalizaci.

4.4.6 Shluková analýza

Veškerá data už jsou předzpracována a normalizována, tak už nic nebrání k analýze těchto dat. Dovolil bych si ještě připomenout, že v našich datech máme některé řádky označené od uživatele jako již existující podvody a úkolem algoritmu je najít takové prvky, které jsou jim podobné a označit je za podezřelé.

K vyřešení tohoto problémů, jsem se rozhodl použít shlukovou analýzu2.4.6 konkrétně hierarchické shlukování. Tento typ shlukování jsem vybral z důvodu, že u ostatních shlukovacích metod většinou potřebujeme předem znát počet shluků. To v tomto případě není možné z důdvodů, které jsou zmiňovány už v kapitole2.5.3 výše.

Na následujících řádcích popíši, co všechno hierarchické shlukování obnáší, jaké jednotlivé metody a metriky lze použít a co je výstupem celého shlukování.

4.4.6.1 Vstupní data

Vstupem pro hierarchické shlukování je většinou matice vzdáleností, která je navíc ve spodním trojúhelníkovém tvaru, kvůli úspoře paměti. Některé knihovny, jako je například Scipy v jazyce Python, si vystačí přímo s čistými daty a tuto matici vzdáleností si vypočítá na pozadí.

4.4.6.2 Vytvoření matice vzdáleností

K vytvoření matice vzdáleností je potřeba změřit podobnost resp. vzdálenost jednotlivých vektorů. Proto je zapotřebí si zvolit nějakou metriku, podlé které se budou vektorové vzdálenosti počítat. Ty nejznámější[34] na následujících řádcích ve stručnosti popíši:

Vektorové vzdálenosti - předpokladem je, že je měřena podobnost dvou vektorů $X = (x_1, x_2, \ldots, x_n)$ a $Y = (y_1, y_2, \ldots, y_n) \in \mathbb{R}^n$, pak

- **Euklidovská vzdálenost** - jedná se snad o nejznámější a nejpoužívanější metriku pro výpočet vektorové vzdálenosti. Je dána předpisem:

$$D(X, Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- **Manhattanská vzdálenost** - také se jí říká součtová, byla inspirována vzdáleností, kterou je třeba ujít mezi dvěma křižovatkami na Manhattanu, mezi kterými se lze pohybovat jen po na sebe kolmých ulicích ve směru obou os.

$$D(X, Y) = \sum_{i=1}^{n} |x_i - y_i|$$

- **Minkowskiho vzdálenost** - je metrika u které bychom mohli říct, že je zobecněním euklidovské a manhattanské vzdálenosti a je definována předpisem:

$$D(X, Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

  kde $p \geq 1$.

- **Kosinová podobnost** - je míra podobnosti dvou vektorů, která se získá výpočtem kosínu úhlů těchto dvou vektorů.

$$D(X, Y) = \cos(\theta) = \frac{X \cdot Y}{\|X\|\|Y\|} = \frac{\sum_{i=1}^{n} X_i \times Y_i}{\sqrt{\sum_{i=1}^{n} (X_i)^2} \times \sqrt{\sum_{i=1}^{n} (Y_i)^2}}$$

Tabulka 4.2: Linkage metody - notace

Proměnná	Popis
d_{mj}	vzdálenost mezi shluky m a j
m	sloučený shluk, který se skládá ze shluků k a l s $m = (k, i)$
d_{kj}	vzdálenost mezi shluky k a j
d_{lj}	vzdálenost mezi shluky l a j
d_{kl}	vzdálenost mezi shluky k a j
N_k	počet pozorování ve shluku k
N_l	počet pozorování ve shluku l
N_m	počet pozorování ve shluku m
N_j	počet pozorování ve shluku j

- **Korelace** - metrika, která je založena na pearsonovi korelační vzdálenosti a je dána předpisem:

$$D(X,Y) = 1 - \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 \sum_{i=1}^{n}(y_i - \overline{y})^2}}$$

4.4.6.3 Linkage metody

Po výpočtu matice vzdáleností bylo dalším krokem vybrat si mezi linkage (česky spojovacími) metodami. Na následujících řádcích budu používat pouze anglickou verzi tohoto pojmenování, protože český překlad některých slovních spojení by byl akorát matoucí.

Jak již bylo dříve řečeno, u hierarchického shlukování[35] je každý prvek jeden shluk a na ty se rekurzivně používá právě linkage metoda, která z menších shluků dělá větší shluky. Tomuto přístupu se také říká aglomerativní, tedy že se začíná s počtem shluků, který je roven počtu vstupů a končí se s jedním velkým shlukem. Opačný přístup je divizní, ale ten se v praxi moc nepoužívá, proto ho nebudu dále rozvíjet.

Linkage metoda tedy vyjadřuje, které dva shluky je potřeba vybrat pro vytvoření nového shluku. Těchto spojovacích kritérií[36] bylo vytvořeno mnoho, proto ty nejznámější teď představím.

Notace - názvy jednotlivých metod budu opět psát v anglickém jazyce. Význam proměnných, které budu používat lze najít v tabulce4.2.

- **Single linkage** - je rovněž nazývána jako metoda nejblížšího souseda. Vzdálenost mezi dvěma shluky je rovna minimální vzdálenosti mezi pozorováním v jednom shluku a pozorováním v jiném shluku. Když jednotlivá pozorování leží blízko u sebe, jedno spojení má tendenci identifikovat dlouhý řetězec shluků s relativně velkými vzdálenostmi oddělujícími pozorování na obou koncích řetězce.

Vzdálenost se počítá z matice vzdáleností, následujícím předpisem:

$$d_{mj} = \min(d_{kj}, d_{lj})$$

- **Complete linkage** - nazývána taky jako metoda nejvzdálenějšího souseda. Tato metoda může připomínat Single linkage metodu s tím rozdílem, že se zde bere namísto minimální vzdálenosti maximální. Complete linkage se tedy vypočítá následujícím předpisem, opět z matice vzdáleností:

$$d_{mj} = \max(d_{kj}, d_{lj})$$

- **Average linkage** - v metodě average linkage je vzdálenost mezi dvěma shluky vypočítána jako průměrná vzdálenost mezi pozorováním v jednom shluku a pozorováním v jiném shluku. Průměrná vzdálenost je vypočítána následovně:

$$d_{mj} = \frac{N_k d_{kj} + N_l d_{lj}}{N_m}$$

- **Weighted linkage** - jedná se o metodu, kde vzdálenost mezi dvěma shluky je vypočítána jako aritmetický průměr, tato vzdálenost je dána předpisem:

$$d_{mj} = \frac{d_{kj} + d_{lj}}{2}$$

- **Centroid linkage** - vzdálenost mezi dvěma shluky je počítána jako vzdálenost mezi dvěma centroidy resp. středy shluků. Vzdálenost této metody může být vypočítána následujícím vztahem:

$$d_{mj} = \frac{N_k d_{kj} + N_l d_{lj}}{N_m} - \frac{N_k N_l d_{kl}}{N_m^2}$$

- **Median linkage** - jak už název napovídá, vzdálenost u této metody se počítá jako střední vzdálenost mezi pozorováním v jednom shluku a pozorováním v jiném shluku. Lze ji opět vypočítat z matice vzdáleností a předpis je násedující:

$$d_{mj} = \frac{d_{kj} + d_{lj}}{2} - \frac{d_{kl}}{4}$$

- **Ward linkage** - je nejznámější metoda a používá se většinou jako výchozí metoda ve většině knihoven. Zde vzdálenost mezi dvěma shluky je součet čtverců odchylek od bodů k centroidům. Cílem této metody je minimalizovat součet čtverců uvnitř shluku. Předpis pro výpočet je dán vztahem:

$$d_{mj} = \frac{(N_j + N_k)\, d_{kj} + (N_j + N_l)\, d_{lj} - N_j d_{kl}}{N_j + N_m}$$

Tabulka 4.3: Příklad výstupu linkage metody

Index	Index	Vzdálenost
0	3	0.152
1	4	0.322
2	5	0.582
7	6	0.992

Stejně jako u vzdálenostních metrik, volba linkage metody by měla být založena na teoretické úvaze z oblasti domény aplikace. Klíčovou otázkou v tomto ohledu je to, co přesně způsobuje rozdíly. Například v archeologii bychom očekávali, že k různým změnám dojde prostřednictvím inovací a přírodních zdrojů. Tedy například pro detekci, zda jsou dvě skupiny artefaktů podobné, může dávat smysl použití single linkage metody resp. identifikace nejpodobnějších členů shluku. Tam kde neexistují jasné teoretické důvody pro výběr kritérií propojení, je ward linkage metoda rozumným výchozím bodem. Jelikož stanoví, které pozorování se seskupí, na základě snížení součtu čtvercových vzdáleností každého pozorování, z průměrného pozorování ve shluku.

4.4.6.4 Výstup linkage metody

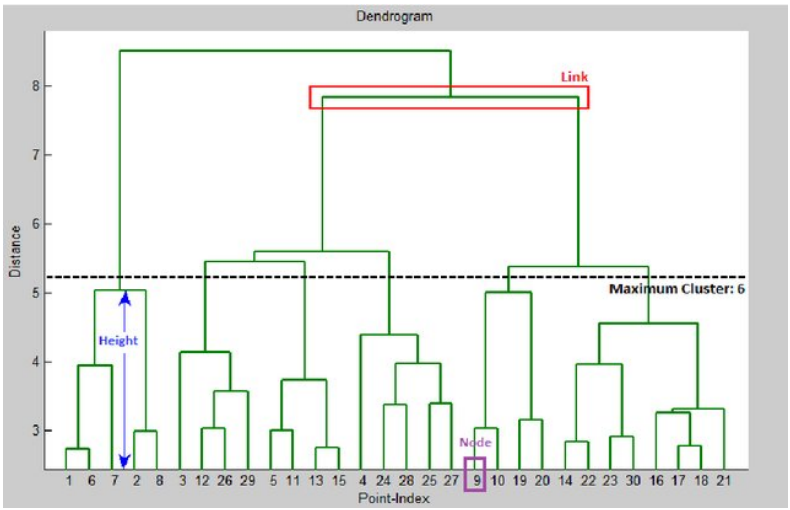
Výstupem linkage metody je seřazené pole dvojic indexů podle podobnosti a příslušné pole vzdáleností stejné velikosti. Pro lepší představu budu demonstrovat výstup na příkladu.

Na vstupu linkage metody je matice vzdáleností o pěti prvcích, pak na výstupu dané metody může být například tato matice4.3. Každý řádek této matice vytváří nový shluk, kterému přiřadí první volný index, tak jak jsou v pořadí za sebou. První řádek matice tedy říká, že se jedná o nejvíce podobné shluky(záleží na výběru metriky linkage metody) o vzdálenosti 0.152. Ve vstupních datech je pět prvků, které mají indexy 0 až 4, poté nově vzniklý shluk bude mít index 5 a bude složen ze shluků 0 a 3.

4.4.6.5 Vizualizace

Hierarchické shlukování lze vizualizovat různými způsoby, jako jsou například grafy, kde se ilustrují navzájem do sebe vnořené shluky. Avšak nejčastější a nejpoužívanější způsob vizualizace je pomocí dendrogramu4.10. Na kterém se může snadno vypozorovat, jaké shluky jsou si navzájem podobné, díky vzdálenosti která je vynesena na ose Y. Na ose X, poté jsou indexy shluků nebo jakoukoli množinu značek, která je k jednotlivým indexům přiřazena.

Dalším velkou výhodou této vizualizace je, že lze na základě nějaké hranice podobnosti(jedná se o řez rovnoběžný s osou X), určit počet shluků nebo naopak.



Obrázek 4.10: Ukázka dendrogramu[37]

4.4.6.6 Klasifikace podezřelých prvků

Nakonec po celém hierarchickém shlukování, jsem určil pomyslnou hranici vzdálenosti, od které vytvořím shluky. Poté jednotlivé shluky procházím a zjišťuji, zda se v ních nenacházejí bílé koně. Pokud ano, tak všechny ostatní prvky ve shluku prohlásím za podezřelé.

Kapitola 5

Evaluace shlukové analýzy

Evaluaci shlukové analýzy jsem prováděl na datech, která jsem si vytvořil následovně. První množinu dat reprezentující bílé koně jsem získal z mediálních kauz, jak jsem již popisoval v kapitole o sběru dat3. Celkem se mi podařilo získat 21 vzorků bílých koní. Druhou množinu představující data, ve kterých budu hledat podobné vzory bílých koní na základě dat z první množiny, jsem získal tak, že jsem použil naagregovaná data z obchodního rejstříku a importoval jsem si do ClueMakeru 200 náhodných firem. Ke všem těmto firmám jsem si nechal dohledat vazby k fyzickým osobám včetně jich samotných.

Měl jsem tedy připravený dataset přibližně o 220 vzorcích. Teď přišla otázka, jak vyhodnotit úspěšnost této shlukové analýzy, když nedokáži říct, zda označený prvek algoritmem je opravdu bílý kůň nebo není. Proto jsem si zvolil dvě metriky, kterými jsem se řídil. První metrika byla počet nově objevených podezřelých podgrafů. Ve druhé metrice jsem z množiny 21 bílých koňů vždy vybral podmnožinu čtrnácti a zkoumal jsem, zda algoritmus dokáže identifikovat zbylé koně. Tento proces jsem opakoval několikrát a mezivýsledky zprůměroval. Dále v textu o této metrice budu mluvit jako o přesnosti.

Výše zmiňované metriky jsem zkoumal v závislosti na výpočtu hranice, od které určuji počet shluků a na zvolené linkage metodě.

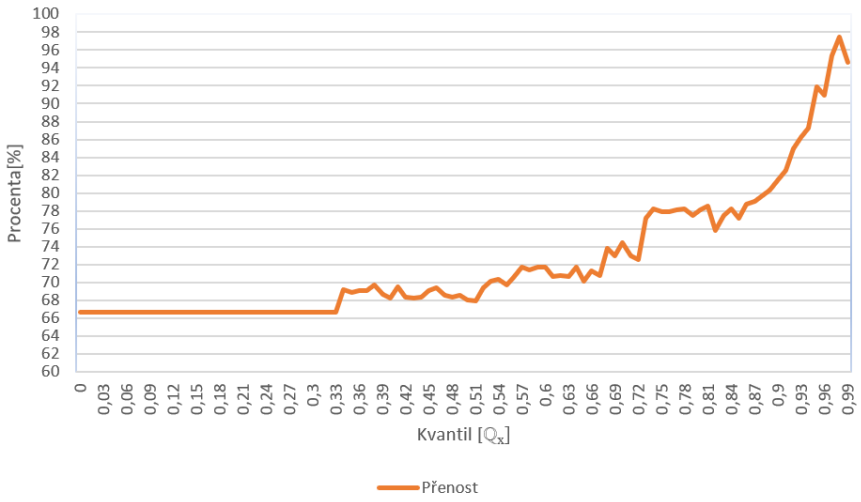
5.1 Volba linkage metody

V tomto testu jsem porovnával všechny linkage metody, které jsem již představil v sekci4.4.6.3. Počet shluků jsem vypočítal z hranice průměrů všech vzdáleností jednotlivých shluků. Tuto hranici jsem určil z empirického zkoumání a vždy se jevila jako rozumná. Jako metriku pro výpočet vzdálenostní matice jsem použil euklidovskou vzdálenost, protože většina linkage metod podporuje právě jenom ji. Výsledné hodnoty jsem poté zanesl do tabulky5.1.

Na základě tohoto srovnání lze vypozorovat, že nejlepších výsledků dosahuje metoda ward linkage, která i z hlediska vlastností v této doméně dává největší smysl.

Tabulka 5.1: Srovnání linkage metod

	Počet podezřelých prvků	Přesnost
Single linkage	5	69.2857
Complete linkage	8	72.5714
Average linkage	6	70.9523
Weighted linkage	6	71.1904
Centroid linkage	6	70.7142
Median linkage	6	70.4761
Ward linkage	8	77.1428

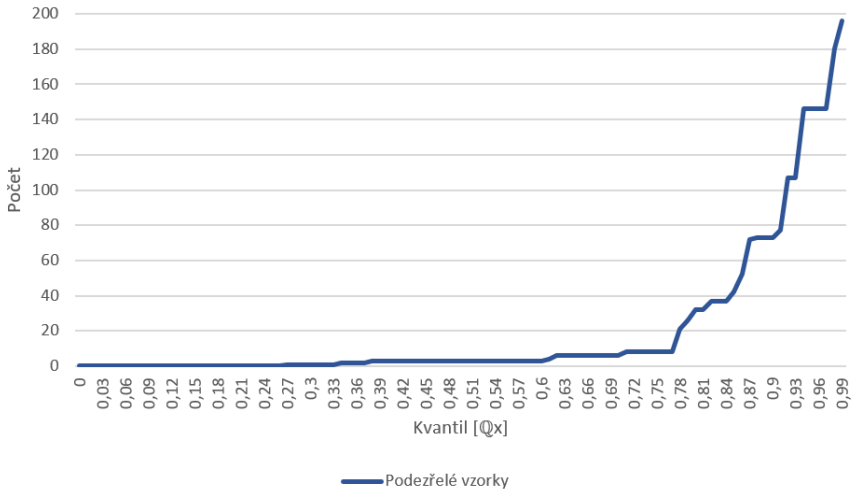


Obrázek 5.1: Závislost přesnosti metody na kvantilu

5.2 Hranice shlukování

Dále jsem se zaměřil na určení hranice pro vytváření shluků. K tomu jsem využil jednotlivé kvantily ze souboru vzdáleností všech shluků. Opět jsem zkoumal jaký vliv má hranice na přesnost a nalezení nově podezřelých prvků. Na tyto dvě metriky jsem vytvořil dva grafy zobrazující závislost dané metriky na kvantilu.

Na prvním grafu5.1 je k vidění přesnost vybrané ward linkage metody. Může se zdát podezřelé, že hranice vždy začíná na 66% přesnosti. To je dáno tím, že vždy nalezneme referenční bílé koně. Je zřejmé, že čím vyšší hranici vzdálenosti sluhků tolerujeme, tím budeme mít méně shluků. S tím přichází větší pravděpodobnost, že bílé koně budou pohromadě a tím budeme dosahovat vyšší přesnosti. Má to ale i negativní stránku, čím budeme mít méně shluků, tím více vzroků budeme označovat jako podezřelé, a to vše kvůli tomu, že shlukujeme shluky s vysokou vzdáleností.



Obrázek 5.2: Závislost nově označených podezřelých prvků na kvantilu

Druhý graf5.2 reprezentuje závislost počtu prvků označených za podezřelé na kvantilu. Lze si všimnout, že asi nejideálnější hranice je přibližně $Q_{0,77}$, poté už je označováno příliš mnoho vzroků s vysokou podobnostní vzdáleností.

5.3 Závěr vyhodnocení

Na závěr bych chtěl říct, že pro tyto problematiku je potřeba najít kompromis mezi těmito metrikami. Pro tento konkrétní případ je ideální vzdálenostní hranice rovna $Q_{0,77}$, ve které dosahují 78% přesnosti a objevil jsem 8 podezřelých vzorků. To ale nemusí být zrovna nejlepší hranice pro ostatní data a jiné domény, proto jsem z empirického zkoumání zjistil, že dobrá aproximace "ideální hranice"je průměr vzdáleností shluků. S hranicí nastavenou na průměr dosahuji výsledků 8 nalezených podezřelých vzorků s 76.5% přesností.

Do budoucna by možná stálo za zamyšlení, zda tuto hranici nenechat na uživateli.

Kapitola 6

Budoucí rozšíření

V poslední kapitole se zaobírám budoucím rozšířením této nově vytvořené funkcionality. Podnětem k napsání této kapitoly bylo to, že jsem během implementace předzpracování dat řešil problém, jak pochopit význam jednotlivých atributů. Aktuální řešení je takové, že setřídím atributy do jednotlivých typů a ty na základě těchto typů transformuji. Toto řešení má však nevýhodu, že nám do výsledného datasetu můžou proklouznost atributy, které s doménou našeho problému nemají nic společného a zbytečně nám zkreslují data.

Po konzultaci s vedoucím této práce jsme se shodli, že tento problém by mohlo vyřešit využití ontologického modelu. Bohužel tento přístup není součástí implemetnace z časových důvodů a rozsahu všech změn, které by bylo potřeba napříč celým nástrojem doimplementovat. Dříve než popíši celou myšlenku, tak nejdříve představím, co to ontologie vůbec je.

6.1 Ontologie

Ontologie[38] je v informatice výslovný a formalizovaný popis určité problematiky. Je to formální a deklarativní reprezentace, která obsahuje definice pojmů a vztahů mezi nimi. Ontologie je také slovníkem, který slouží k uchovávání a předávání znalostí, týkajících se určité problematiky.

6.1.1 Ontologický datový model

Ontologie se používají v umělé inteligenci, softwarovém inženýrství nebo také v sématickém webu jako datový model reprezentující určitou znalost nebo její čas. Datový model ontologie obecně obsahuje čtyři základní typy prvků:

- **Entita** - jedná se o základní stavební prvek datového modelu ontologie. Může být konkrétní (člověk, molekula) nebo abstraktní (událost, pojem).
- **Třída** - je množina entit určitého typu. Podmnožinou třídy je podtřída. Třída může obsahovat zároveň jak entity, tak i podtřídy.

- **Atribut** - popisuje určitou vlastnost či charakteristiku entity.
- **Vazba** - jedná se o jednosměrné nebo obousměrné propojení dvou entit.

6.1.2 Ontologické jazyky

Ontologie mohou být reprezentovány formálními, semiformálními nebo neformálními jazyky. V dnešní době se hlavní vývoj ubírá především v oblasti formálních jazyků, především frameworkem RDF.

6.2 Využití ontologie v ClueMakeru

Myšlenka využití ontologie v ClueMakeru je taková, že už samotný nástroj v sobě bude obsahovat nějaké předem definované ontologické datové modely, popisující různé domény. Poté uživatel vytvářející novou konfiguraci bude mimo mapování SQL dotazů na entity a jejich atributy navíc mapovat tyto entity na ontologický datový model.

Všechny atributy ve vytvořeném ontologickém modelu budou obsahovat informace, jak moc informačně přispívají pro budoucí analýzu a co reprezentují. Na základě těchto informací by bylo snadné určit, zda tento atribut má být rovnou odstraněn nebo naopak z něj mají být extrahovány užitečnější informace(například z kalendářního data věk osoby). Také by se snadněji identifikoval datový typ atributů a ušetřil by se tak čas při předzpracování dat.

Závěr

V této práci jsem postupně přiblížil aplikaci ClueMaker a všechny její funkcionality. Dále jsem shrnul různé druhy přístupů k této problematice, vybral ten nejlepší pro potřeby mnou vytvořené funkcionality a dále ho analyzoval. Následně jsem navrhl řešení a to implementoval jak do klientské strany nástroje resp. aplikace ClueMaker, tak do serverové části resp. ClueMaker serveru. Nakonec jsem celé řešení demonstroval na problematice bílých koní a vyhodnotil jednotlivé metody přístupů.

Výsledkem mé práce je nový fungující modul, který umožňuje uživateli vyhledávat například podezřelé struktury na základě historických dat. Při implementaci jsem dodržel všechny standardní přístupy vývoje softwaru v jazyce Java. Předzpracování dat a extrakci příznaků jsem zobecnil tak, aby algoritmus nebyl omezen na konkrétní doménu dat a fungoval obecně nad libovolnými daty. K hledání podobností mezi referenčními a ostatními strukturami jsem využil hierarchickou shlukovou analýzu, jejíž metody jsem detailně prozkoumal a vybral tu nejvhodnější. Těmito uvedenými kroky se mi podařilo splnit zadání.

Dalším rozšířením tohoto modulu, jak je uvedeno v kapitole6 výše, by mohla být právě zmiňovaná ontologie. Za její pomoci by byl poskytnut lepší popis dat a vedlo by to k vylepšení předzpracování a extrakci dat. Z důvodu příliš obecného zacházení mohou data ve výsledné množině obsahovat irelevantní atributy. Přesně tyto atributy by mohly být na základě ontologického datového modelu eliminovány a model by byl o to přesnější.

Literatura

[1] Profinit EU s.r.o.: ClueMaker logo. [Online], [cit. 27.03.2019]. Dostupné z: http://docs.cluemaker.com/latest/assets/img/logo_home.png

[2] Profinit EU s.r.o.: ClueMaker konfigurace. [Online], [cit. 27.03.2019]. Dostupné z: http://docs.cluemaker.com/latest/assets/img/conf_90.png

[3] Profinit EU s.r.o.: ClueMaker timeline funkcionalita. [Online], [cit. 27.03.2019]. Dostupné z: http://docs.cluemaker.com/latest/assets/img/70_timeline_1_en.png

[4] Profinit EU s.r.o.: ClueMaker funkcionalita GIS. [Online], [cit. 27.03.2019]. Dostupné z: http://docs.cluemaker.com/latest/assets/img/112_gis_window_en.png

[5] Oracle Corporation: Java 8 Overview. 2018. Dostupné z: https://www.oracle.com/technetwork/java/javase/overview/java8-2100321.html

[6] John Kostaras: NetBeans Platform Architecture. 2018. Dostupné z: https://cwiki.apache.org/confluence/display/NETBEANS/NetBeans+Platform+Architecture

[7] Pivotal Software, Inc.s: Spring Boot. 2019. Dostupné z: https://spring.io/projects/spring-boot

[8] Břešťan, R.: Peníze na „analýzy a mediální zastoupení“ Okamurovy SPD šly k lidem spojeným s TV Barrandov a Parlamentními listy. *Hlídací pes - žurnalistika ve veřejném zájmu*, únor 2018, [cit. 2019-04-03]. Dostupné z: https://hlidacipes.org/penize-analyzy-medialni-zastoupeni-okamurovy-spd-sly-k-lidem-spojenym-tv-barrandov-parlamentnimi-listy/

[9] Álvarez García, S.; Freire, B.; Ladra, S.; aj.: Compact and Efficient Representation of General Graph Databases. *Knowledge and Information Systems*, 10 2018, doi:10.1007/s10115-018-1275-x. Dostupné z: https://www.researchgate.net/publication/328079389/figure/fig2/AS:709749297385472@1546228827167/Example-of-a-labeled-directed-attributed-multigraph_W640.jpg

[10] Koehrsen, W.: Neural Network Embeddings Explained. *Towards Data Science*, říjen 2018, [cit. 2019-04-03]. Dostupné z: https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526

[11] Primož Godec: Graph Embeddings - The Summary. *Towards Data Science*, prosinec 2018, [cit. 2019-04-03]. Dostupné z: https://towardsdatascience.com/graph-embeddings-the-summary-cc6075aba007

[12] Li, S. Z.; Jain, A. (editoři): *Embedding Space*. Boston, MA: Springer US, 2009, ISBN 978-0-387-73003-5, s. 259–259, doi:10.1007/978-0-387-73003-5_573. Dostupné z: https://doi.org/10.1007/978-0-387-73003-5_573

[13] Google Inc.: Vector Representations of Words. prosinec 2019, [cit. 2019-04-03]. Dostupné z: https://www.tensorflow.org/tutorials/representation/word2vec

[14] Google Inc.: Vector Representations of Words. [Online], [cit. 27.03.2019]. Dostupné z: https://www.tensorflow.org/images/linear-relationships.png

[15] Zhang, X.-D.: The Laplacian eigenvalues of graphs: A survey. *Linear Algebra Research Advances*, 11 2011.

[16] Gao, X.; Xiao, B.; Tao, D.; aj.: A survey of graph edit distance. *Pattern Analysis and Applications*, ročník 13, č. 1, Feb 2010: s. 113–129, ISSN 1433-755X, doi:10.1007/s10044-008-0141-y. Dostupné z: https://doi.org/10.1007/s10044-008-0141-y

[17] KUČERA, J.: Metody kategorizace dat [online]. 2008 [cit. 2019-04-19]. Dostupné z: https://is.muni.cz/th/w8lgz/

[18] Grover, A.; Leskovec, J.: node2vec: Scalable Feature Learning for Networks. *CoRR*, ročník abs/1607.00653, 2016, 1607.00653. Dostupné z: http://arxiv.org/abs/1607.00653

[19] Narayanan, A.; Chandramohan, M.; Venkatesan, R.; aj.: graph2vec: Learning Distributed Representations of Graphs. *CoRR*, ročník abs/1707.05005, 2017, 1707.05005. Dostupné z: http://arxiv.org/abs/1707.05005

[20] Adhikari, B.; Zhang, Y.; Ramakrishnan, N.; aj.: Sub2Vec: Feature Learning for Subgraphs. In *Advances in Knowledge Discovery and Data Mining*, editace D. Phung; V. S. Tseng; G. I. Webb; B. Ho; M. Ganji; L. Rashidi, Cham: Springer International Publishing, 2018, ISBN 978-3-319-93037-4, s. 170–182.

[21] Liao, L.; He, X.; Zhang, H.; aj.: Attributed Social Network Embedding. *CoRR*, ročník abs/1705.04969, 2017, 1705.04969. Dostupné z: http://arxiv.org/abs/1705.04969

[22] Huang, X.; Li, J.; Hu, X.: Label Informed Attributed Network Embedding. In *ACM International Conference on Web Search and Data Mining*, 2017, s. 731–739.

[23] Li, C.; Wang, S.; Yang, D.; aj.: PPNE: Property Preserving Network Embedding. 03 2017, ISBN 978-3-319-55752-6, s. 163–179, doi:10.1007/978-3-319-55753-3_11.

[24] Bird, S.; Klein, E.; Loper, E.: Natural Language Processing with Python. 2019. Dostupné z: https://www.nltk.org/book/ch07.html

[25] Bird, S.; Klein, E.; Loper, E.: Tree Representation of Chunk Structures. [Online], [cit. 01.04.2019]. Dostupné z: https://www.nltk.org/images/chunk-treerep.png

[26] AI, E.: Industrial-Strength Natural Language Processing. 2019. Dostupné z: https://spacy.io/api

[27] Fielding, R. T.: Representational State Transfer (REST). 2000. Dostupné z: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[28] Bengio, Y.: Learning deep architectures for AI. 2019.

[29] Dasgupta, T.: Deep Autoencoders using Tensorflow. [Online], [cit. 15.04.2019]. Dostupné z: https://cdn-images-1.medium.com/max/880/1*iDcAplzGhttLcYI56MIMxQ.png

[30] Hale, J.: Smarter Ways to Encode Categorical Data for Machine Learning. *Towards Data Science*, říjen 2018, [cit. 2019-04-30]. Dostupné z: https://towardsdatascience.com/smarter-ways-to-encode-categorical-data-for-machine-learning-part-1-of-3-6dca2f71b159

[31] Conte, D.; Foggia, P.; Vento, M.: Challenging Complexity of Maximum Common Subgraph Detection Algorithms: A Performance Analysis of Three Algorithms on a Wide Database of Graphs. *J. Graph Algorithms Appl.*, ročník 11, 01 2007: s. 99–143, doi:10.7155/jgaa.00139.

[32] McGregor, J. J.: Backtrack Search Algorithms and the Maximal Common Subgraph Problem. *Softw., Pract. Exper.*, ročník 12, 1982: s. 23–34.

[33] of Statistics, A. B.: Statistical Language - What are Variables? 2013. Dostupné z: http://www.abs.gov.au/websitedbs/a3121120.nsf/home/statistical+language+-+what+are+variables

[34] Kassambara, A.: Clustering Distance Measures. *DataNovia*, 2018, [cit. 2019-04-30]. Dostupné z: https://www.datanovia.com/en/lessons/clustering-distance-measures/

[35] Bock, T.: What is Hierarchical Clustering? 2019. Dostupné z: https://www.displayr.com/what-is-hierarchical-clustering/

[36] Minitab, L.: Linkage methods for Cluster Observations. 2019. Dostupné z: https://support.minitab.com/en-us/minitab/18/help-and-how-to/modeling-statistics/multivariate/how-to/cluster-observations/methods-and-formulas/linkage-methods/

[37] Alizadeh-Khameneh, M. A.: *Tree Detection and Species Identification using LiDAR Data.* Dizertační práce, 12 2012, doi:10.13140/RG.2.1.3416.2001. Dostupné z: https://www.researchgate.net/profile/M_Amin_Alizadeh-Khameneh/publication/285593710/figure/fig2/AS:302897502933004@1449227800844/The-Dendrogram-and-its-components_W640.jpg

[38] Štencek, J.: Užití sémantických technologií ve značkovacích jazycích. 2009. Dostupné z: http://vse.stencek.com/semanticky-web/

Příloha **A**

Seznam použitých zkratek

SVAT Smart Visual Analytic Tool

SQL Structured Query Language

XML Extensible Markup Language

JSON JavaScript Object Notation

CSV Comma-separated values

PNG Portable Network Graphics

GIS Geographic information system

REST Representational State Transfer

IDE Integrated development environment

NLTK Natural Language Toolkit

UTC Coordinated Universal Time

API Application programming interface

UI User interface

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `cluemaker` ..... zdrojové kódy implementace klientské části
  - `cluemakerServer` ..... zdrojové kódy implementace serverové části
  - `python` ..... zdrojové kódy shlukové analýzy
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS