VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ANALÝZA DAT NA SOCIÁLNÍCH SÍTÍCH S VYUŽITÍM DOLOVÁNÍ DAT

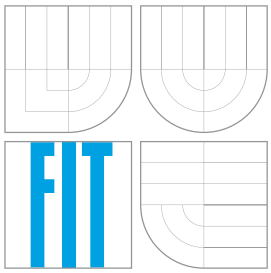
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MAREK FEŠAR

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ANALÝZA DAT NA SOCIÁLNÍCH SÍTÍCH S VYUŽITÍM DOLOVÁNÍ DAT

ANALYSIS OF DATA ON SOCIAL NETWORKS BASED ON DATA MINING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MAREK FEŠAR

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VLADIMÍR BARTÍK, Ph.D.

BRNO 2014

Abstrakt

Práce seznamuje čtenáře s obecnými principy dolování dat a také se specifikami prostředí sociálních sítí. Jsou rozebíráni konkrétní reprezentanti sociálních sítí vybraní s ohledem na jejich návětěvnost a použití českými uživateli. Každé programové rozhraní je diskutováno z hlediska jeho výhod i nevýhod a poté je vybráno vhodné rozhraní pro tvorbu dolovacího nástroje. Práce se dále zabývá analýzou získávání dat ze sítě Twitter, činnostmi v procesu dolování a návrhem dolovacího algoritmu založeného na hustotě, který je užit ve výsledném nástroji. Jeho podrobnému popisu se věnuje kapitola implementace, zvláštní místo se dostává také popisu architektonického vzoru MVC. V závěru práce je ukázáno několik příkladů uplatnění vydolovaných znalostí a nastíněn možný budoucí vývoj.

Abstract

The thesis presents general principles of data mining and it also focuses on specific needs of social networks. Certain social networks, chosen with respect to popularity and availability to Czech users, are discussed from various points of view. The benefits and drawbacks of each are also mentioned. Afterwards, one suitable API is selected for futher analysis. The project explains harvesting data via Twitter API and the process of mining of data from this particular network. Design of a mining algorithm inspired by density based clustering methods is described. The implementation is explained in its own chater, preceded by thorough explanation of MVC architectural pattern. In the end some examples of usage of gathered knowledge are shown as well as possibility of future extensions.

Klíčová slova

dolování dat, sociální síť, programové rozhraní, LinkedIn, Twitter, DBSCAN, MVC, .NET, Entity Framework, relační databáze

Keywords

data mining, social network, API, LinkedIn, Twitter, DBSCAN, MVC, .NET, Entity Framework, relational database

Citace

Marek Fešar: Analýza dat na sociálních sítích s využitím dolování dat, diplomová práce, Brno, FIT VUT v Brně, 2014

Analýza dat na sociálních sítích s využitím dolování dat

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením Ing. Vladimíra Bartíka, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Marek Fešar
3. května 2014

Poděkování

Rád bych vyjádřil poděkování doktoru Bartíkovi za podporu a pomoc při psaní práce.

Obsah

Kapitola 1

Úvod

Sociální sítě dnes obklopují prakticky každého uživatele Internetu a jen málokterý není členem žádné z nich. Neutuchající potřeba řady lidí svěřovat se se svými osobními komentáři na Internetu dává prostor pro relativně nové odvětví dolování z dat – dolování v sociálních sítích. Práce se zaměřuje na několik význačných sociálních sítí a shrnuje možnosti získávání informací prostřednictvím jejich programového rozhraní. Prostor je věnován také samotné problematice dolování z dat a to jak v obecné rovině, tak se zaměřením na problematiku sociálních sítí.

Následně je podrobeno hlubší analýze získávání informací ze sociální sítě Twitter. Rozebírá se konkrétní účel dolování dat a jejich možný užitek pro koncového uživatele dolovacího nástroje.

1.1 Cíl práce

Předmětem diplomové práce je dát čtenáři představu o náročnosti dolování dat ze sociálních sítí a jaké kroky je třeba v tomto procesu uskutečnit. Zaměřuje se na nalézání souvislostí v sociálním grafu, který je charakteristický pro sociální sítě. Pro samotné dolování je pak představen dolovací algoritmus založený na hustotě, který vznikl adaptací shlukovacího algoritmu Density-Based Clustering Method Based on Connected Regions with Sufficiently High Density. Jsou diskutovány jeho nevýhody a také nastíněno jedno z možných vylepšení algoritmu, které je dále zužitkováno v samotné implementaci. Práce vysvětluje charakteristické rysy návrhového vzoru Model-View-Controller, jenž posloužil jako základ pro vzniklou webovou aplikaci na platformě .NET. Po představení několika metrik pro testování je prostor věnován příkladům prakticky vydolovaných znalostí a nastíněno jejich možné využití.

Kapitola 2

Dolování (z) dat

S evolucí informačních technologií souvisí neustálé zvětšování kapacit úložných médií, které s sebou ruku v ruce nese neustálý nárůst objemu skladovaných dat. Jak roste lehkost taková data uskladnit, stoupá obtížnost získat z nich užitečnou hodnotu – informaci. Právě nalézání informací ve velkých datových objemech je předmětem oblasti zvané „data mining“.

V jazyce českém je překlad pojmu data mining poněkud nejednotný, jak se zamýšlí [25]. Ve skutečnosti totiž není předmětem získávání dat, nýbrž získávání informací z dat. Vzhledem k přirozenému sklonu člověka vše zkracovat se však pojem „dolování dat“ uchytil více a proto na něj v této práci pohlížíme jako na ekvivalentní.

2.1 Proč získávat informace

Naléhavá potřeba získávat informace vyvstává do popředí zejména v posledních letech, nemalou zásluhou vlivem sítě Internet, kdy výměna dat je na denním pořádku a opatřit si je v libovolném množství je velmi snadné. Mnoho institucí vnímá hodnotu informací, neboť na nich zakládá svůj byznys. Jmenujme například sektor bankovnictví a pojišťovnictví snažící se předpovědět rizika nesplácení úvěrů klientů či pravděpodobnost pojistné události.

Motivy různých zájmových skupin se velmi liší, cíl je ale obvykle tentýž – získat z nabytých informací prospěch, ať už finančního či jiného charakteru. S rozmachem sociálních sítí a dostupností Internetu se řada uživatelů rozhodla sdílet svůj život s ostatními. Často nevědomky o sobě trousí střípky užitečných informací, které nepokládají za významné (a ony často významné nejsou, pokud je chápeme jednotlivě) a nezvažují možnost, že by je někdo za nimi sbíral a snažil se dovodit obraz celku.

Komerční subjekty prodávající své produkty a služby potřebují získávat neustále další zákazníky a jedním z nabízejících se míst jsou právě sociální sítě. Jejich uživatelé zde zpřístupňují informace o svých zájmech a potřebách a právě na jejich základě se firmy mohou snažit prodávat produkty určitým cílovým skupinám.

Proč se tedy snažit informace získávat by mělo být zřejmé, ostatně každý, kdo si někdy koupil noviny nebo časopis, si je koupil s cílem získat nové informace. Nyní ještě musíme vyřešit otázku, z čeho a jak informace získat či posbírat.

2.2 Zdroje dat

Každý, kdo někdy navrhoval nebo programoval aplikaci, musel řešit otázku ukládání dat. Například do jednoduchého souboru. Již v 60. letech minulého století vznikl naštěstí tak-

zvaný relační model dat, kterému položil základ E. F. Codd [4]. Právě on je dodnes základním kamenem při ukládání dat v řadě oblastí informačních technologií.

2.2.1 Relační databáze

Relační databáze poskytuje zpravidla unifikované rozhraní pro dotazování nad daty uloženými v tabulkách, mezi nimiž jsou zamýšlené (viditelné) a také neviditelné vazby. Právě ty ne na první pohled viditelné vazby mohou často přinést při bližším zkoumání zajímavější informace, ačkoli jejich získání bývá zpravidla o něco složitější.

Relačních databází se po světě vyskytuje celá řada, ne ke všem ale máme volný přístup – o některých nevíme, jiné jsou chráněné a dostupné jen úzké skupině uživatelů. Komplikaci přináší také způsob uložení dat v nich, kdy je logicky preferována pohodlná manipulace s daty (malé a frekventované transakce – Online transaction processing system) před snadností získávání globálního pohledu na data a nových souvislostí (analýza dat – Online analytical processing).

2.2.2 Datová skladiště

Vyústěním nedostatků relačních databází jsou specializovaná datová skladiště. Typicky řeší přesně opačný problém OLTP systémů a tím je onen celkový nadhled nad daty. Datové skladiště má data nejen uspořádána za účelem podrobení širší analýze, ale také může agregovat data z různých zdrojů, například z více filiálek jednoho prodejního řetězce.

Typickým datovým modelem ve skladišti bude datová kostka, znázorňující data v dimenzích s podporou různé granularity (úrovně detailu) v každé dimenzi. Datová skladiště svou podstatou mají blíž k získávání znalostí z dat, nicméně pořád se jedná o analýzu viditelných vztahů.

2.2.3 Ostatní zdroje

S pronikáním informačních technologií do stále více oblastí lidské činnosti vznikají různé specializované aplikace s odlišnými nároky na ukládání dat. Hovoříme například o zaznamenávání prostorových dat (mapové podklady) a objektů (CAD návrhy z oblasti architektury či strojírenství), multimediálních dat (záznamy dopravních kamer, senzorových sítí), historických dat (vývoj kurzu měny nebo akcií), proudových dat (monitorování výroby v automobilce) a dat z prostředí webu (dokumenty různých značkovacích jazyků).

Pro jejich uložení vznikla řada specializovaných databází, jako jsou objektově-relační, čistě objektové, temporální (časové), prostorové, dokumentové a mnohé další vzniklé kombinací více předchozích.

Takováto data bývají kvůli své specifičnosti uložení obtížněji využitelná, o to zajímavější informace v sobě však mohou ukrývat.

2.3 Proces získávání znalostí

Získat znalosti z uvedených zdrojů nelze jednoduchým způsobem. Nepovažujeme za něj prosté dotazování, jako je SQL *select* zpracovaný systémem řízení báze dat. Dokonce ani analytické nástroje pracující s datovými kostkami za ně neoznačujeme. Cílem dolování je dozvědět se netriviálním způsobem nové informace, které v datech na první pohled nejsou patrné – jsou skryté. Očekáváme také potenciální užitečnost získaného, na základě informace

tedy chceme například učinit kvalifikované rozhodnutí. Stejně tak nás nezajímá již jednou objevené a nám známé.

Za dolování se nepovažuje ani dovozování dat z deduktivních databází, ač by se mohlo zdát, že u nich získáváme nové znalosti.

Jak tedy znalosti získat? Uvažované zdroje jsme si již nastínili, nyní se zaměříme na kroky vedoucí k zisku z dat. Schematicky je ilustruje obrázek 2.1.

1. Čištění dat
2. Integrace dat
3. Výběr dat
4. Transformace dat
5. Dolování z dat
6. Vyhodnocení vzorů
7. Prezentace znalostí

2.3.1 Čištění dat

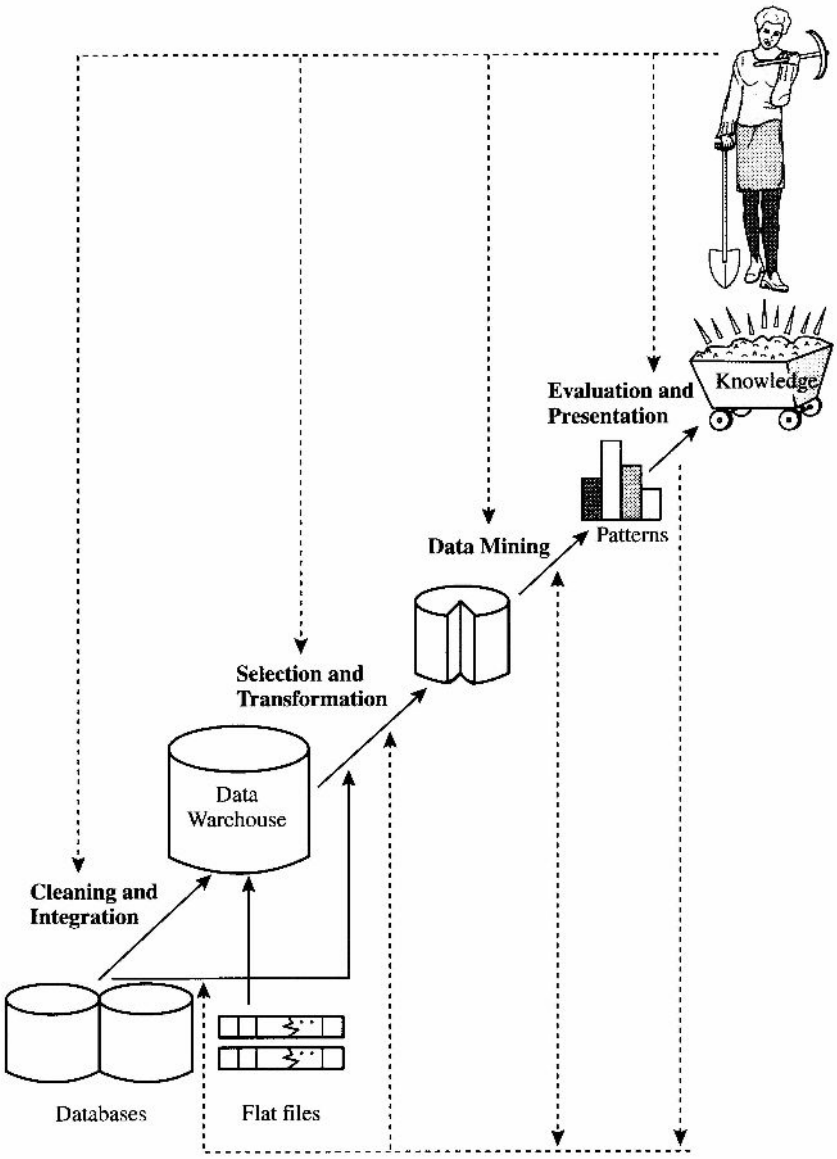
Data přímo ze zdroje zpravidla nejsou v podobě vhodné pro okamžité dolování. V rozsáhlých souborech dat se obvykle vyskytuje větší nebo menší množství chyb, neboť data do databází často procházejí alespoň nějakým krokem lidského zpracování a lidský faktor je zpravidla zároveň zdrojem chyb.

Je zde myšlena například chybějící hodnota, kterou uživatel zapomněl vyplnit, nebo ji vyplnil evidentně nevalidním způsobem (věk 200 let, telefonní číslo s neexistující předvolbou). Data z automatizovaných zdrojů (senzorů) mohou být také různě zašuměná, obsahující odlehlé hodnoty a další nekonzistence. Vlivem sdružení z více zdrojů (databází) dochází někdy k redundanci.

V závislosti na použité dolovací metodě pak takovýto šum může značně negativním způsobem ovlivnit dolovací algoritmus a poskytovat výsledky, které nebudou platné – jejich neplatnost navíc nemusí být vůbec zřejmá.

V zásadě se nejedná o jednoduchou úlohu, neboť na různých atributech je potřeba aplikovat odlišný přístup. Chybějící hodnotu atributu v řádku tabulky můžeme vyřešit vypuštěním celého řádku z dolování, nicméně s ohledem na jiné atributy v daném řádku tím můžeme přijít o část cenných informací. Optimální je nalézt způsob, jak data zachovat a hodnotu nahradit, a zahození celého řádku ponechat jako krajní variantu (více chybějících atributů na stejném řádku).

Manuální nahrazení nepřichází s ohledem na velké množství dat v úvahu, poradit si tedy musíme automatizovaně. Sáhnout lze například po doplnění hodnoty průměrnou hodnotou, námi zvolenou konstantou, nejpravděpodobnější hodnotou či výsledkem jiné statistické funkce aplikované na celý nebo relevantní část souboru dat.



Obrázek 2.1: Schéma získávání znalostí z dat [11]

2.3.2 Integrace dat

Všechna data zpravidla nenajdeme na jednom místě a musíme si tedy poradit s jejich roztříštěností. Může se jednat o zmíněné stažení do centralizované databáze z více filiálek, ale také o integraci provozní databáze z výroby se statickou databází zaměstnanců apod.

Typickým problémem bývá konflikt schématu nebo hodnot, kdy stejná informace je ve sloupcích různého jména, nebo například bydliště je jednou zapsáno jako celek, podruhé strukturováno na úrovni ulice, města a směrovacího čísla. Jednat se také může o různý formát dat nebo užití synonym pro vyjádření téhož (*nezaměstnaný* vs. *bez práce*).

Za zmínku stojí i konflikt identifikace, kdy firma vlastnící řetězec potravin a řetězec drogerií bude mít 2 druhy zákaznických karet a jeden a tentýž člověk bude vystupovat

pod 2 zákaznickými čísly. Řešením je například vygenerovat třetí globální identifikátor.

Častým případem je i redundance dat, kdy určitou informaci lze odvodit z jiných atributů. Typickým příkladem je věk a datum narození, kdy po zákaznících drogerie chceme znát věk a u potravin pro změnu datum narození – po integraci je zbytečné si pamatovat obojí (vhodnější je datum narození, ale co se zákazníky, kteří mají kartu jen v drogerii?).

V praxi nevystačíme jen s jednoduchými, výše popsanými případy, které vyžadují manuální posouzení. Někdy raději sáhneme po korelační analýze, která nám napoví závislost i mezi na první pohled nezřejmými atributy v souboru dat.

2.3.3 Výběr dat

Ne vždy je cílem pracovat se všemi dostupnými daty. Máme li například informace o nákupech za celý rok ze všech prodejen, můžeme chtít jejich množství zredukovat na relevantní soubor neztrácející vypovídací hodnotu o celku. Snahou bývá i snížení množství atributů a redukce dimenzionality dat, aby další zpracování probíhalo rychleji.

Známými postupy je například agregace datové kostky, kdy zmenšujeme granularitu dat na požadovanou úroveň. V anglické literatuře se lze setkat s pojmem „feature selection“ reprezentujícím již zmíněný výběr relevantních atributů. Redukce dimenzionality pak obnáší takový způsob zobrazení (zakódování), kdy je zachována možnost provádět nutné operace nad daty, ale zmenší se jejich dimenze. Poslední dva postupy značně urychlí například shlukování podle dimenzí.

Vybrat z dat lze i určitý vzorek, kdy proces vzorkování by měl zachovat co nejvíce z původních vlastností souboru. Často se lze setkat i s diskretizací, kdy reálná čísla zaokrouhlíme na celé hodnoty, nebo je sdružíme do intervalů (stejné šířky či hloubky).

2.3.4 Transformace dat

Data mohou být *agregována* a *sumarizována*, aby se předpočítané hodnoty užily v následném dolování. Hovoříme také o *vyhlazení*, které snižuje množství šumu v datech, *generalizaci* zmenšující podrobnost obsažené informace (rok narození namísto přesného data), *normalizaci* mapující numerické hodnoty do vymezeného intervalu nebo *konstrukci*, což je postup pro vytvoření zcela nových atributů, které zastupují ty původní a přitom zlepšují proces dolování, ať už z hlediska rychlosti nebo přesnosti výsledků.

Všechny popsané kroky nemusí nutně probíhat v tomto pořadí. Za obvyklé lze považovat i určité slučování kroků do jednoho, kdy data například integrujeme a čistíme zároveň. Výhodou může být, pokud máme pro dolování k dispozici datový sklad. Data v něm typicky prošla všemi již uvedenými kroky a proces dolování tak nemusí být předcházen vším popsanými. S ohledem na velikost dat a to, co reprezentují, se někdy může jednat o náročné operace vyžadující nemalé úsilí.

2.3.5 Dolování z dat

Hlavní část procesu nakládání s daty, jehož cílem je pomocí zvoleného algoritmu získat z dat užitnou hodnotu – informaci. V základu se dolování dělí na 2 hlavní kategorie – *deskriptivní dolování* a *prediktivní dolování*.

Prediktivní dolování vychází ze současných dat s účelem nalézt co nejpřesnější předpověď pro data budoucí. Ukázkou budiž finanční sektor snažící se předpovědět splácení různých skupin klientů nebo vývoj trhu.

Deskriptivní dolování si klade za cíl vystihnout obecné vzory či vlastnosti v analyzovaných datech a zjistit například frekventované množiny. Jejich užití je typické u firem zabývajících se prodejem, kdy se snažíme určit, jaké zboží zákazníci rádi kupují společně a na základě toho stimulovat prodeje vybraného zboží.

Dolovacích algoritmů existuje celá řada, neboť různé cílové skupiny uživatelů chtějí z dat získat jinou formu znalostí a někdy ani není předem jasné, jaké znalosti půjde z uvažovaného souboru dat vytěžit. Některým algoritmům bude později ještě věnována pozornost.

2.3.6 Vyhodnocení vzorů

Identifikace skutečně zajímavých vzorů nalezených v datech. Zpravidla máme definováno určité měřítko zajímavosti nalezeného – není účelné objevovat již známé nebo nevýznamné. Při dolování z dat můžeme narazit na stovky vzorů, z nichž relevantních je někdy jen pár. Nabízí se otázka, co je zajímavé?

Obvykle se zaměřujeme na vzory, které jsou snadno srozumitelné uživatelům, jejichž platnost se podaří ověřit na testovacím vzorku dat s určitou spolehlivostí, jsou potenciálně užitečné a zároveň nové. Někdy také chceme ověřit uvažovanou hypotézu, kterou tušíme ukrytou v datech. Zajímavý vzor pak reprezentuje *znalost*.

Existuje řada objektivních měřítek pro zhodnocení zajímavosti vzoru v závislosti na jeho struktuře. U asociačních pravidel například uvažujeme *podporu* pravidla – množství položek z celku, které splňují ověřované pravidlo. Dále uvažujeme *spolehlivost* charakterizující míru jistoty nalezené asociace. Pro lepší pochopení ilustrujme na příkladu.

Nalezené pravidlo říká, že zákazníci kupující víno kupují často také sýr. Podporou míníme, kolik nákupů obsahuje obě jmenované položky dohromady, tedy $P(\text{víno} \cup \text{sýr})$. Spolehlivost vystihuje, nakolik je jisté, že zákazník mající v košíku víno tam má také sýr, tedy $P(\text{sýr}|\text{víno})$ – jedná se o podmíněnou pravděpodobnost. Jelikož pravděpodobnost se dá vyjádřit číselně, můžeme si nastavit prahovou hodnotu, pod kterou nebudeme pravidla považovat za zajímavá. Tímto eliminujeme nežádoucí výsledky dolování asociačních pravidel.

2.3.7 Prezentace znalostí

Proces získávání znalostí by neměl smysl, pokud bychom neuměli uživateli znázornit, jaké informace byly objeveny. Zde se uplatní techniky vizualizace formou grafů, diagramů, tabulek nebo rozhodovacích stromů. Je to podobné, jako když datová kostka je svého druhu synonymem pro analýzu dat.

Současné nástroje mívají bohatou škálu prezentačních možností, nicméně není předmětem této práce se jim obsáhleji věnovat. Později se seznámíme s vybranou formou prezentace vhodnou pro námi vyšetřovaná data.

2.4 Prostředí sociálních sítí

Prostředí sociálních sítí, kde jsou vztahy mezi entitami (uzly) reprezentovány hranami v grafu, přitahuje v posledních letech nemalou pozornost, úměrně rostoucí s jejich oblíbeností u uživatelů. Analýza sociálních sítí z hlediska dolování z dat se často nazývá *analýza vztahů* nebo také *dolování vztahů*. Následující podkapitola se hlouběji věnuje fenoménu sociálních sítí a jejich možnému využití pro účely dolování.

2.4.1 Sociální sít z pohledu dolování z dat

Sociální síť lze považovat za heterogenní množinu datových objektů s násobnými vazbami. Vhodnou příměrou je zmíněný graf, jehož velikost bývá v praxi značná, což klade určité nároky na práci s ním. Uzly grafu představují objekty sociální sítě, hrany poté uvažované vztahy. Ve složitějších případech mohou být uvažované hrany orientované a jsou s nimi svázány nějaké atributy.

Ač si to na první pohled nemusíme uvědomovat, sociální síť ve skutečnosti nemusí být jen taková, na které uživatelé přímo zamýšleně participují. V různých odvětvích se pracuje s grafy telefonních hovorů, biologickými sociálními sítěmi (mapování nemocí) či sítěmi mapujícími šíření počítačových virů. Každý člověk je tak nevědomky členem řady takovýchto sítí.

Sociální sítě přitahovaly pozornost ještě před začátky svých virtuálních ekvivalentů. Již v roce 1967 uskutečnil harvardský vědec Stanley Milgram se svými kolegy experiment spočívající v doručování dopisů náhodně vybraným adresátům ve městě Boston. Polovinu dopisů se podařilo doručit skrze méně než 5 prostředníků. Byla to jedna z prvních ukázek „malého světa“, který pracuje s teorií, že dva jedinci jsou od sebe vzdáleni přibližně o uvedený počet prostředníků. S rozkvětem počítačových sítí byl uvedený pokus mnohokrát opakován v různých adaptacích.

2.4.2 Charakteristika sociálních sítí

Znát strukturu sociálních sítí bývá užitečné z více důvodů, než nastiňuje předešlá část. Můžeme se například pokusit vytvořit více generací grafů reprezentujících stavy sítě v různých časových okamžicích. Na jejich základě pak předpovídáme, jaké bude množství propojení za den, za měsíc či za rok. Stejně tak se můžeme ptát, s jakou pravděpodobností se propojí vybrané dva uzly či jak bude síť vypadat, až se počet hran zdvojnásobí. Jakými rysy tedy pospat sociální síť? Nejčastěji se zaměřujeme na stupně uzlů, tedy jak silně je objekt sociální sítě zapojen do interakce s ostatními. Může nás také zajímat nejkratší vzdálenost mezi libovolnými 2 uzly nebo jak silně (z hlediska násobnosti vazeb) jsou 2 objekty propojeny skrze prostředníky. Klasickými vlastnostmi zkoumanými v teorii grafů budou průměr sítě, průměrná vzdálenost nebo efektivní vzdálenost (maximální vzdálenost, kterou splňuje určité procento uzlů).

2.4.3 Dolování vztahů

Pro získávání znalostí o vztazích se užívají specializovanější metody, než tradiční strojové učení, beroucí na vstupu náhodný vzorek objektů z jedné relace. Důvodem je to, že sociální sítě pracují s různými druhy objektů (uživatel, příspěvek, fotografie atp.) mající mezi sebou vícenásobné a různě strukturované vazby, často doplněné atributy. Z toho důvodu vzniklo postupem času nové odvětví věnující se tomuto fenoménu zvané *dolování vztahů*. Zahrnuje v sobě metody pro dolování z webu a hypertextových dokumentů, analýzu odkazů, grafů či popisné a prediktivní modelování. Následující část přináší nástin možných předmětů zkoumání včetně příkladů praktického užití.

Klasifikace objektů založená na vztazích

Klasifikace, jak ji známe, se zabývá dělením objektů do kategorií na základě atributů svázaných s klasifikovaným objektem. Zde je však toto posíleno o vazby na jiné objekty, což samo

o sobě přináší určitou informaci navíc, v rozšířené podobě pak uvažujeme i atributy navázaných objektů pro určení výsledné kategorie.

Příkladem může být třídění webových stránek na základě slov, jaká obsahují, a dále na základě odkazů na jiné stránky. Atributem vazby je zde text odkazu, případně jeho titulek, jedná se navíc o vazbu orientovanou. V odborných textech je běžné uvádět bibliografické citace, které také vytvářejí určitou formu grafu.

Predikce typu objektu

Opět si vypůjčíme příklad z oblasti odborných publikací. Na základě atributů a vazeb na jiné objety se snažíme předpovědět, o jaký typ objektu se jedná. Můžeme tak předpovědět, zda jde o příspěvek na konferenci, článek do vědeckého časopisu nebo odbornou knihu. Takto lze automatizovaně třídit i relativně velké soubory publikací.

Predikce typu vazby

Předpovídá účel nebo druh vazby na základě vlastností objektů participujících na vazbě. Můžeme se snažit určit, zda vztah mezi dvěma uživateli sítě je pracovního charakteru, příbuzenského nebo čistě přátelský, či zda odkaz na webové stránce je reklamní nebo navigační.

Predikce existence vazby

Na rozdíl od předešlého se nesnažíme přítomné vazbě přiřadit sémantický význam, nýbrž chceme říci, zda mezi objekty vazba bude nebo nebude. Takto můžeme například předvídat, zda na sebe dvě podobně tematicky zaměřené stránky budou odkazovat, nebo jestli pacient přišel před projevem onemocnění do kontaktu s jiným nakaženým.

Odhad násobnosti vazby

V zásadě můžeme rozlišovat dva druhy odhadů. Kolik vazeb vede k danému objektu, nebo kolik vazeb z něj naopak vychází. První uvedené nám dovolí identifikovat prvotní zdroj informace, kdy ostatní na něj již jen odkazují. Takto se například hodnotí impakt vědeckých časopisů nebo hodnotnost publikace. Druhé pak napoví, že objekt je svého druhu rozcestníkem k dalším objektům. V terminologii webu tak například webový katalog nebude uživateli vrácen jako výsledek vyhledávání, zato bude vyhledávač uvažovat odkazy v katalogu nalezené.

Porovnávání objektů

Snaha zjistit, zda dva objekty jsou si podobné, nebo spíše stejné, má také nemalé uplatnění v sociálních sítích. Můžeme například vyhodnocovat, s jakou pravděpodobností jde o jednoho a tentýž uživatele, který navštívil v různých časech naše stránky – v závislosti na prošlých odkazech, známých atributech (prohlížeč, operační systém) či frekvenci návštěv.

Ve větším měřítku může inzertní server mající reklamy na velkém množství webů (např. Google Ads) identifikovat uživatele a nabízet kontextovou reklamu ne podle aktuálně navštívené stránky, ale podle celkových preferencí uživatele.

Detekce skupin

Jedná se o shlukovací úlohu, kdy na základě atributů a vazeb mezi objekty vyhodnocujeme, zda mají něco společného. Takto můžeme nalézt komunity uživatelů na webu nebo kolekce stránek zaměřené na jednu tematiku.

Detekce podgrafů

Hledání podgrafů majících společné vlastnosti patří mezi další typické úlohy z oblasti grafů. V biologii se užívá pro hledání podgrafů odpovídajících proteinovým strukturám, na sociálních sítích k identifikaci silně provázané skupiny objektů.

Dolování metadat

Chceme-li využít popisná data (metadata) o určitých datech, hovoříme o dolování metadat. Praktickým užitím může být využití strukturovaných hodnot metadat k hledání vazeb mezi nestrukturovanými daty – například multimediálními soubory, které nejsme s to interpretovat.

Kapitola 3

Programová rozhraní sociálních sítí

V kapitole se zaměříme na to, co nám sociální sítě nabízejí z hlediska aplikačního programového rozhraní (API). Podíváme se blíže na vybrané význačné sociální sítě a budeme diskutovat jejich vhodnost pro dolování z dat s ohledem na možnosti poskytované jejich rozhraním. Cílem dolování není vlastní HTML kód sociálních sítí zobrazovaný běžnými prohlížeči, ačkoli si ukážeme, že i v něm může být potenciál někdy větší, než v samotném API. To naproti tomu nabízí programátorsky přívětivější přístup k datům.

3.1 Facebook

Pravděpodobně nejznámější sociální síť současnosti založená v únoru roku 2004. Původní určení bylo pouze propojení studentů Harvardské univerzity, které však brzy vzalo za své a od roku 2006 ji může užívat kdokoli starší 13 let. V dnešní době ji užívá přibližně 1,11 miliardy uživatelů [1], z nichž ne všichni jsou však různými reálnými osobami (ačkoli je to v rozporu s pravidly Facebooku).

Pro zajímavost doplňme, že původní význam slov „face book“ byl adresář se jmény a fotografiemi všech studentů, rozdávaný některými americkými univerzitami svým studentům na začátku akademického roku za účelem snazšího seznámení se s kolegy. Později se nemuselo jednat jen o tištěné, nýbrž i elektronické verze.

3.1.1 Programové rozhraní

Facebook sám se nazývá spíše platformou pro vývojáře, což naznačuje jeho nemalé ambice v oblasti tvorby softwaru třetích stran. Hlavním zdrojem informací obsažených na Facebooku je takzvané *Graph API* [7], což je dotazovací rozhraní založené na protokolu HTTP. S jeho pomocí se lze nejen doptávat na informace, ale také publikovat nové fotografie, statusy, zprávy a jiné. V současnosti se nabízí řada implementací pro různé programovací jazyky, které zapouzdřují celý mechanismus, a není tedy nezbytně nutné vytvářet vlastní HTTP dotazy – s celým API lze pracovat objektově, což je typická volba pro většinu vývojářů.

Alternativou Graph API je *FQL* (Facebook Query Language), což je dotazovací jazyk podobný relačnímu SQL. Dle tvrzení Facebooku je *přibližně* stejně bohatý jako Graph API, nicméně to je preferováno. Odpovědi na dotazy jsou zasílány strukturovaně ve formátu Java Script Object Notation (JSON).

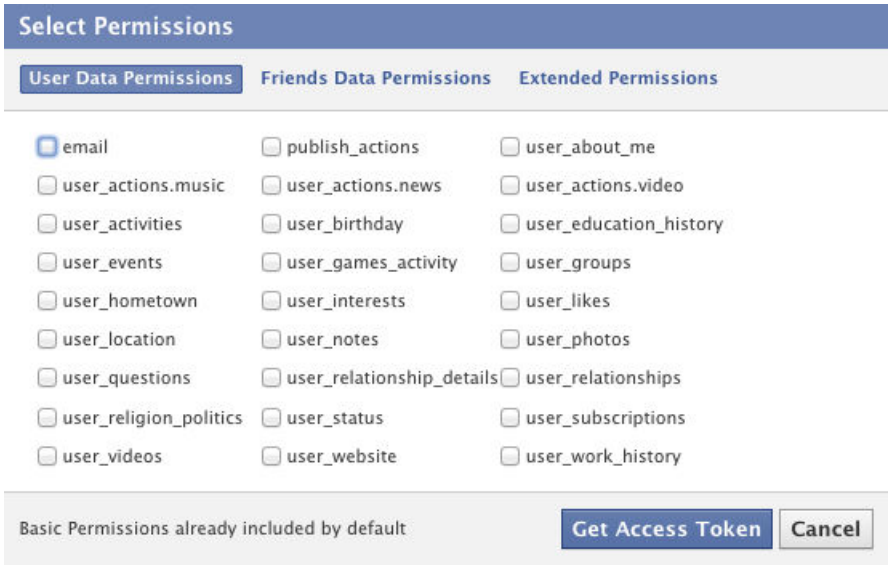
Význačnými uzly sociálního grafu Facebooku jsou *uživatelé* a *stránky*. Z hlediska dolování ze sociálních sítí tak tvoří přirozený cíl snažení. Jedná se o jedny ze základních

kamenů, neboť síť provázejí prakticky od jejího vzniku. Nezůstávejme však jen u těchto. Sociální graf je skutečně bohatý a nabízí celou řadu metadat popisujících vazby na další objekty – adresa bydliště, přátelé, rodina, platby, aktivity, zájmy, hudba, knihy, filmy, televize, hry, příspěvky, „lajky“, poznámky, tagy, statusy, odkazy, alba, události, skupiny, videa, obrázky a další. Je patrné, že některé vazby jsou citlivé povahy (platby, rodina), takže nemusí být dostupné všem.

Facebook skutečně vybízí své uživatele ke sdílení značné části svého života (a soukromí). Uvedené vlastnosti mají naznačit velký potenciál v této síti, zároveň však buďme kritičtí a shrňme si i stinné stránky skutečnosti.

Nevýhody Facebook API

Každá mince má 2 strany a zde si rozebereme tu odvrácenou. Z hlediska tvorby dolovacího nástroje není vůbec snadné se k výše popisovaným datům dostat. Facebook v minulosti čelil určitým problémům se soukromím svých uživatelů, které se dnes snaží chránit lépe než dříve – otázka je však před kým.



Obrázek 3.1: Část oprávnění sociální sítě Facebook [7]

Na obrázku 3.1 lze vidět část z celého výčtu oprávnění užívaných za účelem ochrany soukromí uživatele. Jedná se o seznam čítající desítky položek, v němž je nesnadné se orientovat i po čase programování s jeho aplikačním rozhraním. Obrázek zachycuje tabulku, kterou musí uživatel odsouhlasit předtím, než začne užívat aplikaci pracující s jeho daty. Vyobrazená podoba má ještě další 2 sekce. Nabízí se otázka, nakolik je sám uživatel kompetentní zodpovědět, jaká všechna práva chce aplikaci přidělit a zda tedy neodsouhlasí cokoli. Pro upřesnění doplníme, že programátor musí ve své aplikaci sám nastavit, o jaká práva bude uživatele žádat – možná o všechna, což se zdá nejjednodušší.

Bez přidělených oprávnění nelze přistupovat téměř k žádným datům uživatele (v závislosti na nastavení jeho soukromí). Dá se říci, že pokud je v zájmu uživatele aplikaci užívat, odsouhlasí pravděpodobně cokoli, naproti tomu anonymní dolovací nástroj se přes takovouto překážku přenese hůře. Pokud pomineme možnost, že by dolovací nástroj uží-

val uživatelského účtu někoho s velmi bohatým seznamem přátel, přes který by se dostal k podrobnostem řady uživatelů v jeho kontaktech, nezbývá, než úsilí o zcela samostatnou dolovací aplikaci pracující s API vzdát.

Je paradoxní, že při přístupu na stránku náhodného uživatele bez přihlášení se k Facebooku (například z vyhledávače) se o uživateli zobrazí více informací, než při stejně neautorizovaném dotazování API. V prohlížeči uvidíme například oblíbenou hudbu, filmy nebo skupiny uživatele (respektive zkrácený výčet), ze kterého by se informace daly dolovat. Nicméně Graph API odpoví prázdným seznamem uvedeného. Vyvstává proto myšlenka, zda to uživatele skutečně chrání a zda by API nemělo vracet stejná data jako při přístupu přes webový prohlížeč.

Schůdná řešení v této oblasti se jeví dvě. První je zakomponovat dolovací aplikaci jako součást jiné aplikace, která osloví velký počet uživatelů. Tito pak odsouhlasí obsáhlou tabulku oprávnění, kterou zřejmě nebudou schopni posoudit a programátor dostane do rukou velké množství uživatelských a potenciálně citlivých informací. Příkladem může být katalog módního řetězce jako aplikace pro mobilní telefon, která provázáním s Facebookem nabídne uživateli relativně málo výměnou za množství jeho dat.

Druhou alternativou je nepracovat při dolování se zdrojem dat v podobě Graph API, nýbrž si vytvořit vlastní nástroj pro parsování HTML kódu, který se bude chovat podobně jako webový prohlížeč a přečte tak i jinak programově neviditelná data o uživatelích. Nevýhodou však může být nutnost interpretovat JavaScript, neboť části stránky se načítají dynamicky prostřednictvím AJAX, pravděpodobně (mimo jiné) za účelem ztížení cesty takovýmto nástrojům. Další negativum představuje možnost měnícího se HTML kódu stránky, kdy není nikde zaručeno, jak dlouho bude parser fungovat bez nutnosti jej upravit. Pro zajímavost zmiňme, že i programové rozhraní se čas od času mění a ne vždy to Facebook dopředu ohlásí.

Dá se říci, že rozsáhlý systém oprávnění je výhodou zejména pro Facebook, neboť on sám zobrazuje uživatelům kontextovou reklamu a lze předpokládat, že její obsah je založen na analýze a dolování z dat v uživatelských účtech a jejich vazbách na okolí. Svým způsobem tak omezením jiných posiluje své postavení na trhu s reklamou.

3.2 Twitter

Sociální síť a mikroblogovací služba v jednom – služba umožňující svým uživatelům posílat krátké, 140 znakové zprávy zvané *tweety*. Twitter byl založen v březnu 2006 a o pár měsíců později spuštěn, dá se tedy říci, že je stejně starý jako Facebook (od období veřejného působení). Přesto zaujal jen zhruba poloviční počet uživatelů – přibližně 500 milionů [14]. Z toho aktivních je pouze okolo 200 milionů [24] (únor 2013). Vzhledem k jeho možnostem je to však vysoké číslo. Hlavním principem sítě je psaní krátkých zpráv a odpovědí (*retweetů*) na ně. Oproti Facebooku tedy nepoměrně méně možností.

Tweety však nespojuje jen autor a vazba odpovědí na původní zprávu. Uživatelé mohou do svých zpráv zakomponovat tzv. „hashtag“, což je krátká fráze uvozená značkou „#“. Takto lze odlišit zprávy různých témat nebo typů a ostatní uživatelé je na základě fráze mohou následně vyhledávat a sledovat. Díky tomu vznikají vazby, které lze dále studovat.

3.2.1 Programové rozhraní

Programové rozhraní Twitteru lze označit za na nepodobné Facebooku, neboť i zde se používá v základu HTTP protokol. Můžeme nalézt implementace poskytující vyšší úro-

veň abstrakce pro jednotlivé programovací jazyky, například LinqToTwitter [15] pro platformu .NET. Díky tomu se nemusíme potýkat s implementačními detaily HTTP požadavků a odpovědí a také můžeme pracovat na objektové úrovni, což je dnes jakýsi trend. Odpovědi užívají poměrně rozšířený strukturovaný formát JSON, který lze relativně snadno číst a zpracovávat.

Za jednoznačnou výhodu považujeme možnost vidět přes API vše, co je viditelné běžnému uživateli z prostředí webového prohlížeče. Není zde tedy určitá schizofrenie v systému přístupových práv. Přístupová práva je vlastně zbytečně složité spojení, neboť s v kontextu sítě může číst každý prakticky libovolné tweety bez zvláštních nároků. Výjimku tvoří příspěvky, které uživatelé publikují pouze pro své následovatele (*followery*). Přístup k nim je pak omezen. Je to však jen nepatrné omezení z globálního pohledu.

Vzhledem k menší rozšířenosti Twitteru (v Česku i celosvětově) věnujme větší prostor popisu získatelných informací a co reprezentují. Lze se dotazovat na časovou osu uživatele (*timeline*), což je chronologicky seřazená posloupnost tweetů (nazývaných také *statusy*). K jednotlivým tweetům se můžeme dotázat na odpovědi (*retweety*) a tím rozvíjet určitou stromovou strukturu. Získáním identifikátoru autora odpovědi pak dohledáme jeho další tweety na časové ose a můžeme poměrně snadno mapovat síť provázaných uživatelů.

Uživatelé mohou označit jiné uživatele za své přátele, či se stát následovatelem někoho. Na obojí se lze opět dotázat prostřednictvím API a s tímto dále pracovat. Nechybí možnost označit vybraný tweet jako oblíbený (*favorite*). I přes programové rozhraní můžeme přistoupit k informacím o *navrhovaných uživatelích*. Zde vidíme určitou paralelu k Facebooku, který svým uživatelům také neustále nabízí nové uživatele, které by mohli znát. Avšak u Twitteru může i aplikace třetí strany zobrazovat takto navržené uživatele. Algoritmus navrhování však obě sítě nezveřejňují.

Seznamy jsou dalším pojmem Twitteru. Jedná se o kolekci tweetů vzniklou z časových os všech zúčastněných uživatelů. Seznamy jsou spravovány samotnými uživateli a mohou obsahovat až 500 členů, kteří do nich přispívají. Jedná se o obdobu určitých zájmových skupin uživatelů, méně veřejných než hashtag. Užitím hashtagu se totiž uživatel veřejně přihlašuje k nějaké skupině nebo tématu, aniž by byl členem jakéhokoli uskupení.

Nevýhody Twitter API

Ani Twitter nezůstává stran negativních stránek svého programového rozhraní. Nevýhody však můžeme považovat za menší, než u jiných a lze se s nimi vypořádat. Nejzásadnější nevýhodou pro psaní aplikací je omezení počtu dotazů v čase na Twitter API [21]. Každý dotaz od verze API 1.1 musí pocházet od autentizovaného uživatele nebo aplikace, přičemž na obojí se uplatňuje limit vztažený na časové okno.

Časové okno je v délce 15 minut a v závislosti na typu dotazu je vyhrazeno 15 nebo 180 možných volání v uvedeném okně. Dotazy jsou rozděleny do různých skupin a každá disponuje vlastním počítadlem. Pro vyhledávání tweetů se tak lze dotázat například 180x, kdežto při hledání uživatele nebo retweetů na daný tweet jen 15x.

Za výhodu označme, že Twitter ve svých odpovědích zahrnuje vlastní řádky HTTP hlavičky, které shrnují uplatněný limit, kolik dotazů zbývá a kdy proběhne restartování počítadla. Pokud nepracujeme na takto nízké úrovni (a zapouzdřující knihovna hlavičky nezpřístupňuje), lze se přímo dotázat na zbylé množství dotazů. I tento druh dotazu je však limitován, je tedy vhodné vyvinout v aplikaci jistou logiku samostatně počítající dotazy na API v čase.

Preferovaně se uplatňuje limit ve vztahu k přihlášenému uživateli, pakliže se ale používá

autentizace pomocí aplikace, jdou limity na vrub jí. Aplikace dovolující svému uživateli přispívat na Twitter tedy jednoduše zobrazí hlášení o tom, že je třeba chvíli počkat, naproti tomu dolovací aplikace se bude muset s limitem vypořádat po svém – rozložením dotazů v čase a hlídáním jejich celkového počtu.

Aplikace nehlídající si své dotazy mohou být umístěny na *blacklist* a přístup na Twitter je tím pro ně znemožněn. Nutno dodat, že pro své partnery Twitter nabízí neomezené dotazování [22]. Výměnou za to však aplikace musí splňovat kritéria vyměřená Twitterem, aby se o partnerství a neomezený přístup mohla ucházet. Lze to však považovat za řešitelnou obtíž, která ale pro menší aplikace bude těžko překonatelná.

3.3 LinkedIn

Profesně orientovaná sociální síť se zaměřením na firmy a jejich potenciální pracovní síly. Síť byla založena na konci roku 2002 a spuštěna v květnu roku následujícího. Ačkoli je tak starší než dříve diskutované sociální sítě, pyšní se v současnosti počtem uživatelů přesahujícím 200 milionů [19], čímž je srovnatelná s daleko mladším Twitterem.

Uživatelé na síti sdílejí svou kariérní historii, pracovní zájmy a dovednosti, jedná se o formu životopisu, který lze neustále udržovat aktuální. Není proto asi náhoda, že od roku 2012 již uživatelé nenahrávají na síť své vlastní životopisy – podoba sítě je prakticky zcela nahrazuje.

3.3.1 Programové rozhraní

Rozhraní LinkedIn pro programy sází na HTTP protokol jakožto rozšířený a dostupný prostředek napříč programovacími jazyky. Opět je na programátorovi, zda zvolí vlastní implementaci dotazování, nebo vzhledem k rozšířenosti sítě využije některé z existujících knihoven nabízených přímo na stránkách sítě [13]. Poněkud nezvykle byl zvolen formát odpovědí v jazyce XML, který je pro svou datovou neúspornost používán spíše ojediněle mezi sociálními sítěmi.

Uživatelé na stránkách vytvářejí sociální graf pomocí *propojení* s dalšími uživateli. Jedná se o přímá propojení, kdy se dva uživatelé vzájemně znají. Dále se uvažují nepřímá propojení přes známé osoby (propojení druhého stupně) a dokonce i dále přes tyto nepřímé osoby (propojení třetího stupně). Sociální graf se tak může podstatně rozrůst, což může být velmi významné v profesně orientovaném prostředí, kdy není podstatná blízkost lidí z osobního hlediska, nýbrž jejich kvalifikace.

Bohužel z hlediska API se lze doptávat pouze na propojení prvního stupně, tedy přímá spojení s dalšími lidmi. Navíc k tomuto dotazování je vyžadováno oprávnění přidělené uživatelem, na jehož vazby se dotazujeme. Systém oprávnění je zde však o poznání umírněnější, než v případě sítě Facebook. Jeho rozsáhlost ilustruje tabulka zachycená na obrázku 3.2, jedná se o všechna současná oprávnění. Díky tomu je menší šance využít neznalosti uživatele a získat více oprávnění, než je v jeho zájmu.

I bez přístupových oprávnění můžeme přistupovat k veřejné části profilu libovolného uživatele sítě, z nějž získáme informace o jméně (včetně rodného) a bydlišti, profesní oblasti, specializacích, krátkém resumé jejich profesního života nebo počtu spojení. Lze například vyhledat osoby pomocí vyhledávacího dotazu na API a následně nasbírat veřejné poznatky o nich. Právě vyhledávací dotaz však bude jediné pojítko mezi nimi. Z tohoto hlediska se zdá síť méně vhodná pro dolovací algoritmy zaměřené na sociální sítě, které kladou důraz na vazby uživatelů.

Obrázek 3.2: Oprávnění sociální sítě LinkedIn [12]

Nevýhody LinkedIn API

Do značné míry již bylo nastíněno, že bez přístupových oprávnění k uživatelským informacím nelze pracovat se spojeními uživatele. Podobně jako u Facebooku je zde vidět nekonzistence v ochraně uživatelů, kdy API nevrací bez oprávnění například informace o dosaženém vzdělání nebo působnosti ve firmách, naproti tomu stránka uživatele zobrazená ve webovém prohlížeči toto obsahuje. Dá se říci, že dolování z webu by tak poskytlo lepší možnost zcela anonymně sbírat data a studovat poté vazby osob na vzdělávací instituce, firmy a skupiny s ohledem na jazykové dovednosti, specializaci či bydliště.

Dalším negativem programového rozhraní je omezení počtu dotazů na API za časové okno. Zde se jedná vždy o celý den, počítáno od půlnoci dle UTC času. Limity jsou uplatňovány na aplikaci nebo na uživatele, v závislosti na kontextu dotazování. Dotazy jsou rozděleny do skupin a například uživatelský profil (veřejný či nikoli) lze doptávat 100 tisíckrát za den, vyhledávat spojení uživatele (s patřičným oprávněním) můžeme však jen 20 tisíckrát denně.

Dosud jsem nezmínil stránky společností na síti. Na ně lze přistupovat bez oprávnění, podobně jako na veřejnou část profilu uživatele, avšak kromě nabízených pracovních míst zde opět chybí vazba na okolí a studovat tak můžeme jen výčet jejich vlastností. Šlo by však objevovat skryté vazby mezi firmami – zaznamenávat lokalitu, v níž firma působí, tu překládat prostřednictvím externí služby na zeměpisnou polohu a následně analyzovat, jak se v jednotlivých oblastech daří kterému průmyslu, velkým či malým firmám a kolik nabídek práce inzerují.

3.4 Google+

Sociální síť jedné z předních technologických společností světa se mírně odlišuje od ostatních sítí. Jejím cílem je poskytnout bohatší zážitek uživatelům napříč webem, zejména ve webových službách společnosti Google. Není zde tedy tradiční koncept jednoho místa, kam se uživatelé přicházejí přihlásit a zpravit o novinkách, nýbrž do popředí vyvstává snaha obklopit uživatele. Lze se tak dočíst, že označení sociální síť pro Google+ není správné [20].

Jedná se o velmi mladou síť, jejíž otevření je datováno teprve do roku 2011, přesto disponuje více než 500 miliony uživatelů [10] (leden 2013), z nichž údajně 359 milionů je aktivních [23]. Takto vysoké procento lze přisuzovat rozsáhlosti portfolia služeb Google, kdy možnosti využití účtu jsou širší než u předešlých sítí.

3.4.1 Programové rozhraní

Rozhraní sítě pro programátory využívá rozšířeného HTTP protokolu, což poskytuje stejnou volbu jako ostatní zmiňované sítě – pracovat na úrovni HTTP dotazů, nebo využít některé

z existujících knihoven, jež ho implementují [9]. Server zasílá odpovědi v již dříve zmiňovaném textovém formátu JSON, který je co do objemu metadat docela úsporný. Dotazovat se můžeme na uživatelské profily, které obsahují z větší části veřejné informace a chráněn je především věk či emailová adresa uživatele. Zjistíme tak jméno, datum narození, pohlaví, rodinný stav, ale také v jakých organizacích byl uživatel účasten, stejně jako místa, na kterých žil.

Hlavní vazbu mezi uživateli formují takzvané *aktivity*, což je svého druhu událost vytvořená některým uživatelem, ke které se ostatní mohou přihlásit. Aktivity jsou doprovázeny dalšími atributy, které je popisují. Ke zjištění aktivit daného uživatele je však nezbytné oprávnění přidělené uživatelem. Podobná situace panuje v oblastech zájmu uživatele (*circles*). I zde je potřeba vyžádat si svolení.

Systém oprávnění je vzhledem k rozsahu sítě poměrně stručný a přehledný. Jsou definovány pouze 3 rozsahy práv [8] – *plus.login*, *plus.me* a *userinfo.email*. Ty dovolují identifikovat přihlášeného uživatele ve vztahu k jeho veřejnému profilu, navíc s možností zjištění věku a preferovaného jazyka uživatele, a v posledním případě také zjistit jeho emailovou adresu.

Nevýhody Google+ API

Ačkoli přístup k informacím o konkrétním uživateli je poměrně velkorysý, nalézání vazeb na další uživatele a zájmové skupiny je o poznání komplikovanější. Dá se říci, že anonymní dolování z dat sítě je za použití API téměř nerealizovatelné. Síť tak skutečně plní své poslání obohatit uživatelský zážitek, kdy integrací do aplikace třetí strany se můžeme uživateli přizpůsobit na základě věku, jazyka, polohy, či pohlaví, avšak zjišťovat informace o uživatelích ve velkém měřítku touto cestou nelze.

Podobně jako u konkurence, také Google omezuje nejvyšší možný počet volání programového rozhraní za jednotku času. V základu dostáváme limit 1000 volání na den, což není mnoho. Lze požádat o navýšení této kvóty, ačkoli se jedná o negarantovaný nárok. Dokonce ani oněch 1000 volání není zaručeno a je nutné to chápat jako dobrou vůli této společnosti. Pokud by tak bylo usouzeno, že aplikace se nechová podle představ Googlu, lze ji snížením limitu snadno omezit ve funkčnosti.

API sítě Google+ lze těžko považovat za vhodný datový zdroj pro dolování dat ze sociálních sítí, nicméně si lze představit aplikaci, která bude podle nějakého klíče vyhledávat uživatelské profily a na těchto profilech sbírat veřejná data, která jsou poměrně bohatá (v závislosti na otevřenosti uživatele, nikoli na oprávnění). Jedinou vazbou mezi uživateli by byl vyhledávací dotaz, ze kterého jejich profil vzešel. Na vytvořené databázi uživatelů by šlo uplatnit klasické principy dolování z dat bez specifického zaměření na sociální grafy.

3.5 Zhodnocení programových rozhraní

Rozebrali jsme API několika sociálních sítí, z nichž každá se profiluje mírně odlišně co do nabízené funkčnosti a poskytovaných služeb. Brány v potaz byly zejména nejpopulárnější sítě s ohledem na počet uživatelů s využitím podpůrných ukazatelů jako návštěvnost [5] nebo působnost v prostředí Česka. Existují například sítě Tencent QQ nebo LINE, které jsou také poměrně velké, ale jsou doménou Číny respektive Japonska. Do srovnání nebyla zahrnuta žádná ryze česká sociální síť (Lidé, Spolužáci), neboť tyto nekladou prakticky žádný důraz na programovou přístupnost, a tudíž neposkytují žádné nebo velmi omezené API. Jsou tak vhodné především pro dolování pomocí specializovaného parseru, který by data získal přímo z HTML kódu.

Všechny uvažované sítě sázejí na HTTP metody pro práci s daty, což lze považovat za velké plus v oblasti kompatibility napříč jazyky i platformami. Formáty odpovědí využívající standardu XML nebo JSON jsou nepochybně přínosem pro práci s nimi, neboť pro ně existuje řada knihoven pro čtení a zpracování.

Většina sítí bohužel omezuje množství volání programového rozhraní v čase, což je potřeba zohlednit při návrhu dolovací aplikace. Na druhou stranu je jejich postup pochopitelný a povolený limit se zdá pro dolování většinou použitelný. Komplikace přinášejí užívané systémy oprávnění, neboť realizace aplikace, která by k datům přistupovala anonymně, se někdy stává nemožnou. Anonymitou zde nechápejme zcela skrytou identitu, neboť každá aplikace musí být na sociální síti zaregistrovaná, nýbrž anonymitu ve vztahu k uživateli, který o existenci a činnosti aplikace nemusí mít vůbec ponětí.

Jako nejslibnější se jeví API sítě Twitter, která nabízí velmi dobrý kompromis mezi povoleným množstvím volání za čas, omezeními způsobenými systémem oprávnění a použitelností dotazovaných dat. Většina tweetů je totiž určena široké veřejnosti a můžeme je proto libovolně číst a zpracovávat.

Kapitola 4

Analýza problému a dolovacího nástroje

Předmětem analýzy bude nástroj zaměřený na dolování dat ze sociální sítě Twitter. Dříve uvedený rozbor programových rozhraní sociálních sítí jej staví do role vhodného zdroje dat pro tyto účely. Twitter nabízí možnost dolování ve specifickém sociálním grafu a jeho omezení se nezdají být vážnou překážkou v realizaci aplikace.

Aplikace bude z API získávat informace o příspěvcích (tweetech) uživatelů, přičemž se bude zaměřovat především na autora tweetu a jeho obsah. Konkrétně uvažujme značky, kterými uživatelé obsah svých příspěvků kategorizují – hashtagy. Hashtag je svého druhu vyjádření oblasti zájmu, která uživatele nějakým způsobem oslovuje. Zároveň se nejedná o atribut založený na pevném výčtu hodnot, tudíž se uživatelé nemusí nechat ovlivňovat omezenou nabídkou sítě a naopak mohou vytvářet libovolné takového značky. Jeden tweet má povoleno obsahovat značek libovolné množství, díky čemuž lze hovořit o jistém kontextu, ve kterém se značka vyskytuje (pokud pomineme příspěvky s jedním či žádným hashtagem). Tyto jsou navíc předzpracovány sociální sítí Twitter, proto se přes API dá získat se samotným příspěvkem také kolekce skýtající obsažené hashtagy, ve výsledku tak budou kladeny o něco menší nároky na analýzu celého obsahu tweetu.

Na užitečnost obsažených informací pohlížejme ze dvou úhlů. První klade důraz na hashtag a skrytý kontext, ve kterém se vyskytuje. Díky možnosti vyhledávání příspěvků podle hashtagu a zároveň možnému prohlížení časové osy konkrétního uživatele se dozvíme, jaké značky uživatel využívá a zároveň, jací další uživatelé sympatizují s danou značkou. Navíc, jak se zmiňuje výše, obsahuje hashtag kontext v podobě dalších značek. Tento kontext však nemusíme uvažovat pouze v jeho viditelné, přímé formě, kdy se více značek objevuje v jednom příspěvku. Díky vyhledávání značek z kontextu daného hashtagu se můžeme dozvědět, které další značky z jejich kontextu k němu mají blízko, avšak se s ním nevyskytují společně. Vzniká tak určitá síť, zvětšující se s nejvyšším přípustným počtem prohledávaných kontextů. Můžeme vyhledat značky z kontextu tagu a následně vyhledat značky z kontextu těchto vyhledaných značek. Lze usuzovat, že čím více zprostředkujících kontextů prostudujeme, tím bude nalezená vazba méně relevantní. Studiem 5 kontextů tak nalezneme značky, které mohou s původní mít pramálo společného.

Výsledkem tak bude grafová struktura mapují vazby mezi značkami do zvolené šířky. Důležitým aspektem je také četnost výskytu určitého tagu, a to jak v přímém kontextu, tak tom nepřímém. Z informací o vazbách a četnostech vytvoříme shluky značek, objevující se více či méně společně. Dalším zdrojem dat a potenciálně i informací jsou uživatelé,

kteří jsou autory kontextových hashtagů. Studiem časové osy každého zainteresovaného uživatele se dozvíme o jiném druhu vazby mezi značkami – jaké značky uživatel používá napříč příspěvky, tedy jiný druh kontextu. Je zde předpoklad, že uživatele oslovuje nějaká oblast a k ní má potřebu se opakovaně vyjádřit ve formě tweetů. Užití jednoho tagu ve více příspěvcích značí větší zájem o danou problematiku. Příspěvky označené zcela odlišnými značkami naopak ukazují na jiné oblasti zájmu, které uživatele oslovují.

Vzniká přenesený kontext značky, kdy dva různé příspěvky označené vzájemně disjunktními značkami spojují dvě problematiky, které spolu obsahově nemusejí souviset, ale uživatele obě zajímají. Abychom dokončili myšlenku prvního úhlu pohledu – snažíme se o nalezení hashtagů majících relevantní vztah k zadanému tagu. Vydolované tagy pak mohou sloužit pro účely marketingu. Vše si popišme na příkladu. Uživatel – student píše tweety obsahující tagy *vut* a *škola*. Jiný uživatel zveřejní tweet s tagy *škola* a *mu*. Tagy reprezentující VUT a Masarykovu univerzitu jsou takto spolu spřízněny, ačkoli se nevyskytují společně v jednom příspěvku. Společnost zaměřující se na prodej elektroniky cílící kampaň na studenty tak získá kolekci tagů, které má ve svých příspěvcích užít, neboť je předpoklad, že tyto tagy osloví studenty.

Větší zajímavost však může skýtat nezávislý kontext, kdy zjistíme, že první uživatel psal také příspěvky obsahující *víno*, *bar* nebo *tanec*. Uvedená společnost pak může propagovat nový telefon tweety v duchu „telefon XY vám budou kamarádi v baru závidět“, přičemž jej doplní tagem bar. Uživatel se tak stává osloven nabídkou na elektroniku z oblasti, kde by to původně neočekával. Je totiž předpoklad, že sleduje tweety doplněné hashtagy, které sám užívá. Pro marketingová oddělení je to nástroj na oslovení nových skupin uživatelů prostřednictvím neotřelých tagů. Nemusí pak opakovat neustále stejné „prověřené“ kategorizující značky, vůči kterým se lidé po čase stanou „imunní“.

Druhým úhlem dolování z Twitteru je hledání uživatelů majících podobné zájmy. Vybereme počátečního uživatele, který je nám znám (například současný zákazník) a budeme chtít získat podobně zaměřené uživatele. Začít můžeme tím, že se zaměříme na časovou osu inkriminovaného počátečního člověka. Získáme z ní kolekci tagů, které rád užívá a jejím postupným procházením a vyhledáváním získáme kontext, tentokráte v podobě dalších uživatelů, kterým se daná značka z jeho časové osy také líbí. Tito lidé sami užívají na své časové ose další hashtagy a proto i je můžeme stejným způsobem vyhledávat a nalézat autory dalších tweetů. Otevírá se nám síť uživatelů majících něco společného s původním uživatelem, přičemž těsnost těchto vazeb lze změřit pomocí četnosti společných hashtagů v tweetech.

Nově získané uživatele se pak společnost může snažit přímo oslovit, nebo se zaměřit na hashtagy, které nově objevené uživatele spojují s iniciálním člověkem. Ve druhém případě tak vzniká shluk uživatelů, kteří o sobě nemusejí nic vědět, ale přesto svými vzkazy na sítí netušeně vyjadřují spojení s dalšími lidmi.

Ve svém dalším snažení se zaměříme na první zmiňovaný přístup, tedy dolování shluků hashtagů, které jsou vzájemně propojeny užitím v příspěvcích nebo prostřednictvím uživatelů, kteří je napsali.

4.1 Získávání dat

Implicitním předpokladem je, že dolovací nástroj bude umět dotazovat rozhraní sociální sítě Twitter. Nyní si rozeberme, jak by po technické stránce mohl proces vypadat. Aplikace bude obsahovat robota (často nazývaného *crawler* v kontextu webových vyhledávačů). Ten bude schopen po zadání počátečního hashtagu objevovat přípěvky a uživatele, kteří je napsali.

Zároveň bude sbírat z takto obdržených tweetů nalezené hastagy. Robot si musí sám být schopen ohlídat, zda neprovádí příliš mnoho dotazů na API sociální sítě, aby se nestalo, že bude po čase zablokován.

Jako vhodné opatření, nebo spíše prevence se jeví užívání autentizace uživatelským účtem na Twitteru namísto autentizace aplikací. Jednak to dovolí použití aplikace více uživateli bez toho, aby se vzájemně omezovali nastavenými limity Twitteru, navíc však v případě nedopatření v podobě blokování aplikace (například v případě změny limitů počtu volání API) půjde v dolování pokračovat s jiným účtem (ač nemusí jít nutně o navázání tam, kde předchozí dolování skončilo).

Robot bude získaná data ukládat vhodnou formou. Jako příhodná se jeví některá relační databáze. Navíc si musí umět poznačit stav dolování, ve kterém se nachází a co bude dolovat v budoucnu. Povaha informací o budoucím dolování se mírně odlišuje od informací o již vydolovaném. Minimálně se musí zaznamenat, v jaké vzdálenosti se nový, dosud nevyhledávaný hashtag, nachází od původního zadaného hashtagu. Metrika vzdálenosti musí vhodně zohledňovat, zda se nalezený a dosud neprozkoumaný hashtag vyskytl ve viditelném kontextu iniciálního hashtagu, nebo byl obdržen z časové osy některého uživatele, který použil iniciální hashtag v jiném ze svých příspěvků. V širším kontextu se pak nebude jednat o iniciální hashtag zadaný tím, kdo dolování spustil, nýbrž o hashtag objevený a dosud nevyhledaný, který bude sloužit jako nový počátek hledání. Lze pozorovat jisté zanoření v popsaném prohledávání sítě.

Ze seznamu dosud nehledaných tagů by měly mít přednost ty, které se nacházejí blíže. Považujme to za obdobu algoritmu iterativního prohledávání do hloubky (IDS – iterative deepening search). Zároveň však musíme zabránit, aby se robot ve svém počínání rozjel příliš do šířky a tím snižoval relevanci nalezeného a zároveň prodlužoval proces zisku dat. Nemá například smysl doptávat se na co možná nejstarší příspěvky na časové ose uživatele Twitteru, neboť preference lidí se v čase mění a užití hashtagu před několika měsíci nemusí vypovídat o současném zaměření uživatele. Lze sice uvažovat, že hashtagy mající delší historii použití u dotazovaného člověka jsou mu bližší, avšak zkoumat tento faktor budeme jen v rozumné míře.

Značky užívané častěji (tedy nalezené u více uživatelů) by měly mít větší váhu než ty, které se sice vyskytly ve stejné vzdálenosti, ale jejich frekventovanost je nízká nebo jsou zcela unikátní. Již dříve jsme si řekli, že značka může být zcela libovolná a tak není uživatel ničím svazován. V praxi to pak může vést k tomu, že nalezeme raritní značky, které člověk použil například proto, aby byl odlišný od ostatních, avšak pro nás se jedná o odlehlou hodnotu, nehodnou dalšího zájmu. Robot se tedy odřízne o prohledávání částí grafu, které mají příliš volnou vazbu na zbytek nalezených dat. Je na zváženou, zda se bude jednat o adaptivní hranici, závisející na zkoumané oblasti, nebo půjde o pevně zvolenou absolutní hodnotu, pod kterou bude frekvence výskytu považována za nedostatečnou. Můžeme zvolit kompromis v podobě obojího, kdy adaptivní hranice nebude chápána jako ostrá (prostor se již neprohledá), ale bude například zhoršovat metriku vzdálenosti pro danou značku (prostor se prohledá později, pokud hledání nebude ukončeno).

4.2 Čištění a integrace dat

Nezbytnou součástí procesu získávání znalostí je získaná data vhodně vyčistit. Uvedli jsme, že již samotný robot schraňující pro nás data ze sítě se nebude zabývat značkami, jejichž frekventovanost bude nízká. Po získání souboru dat ale můžeme narazit na pravý opak, kdy vybraní uživatelé mají potřebu značkovat své příspěvky skutečně extenzivně a hodnota

jejich tweetů pro nás klesá. Odstranit takového nedostatky v procesu sběru je obtížnější, neboť chybí globální pohled na data, která ve chvíli sběru ještě nemusí být kompletní. Uvažované řešení může diskriminovat pouze jednotlivé tweety, nebo rovnou vypustit z dolování všechny tweety uživatele, u něhož je nevyhovující příspěvek nalezen. Za vhodnější se jeví selektivní vyřazování tweetů, jako rozšíření si lze představit chování, kdy vyřadíme uživatele za nadlimitní počet nevyhovujících příspěvků (tedy ne za jediný „prohřešek“).

Pokud bude již samotný dolovací robot vynechávat příspěvky mající nula či jeden hashtag, není v této věci potřeba nasbíraná data dále čistit. V opačném případě bychom se i těchto tweetů pro účely dolování zbavili.

Integrace dat nás vzhledem k jednotnosti zdroje nemusí výrazně znepokojovat. Dá se říci, že pro naše účely je v daný okamžik postradatelná. Pokud bychom v budoucnu chtěli aplikaci rozšířit a uvažovat možnost paralelního prohledávání sociální sítě (za pomoci více robotů přihlášených na více uživatelských účtů, tedy s vlastními limity na počet volání API), pak by byla integrace možná na místě. Záleželo by, zda by každý z nich užíval vlastní úložiště a data se sjednocovala dodatečně (tím by každý robot měl jiné vnímání prohledávaného grafu, například frekventovanost hashtagů), nebo by kooperovali nad jednou databází tak, že by každý odebíral ze seznamu dosud neprohledaných hashtagů (jeví se jako příhodnější).

4.3 Selekce a transformace

S ohledem na vlastního robota sbírajícího data můžeme již v procesu sběru značně omezit získávání nadbytečných informací, které by pouze zatěžovaly databázi a nepřidávaly žádnou hodnotu samotnému dolovacímu procesu. Proto se do značné míry obejdeme bez procesu selekce. Robot však patrně bude potřebovat nějaké informace vztahující se k času, kdy byla data obstarána, aby byl schopen navázat na proces dolování a zároveň nepřekročil limit počtu dotazů na API Twitteru. V tomto případě se selekcí omezíme pouze na atributy zahrnující samotné hashtagy a dále na vazební tabulky.

Proces transformace pro nás může být o poznání zajímavější. Vzhledem k většímu předpokládanému množství stejných tagů jednoho uživatele můžeme vytvářet agregované statistiky, které nám určí celkový počet tagů, dále se můžeme zaměřit na četnost kontextových vazeb mezi jednotlivými hashtagy, což je důležitý předpoklad pro vytvoření grafu znázorňujícího vztahy mezi jednotlivými hashtagy.

Grafová struktura se obecně jeví jako příhodná pro následující využití dat v procesu dolování, proto by formát uložených dat měl tomuto záměru co nejvíce odpovídat. Agregované mohou být samozřejmě i další atributy, například počet uživatelů daného tagu, není však cílem poskytnout zde vyčerpávající výčet možností. Naopak je dána přednost agregovat data v závislosti na konkrétních potřebách implementace, neboť operace agregace stojí výpočetní čas a prostor a tudíž by se neměla vyskytovat v množství větším, než bude nezbytné.

4.4 Dolovací proces

Již dříve bylo naznačeno, že nasbíraná data by bylo vhodné reprezentovat formou grafu. Důvodem je skutečnost, že grafová struktura je velice blízká podstatě sociálních sítí, kde uzly představují entity sítě – v našem případě hashtagy a uživatele. Hrany pak reprezentují kontextové vazby mezi jednotlivými entitami. Může existovat více druhů hran, kdy každá odpovídá určité sémantické vazbě – přímý výskyt hashtagů v kontextu tweetu, výskyt hashtagů

na časové ose jednoho uživatele ve více tweetech apod.

Vzniklá grafová struktura tak bude obsahovat různé množství vazeb mezi rozlišnými entitami, které budeme s ohledem na provedenou agregaci uvažovat jako vazby ohodnocené násobností. Z uvedeného grafu je žádoucí dostat informaci o shlucích vrcholů, které jsou vzájemně dostatečně propojeny a zároveň jsou odděleny od okolí vazbami s menší násobností. Jeden z možných přístupů k nalezení oněch shluků je následující.

4.4.1 Dolování hustých podgrafů

Při analýze dolování vzorů z grafů [11] došli výzkumníci k formě grafu zvané *relační graf*, kde každá entita je v grafu zastoupena právě na jednom místě. Kromě sociálních sítí nachází uvedený graf uplatnění v prostředí biologických sítí nebo webových dokumentů.

Princip relačního grafu je prakticky shodný s tím popsaným v úvodu této podkapitoly. Obzvláště zajímavou strukturou v takovýchto grafech jsou pak *vysoce propojené* nebo také *husté* podgrafy nalézající se ve velkých relačních grafech. V sociálních sítích lze takto identifikovat osoby (v našem případě také hashtagy), které mezi sebou vykazují silnou asociaci.

Nevýhodou klasických algoritmů založených na grafech je skutečnost, že se zaměřují zejména na jednotlivé hrany, které mají všechny stejnou váhu a významnost vrcholu je pak posuzována na základě jeho propojení se zbytkem grafu. Pro nás jsou však význačné i ty vrcholy, ke kterým vede například jen jediná hrana, avšak váhově vysoce hodnocená. Z toho důvodu se inspirujeme shlukovacími algoritmy pracujícími s hustotou, která ve výsledku lépe pokryje naše potřeby.

Kapitola 5

Shlukování založené na hustotě

Metody založené na hustotě se vyvinuly s cílem dolovat shluky objektů, které nemají pravidelný tvar a mají různou velikost. Pracují na principu nalézání částí prostoru s vyšší hustotou objektů, které jsou od okolí odděleny prostorem s menší hustotou. Představíme si zde metodu zvanou DBSCAN, kterou se budeme inspirovat při dolování v našem relačním grafu.

5.1 DBSCAN

Celým jménem Density-Based Clustering Method Based on Connected Regions with Sufficiently High Density je shlukovací metoda schopná objevovat shluky zcela libovolného tvaru v prostorových databázích. Metoda si poměrně dobře umí poradit s šumem či odlehlými hodnotami. Shluk je definován jako maximální množina objektů propojených na základě hustoty.

DBSCAN pracuje s několika pojmy, které si hned v úvodu vysvětlíme

ϵ-okolí je oblast do poloměru ϵ od zadaného objektu,

jádro je objekt, v jehož ϵ-okolí se nalézá alespoň zadané množství dalších objektů (dáno hraniční hodnotou *MinPts*),

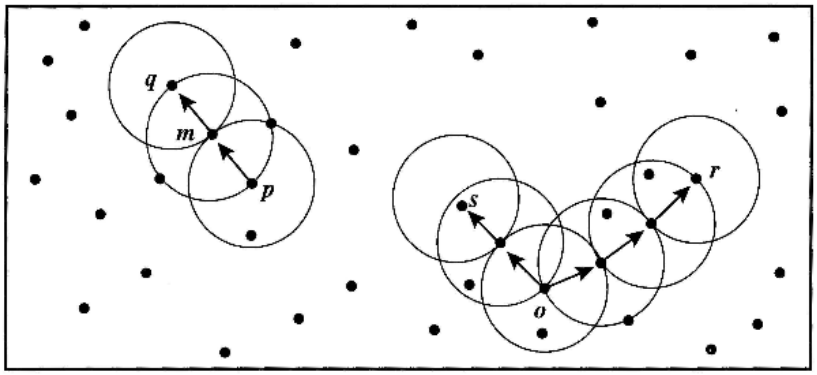
přímá dosažitelnost objekt p je *přímo dosažitelný na základě hustoty* z objektu q, jestliže p se nachází v ϵ-okolí objektu q a zároveň q je jádro,

dosažitelnost objekt p je *dosažitelný na základě hustoty* z objektu q (v množině objektů D) tehdy a jen tehdy, když existuje posloupnost objektů $p_1, ..., p_n$, kde $p_1 = q, p_n = p$ taková, že p_{i+1} je přímo dosažitelný na základě hustoty z p_i pro $\forall i, 1 <= i <= n, p_i \in D$,

propojenost objekt p je *propojený na základě hustoty* s objektem q (v množině objektů D) tehdy a jen tehdy, když existuje objekt $o \in D$ takový, že p je dosažitelné na základě hustoty z o a stejně tak q.

Z definice je patrné, že dosažitelnost na základě hustoty je tranzitivní uzávěr relace přímé dosažitelnosti na základě hustoty a jedná se o asymetrickou relaci. Pouze jádra jsou vzájemně dosažitelná na základě hustoty. Propojenost na základě hustoty je pak symetrickou relací.

Obrázek 5.1 ilustruje popsané pojmy ve 2D prostoru. Objekty jsou reprezentovány body v prostoru, kolem kterých jsou kružnice znázorňující jejich ϵ-okolí. Je uvažována hraniční



Obrázek 5.1: Ilustrace pojmů DBSCAN [11, 6]

hodnota $MinPts = 3$. Na základě dříve psaného jsou objekty označené písmeny m, p, o, r jádry, neboť v jejich ϵ-okolí se nachází alespoň zadané množství dalších objektů.

Objekt m je přímo dosažitelný na základě hustoty z p, stejně tak q je přímo dosažitelný z m. Objekt q je dosažitelný na základě hustoty z p, podobně r či s jsou dosažitelné z o. O objektech r a s navíc můžeme říci, že jsou propojené na základě hustoty.

Shluk objektů na základě hustoty je pak množina objektů propojených na základě hustoty, která je největší možná. Objekty, které nenáleží do žádného shluku, jsou považovány za šum nebo také odlehlé hodnoty.

Proces hledání shluků pak vypadá následovně. Na počátku prochází DBSCAN všechny objekty a hledá jádra. Následně se iterativně vyhledávají objekty, které jsou přímo dosažitelné z některého jádra. Při tom může docházet ke spojování shluků. Algoritmus skončí, když není žádný nový objekt, který by šel přiřadit (byl přímo dosažitelný z jádra) do nějakého shluku.

Výpočetní složitost algoritmu majícího k dispozici prostorový index je $O(n \log n)$, kde n je počet prohledávaných objektů.

5.2 OPTICS

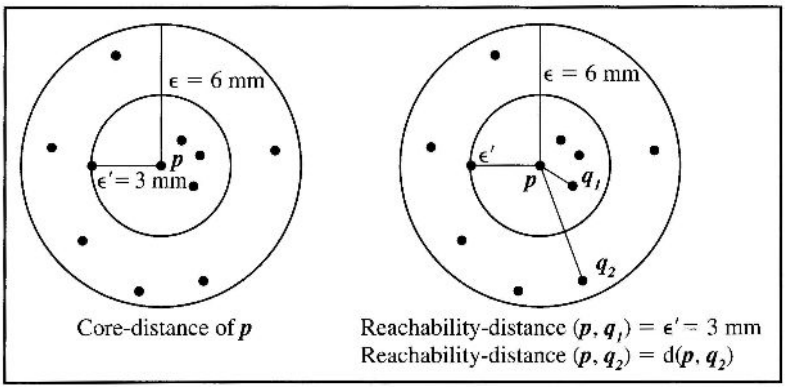
Nevýhodou samotného algoritmu DBSCAN je potřeba vhodně určit hodnoty parametrů $MinPts$ a poloměru ϵ. Různé hodnoty dávají někdy značně odlišné výsledky shlukování, což přináší obtíže spojené s jejich potřebou experimentálního určení pro daný soubor objektů. Nevhodné parametry pak znamenají opakovat dolovací proces od počátku. Dodejme, že uvedený problém se vyskytuje u většiny algoritmů založených na hustotě. Nejedná se však o něco zcela neřešitelného. Algoritmus zvaný Ordering Points to Identify the Clustering Structure přináší v tomto směru značné vylepšení.

OPTICS nesměřuje k jedinému výsledku shlukování, nýbrž se snaží spočítat *pořadí shlukování*. Vychází z předpokladu, že při hledání hustějších shluků objektů (menší poloměr ϵ) pro zadaný parametr $MinPts$ nacházíme zároveň části řidších shluků (větší ϵ). Principem je pak neprovádět výpočet pouze s jednou zadanou hodnotou ϵ, ale rovnou s celou sadou hodnot. Mezi získanými výsledky shlukování je pak dáno pořadí na základě ϵ. Pro vytváření více různých shluků najednou je vhodné procházet zadané objekty v určitém pořadí. Kritériem výběru objektů je velikost parametru ϵ potřebná pro přidání objektu do shluku. Díky tomu jsou dříve objeveny hustější shluky a později dostáváme řidší.

Za účelem takového procházení objektů je potřeba si o nich ukládat dvě nové informace

vzdálenost jádra nejmenší taková hodnota ϵ', kdy objekt p je jádrem. Pakliže p není jádrem pro žádnou z uvažovaných hodnot, je vzdálenost jádra nedefinovaná,

vzdálenost dosažitelnosti pro uvažovaný objekt p ve vztahu k objektu q je větší z hodnot vzdálenosti jádra p a Eukleidovské vzdálenosti mezi p a q.



Obrázek 5.2: Ilustrace pojmů OPTICS [11, 3]

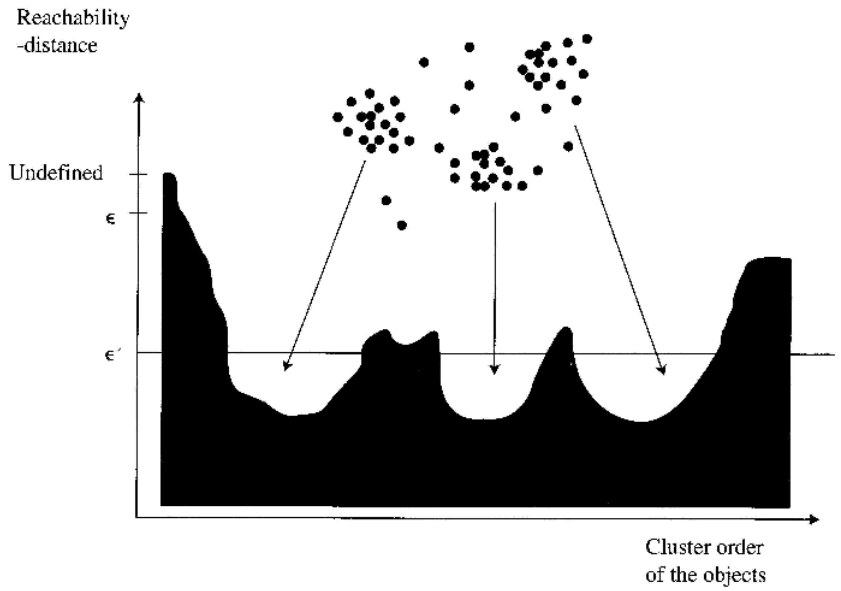
Pro lepší pochopení si vše vysvětleme na obrázku 5.2. Předpokládejme hodnoty parametrů $MinPts = 5$ a $\epsilon = 6mm$. Vzdálenost jádra objektu p je rovna $3mm$, neboť pro ni platí, že ve shluku je alespoň 5 objektů. Vzdálenost dosažitelnosti objektu $q1$ je pak dána jako větší z hodnot vzdálenosti jádra p a Eukleidovské vzdálenosti od p. Důvod je ten, že pro Eukleidovskou vzdálenost menší než vzdálenost jádra by neplatilo, že p je jádrem a tudíž nelze hovořit o přímé dosažitelnosti. Vzdálenost dosažitelnosti pro $(p, q1) = 3mm$. Nyní uvažujme objekt $q2$. Pro něj platí, že se nachází za vzdáleností jádra a proto vzdálenost dosažitelnosti $(p, q2) = d(p, q2)$, kde $d(p, q2)$ značí Eukleidovskou vzdálenost.

Algoritmus OPTICS si uspořádá objekty v databázi a ke každému si poznačí vzdálenost jádra ϵ' a potřebnou vzdálenost dosažitelnosti. Takové informace pak postačují pro generování všech shluků založených na hustotě, které používají ϵ' menší než ϵ použité pro generování pořadí.

Pořadí řazení je znázorněno na obrázku 5.3. Ten ilustruje vzdálenost dosažitelnosti pro dvou rozměrná data. Objekty jsou řazeny v pořadí shluků (mapováno do osy x), kde každému objektu na ose x odpovídá vzdálenost dosažitelnosti na ose y. Obrázek má spíše ilustrativní charakter, konkrétní mapovací funkce pro objekty do osy x není dána. Patrná lokální minima v grafu ukazují jádra shluků. Algoritmus má totožnou složitost s algoritmem DBSCAN, na jehož principech staví.

5.3 Adaptace shlukování na základě hustoty

Algoritmus DBSCAN a jeho varianta OPTICS jsou primárně určeny pro data mapovaná do prostoru. Můžeme však uvažovat jejich přizpůsobení potřebám relačního grafu. Namísto Eukleidovské vzdálenosti budeme pracovat s váhou přiřazenou hraně spojující dva uzly – objekty v algoritmu DBSCAN. Vztah přímé dosažitelnosti na základě hustoty tak bude analogický s původním algoritmem a můžeme proto rozšiřovat shluk kolem jádra. Jádrem



Obrázek 5.3: Řazení objektů algoritmu OPTICS [11, 3]

budeme myslet hashtag, od něhož směřuje alespoň $MinPts$ hran o váze větší než zadaná hodnota ϵ.

Překážkou použití většiny shlukovacích algoritmů je, že pro hashtagy spojené hranami o dané váze neplatí princip triangularity, nelze proto hovořit o metrice. Naše adaptace algoritmu DBSCAN si však s touto skutečností poradí a bude schopná nalézat shluky. Algoritmus OPTICS pak popisuje jeden z principů, díky kterému není třeba přesně zadaná hodnota parametrů pro dolovací algoritmus. Obdobný princip můžeme uvažovat jako případné rozšíření našeho dolovacího algoritmu.

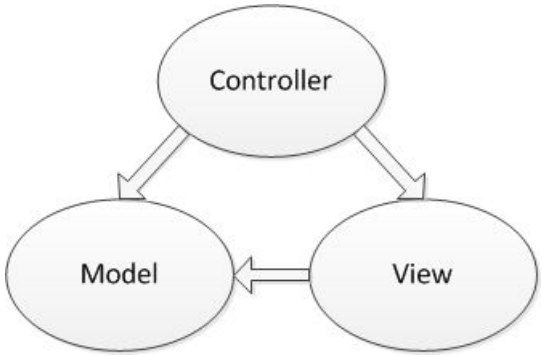
Kapitola 6

Model-View-Controller v prostředí .NET

Aplikace dolovacího nástroje využívá návrhový vzor Model-View-Controller (MVC), jehož princip a architekturu si nyní popíšeme. Prostor je věnován jako obecným vlastnostem, tak konkrétním detailům platformy Microsoft .NET [16].

6.1 Dekompozice aplikace

Návrhový vzor MVC chápe aplikaci jako tři hlavní vzájemně spolupracující části. Z názvu je patrné o jaké se jedná – model aplikace realizující business logiku, view starající se o prezentaci dat a controller stojící mezi uvedenými částmi, zajišťující získávání dat z modelu a jejich předávání pohledům (view) a zpracování akcí uživatele (viz 6.1).



Obrázek 6.1: Základní schéma Model-View-Controller

MVC je jednou z možností vývoje v prostředí Microsoft .NET, framework nabízí také možnost vývoje ve *Web Forms*, kterým se zde nebudeme dále věnovat. Výhodou MVC frameworku poskytovaného přímo Microsoftem je jeho plná integrace do stávající platformy .NET, a tudíž dostupnost všech výhod, jako jsou různé formy autentizace či master pages – rozložení stránek sdílená napříč aplikací ve více pohledech.

6.2 Model

Část aplikace implementovaná za účelem vykonávání logiky spjaté s konkrétní doménou, na kterou aplikace cílí. Typickou činností modelu je zajišťování zisku dat z persistentního úložiště (relační/objektová databáze, souborový systém, webová služba) a jejich vrácení formou vhodných datových typů. Stejně tak zpětné promítnutí změn do úložiště je realizováno zde.

Příkladem může být načtení informací o dolovací úloze z MSSQL databáze a vrácení controlleru, který se získanými daty dále nakládá podle své zodpovědnosti.

Dobře implementovaný model ctí zásadu nezávislosti na zbytku aplikace, tedy controlleru a pohledech. Pro odhalení chyb v návrhu a zamezení jejich promítnutí do výsledné aplikace je výhodné realizovat implementaci modelu jako samostatný projekt v rámci solution (jednotka strukturování aplikace z pohledu vývojového nástroje Microsoft Visual Studio).

Výhodou uvedeného pak je samostatná proveditelná knihovna (*.dll*), která nesmí referencovat žádnou z ostatních částí, a tudíž se v ní programátor nemůže odkazovat na nepatřičné vrstvy aplikace, což by bylo v rozporu s architektonickým vzorem. V menších aplikacích se lze setkat se skutečností, že z důvodu malého rozsahu celého aplikačního modelu je tento integrován se zbytkem. Důsledkem pak mohou být nedokonalosti v podobě nerespektování dobrých zvyklostí typické zejména pro začínající programátory.

6.3 View

Prezentační vrstva aplikace sestává z řady na sobě nezávislých pohledů, které předkládají data koncovému uživateli. Tím nutně nemusí být jen člověk skrytý za webovým prohlížečem, nýbrž data lze obecně prezentovat i ve strukturované podobě XML či v notaci JSON, jež budou sloužit pro další strojové zpracování, například nějaké webové službě.

Klasickými případy pohledů jsou však ty implementované pomocí *aspx* stránek, využívajících kombinace HTML značek a zdrojového kódu psaného v C#. Novější přístup poskytovaný platformou .NET sází potom na tzv. *Razor*. Jedná se v podstatě o doplnění HTML značek dalšími speciálními značkami, které na první pohled nejsou spjaty s klasickými programovacími jazyky. Faktem však zůstává, že i za použití Razor lze pohled opět doplnit příkazy v jazyce C# (případně Visual Basic).

Je především preferencí konkrétního programátora, jaký přístup je mu bližší, neboť co se týká poskytovaných možností, nejsou rozdíly zásadní. S ohledem na to, že Razor je novější, lze ale předpokládat jeho prosazování ze strany Microsoftu, a tudíž i jeho další vývoj.

6.4 Controller

Úloha contolleru již byla nastíněna – slouží jako mezivrstva postavená nad modelem a připravující data pro pohledy. Controller přijme data zaslaná uživatelem, což může být prostý požadavek na zobrazení stránky, ale také odeslání webového formuláře. Podle jejich charakteru s nimi vhodně naloží, například prostřednictvím modelu uloží nového uživatele do databáze apod. Jednou z hlavních zodpovědností je pak výběr správného pohledu, do něhož se budou uživateli data zobrazovat.

Právě výběr pohledu je úzce svázán s tím, jaká data se budou požadovat po modelu tak, aby samotný pohled měl již vše přichystáno a nemusel realizovat žádnou logiku spjatou

s vybíráním dat z databáze. Pohled by tak neměl například potřebovat číst parametry GET a na jejich základě vybírat informace o dolovací úloze z DB, stejně tak není jeho úlohou ověřovat přihlášení uživatele. To vše leží na zodpovědnosti controlleru.

6.5 Výhody MVC

MVC do značné míry pomáhá vytvářet aplikace, které vhodně oddělují různé aspekty řešených problémů při vývoji aplikace. Předností je volná vazba mezi jednotlivými částmi, které jsou tak snáze zaměnitelné. Například model nemusí sloužit jen webovému serveru prezentujícímu HTML stránky, ale například webové službě případně konzolové aplikaci. Vzor předepisuje, kam patří jednotlivé části řešeného problému, ale zároveň nenutí k žádným konkrétním řešením na nižší úrovni v rámci modelu nebo controlleru. Zda samotný business model bude dále nějak strukturovaný či vrstvený není v kompetenci MVC.

Oddělení problémů dovoluje nezávislý vývoj, kdy se v jednu chvíli můžeme soustředit pouze na jeden konkrétní aspekt celku.

Můžeme si dovolit ještě malé srovnání s *Web Forms* (WF) z hlediska testovatelnosti výsledné implementace. Ve WF aplikacích zajišťovala jedna třída zobrazování výstupu pro uživatele a taktéž zpracování jeho vstupu. Kvůli tomu bylo obtížné vytvářet automatizované testy pro aplikace ve WF, neboť otestování jedné stránky v sobě zahrnovalo vytvořit instanci třídy stojící za danou stránkou, všech elementů, ze kterých daná stránka sestávala, a tříd, na kterých daná třída závisela. Do této složité struktury se následně musela podvrhnout testovací data. Naproti tomu MVC více sází na nezávislá rozhraní, která dovolují testovat dílčí akce dokonce i bez potřeby webového serveru.

Díky důrazu Microsoftu na testovatelnost a podporu test-driven development (TDD) je možné vytvářet *mockovací objekty*, což jsou objekty simulující chování skutečných objektů v aplikaci. Jejich cílem je izolovat část testovaného problému od okolí a zamezit tak vlivům (především chybám) v jedné části aplikace na jiné testy. Testy lze vytvářet přímo v nástroji Visual Studio, nicméně nechybí podpora i pro alternativní unit-test frameworky určené pro .NET.

6.6 Zpracování požadavku na serveru

Při odeslání požadavku na webový server musí dojít k instancování řady tříd, které se starají o bezproblémové vyřízení žádosti klienta. Zaměříme se nyní na proces, ke kterému dochází při přijetí požadavku od samého začátku v aplikaci založené na MVC.

Jako první přichází s požadavkem do styku *UrlRoutingModule*, což je HTTP modul, jehož zodpovědností je správně přeložit adresu a cestu v požadavku na odpovídající cestu na serveru. S pomocí vlastního *UrlRoutingModule* lze například realizovat zpracování SEO adres nebo jiných zvláštností v adresách.

UrlRoutingModule pracuje interně s uspořádanou množinou cest na serveru, které postupně prochází a konfrontuje s požadavkem. Vybrána je vždy první cesta, která odpovídá požadavku. Může samozřejmě nastat situace, kdy žádná cesta není vyhovující, v tom případě se o zpracování požadavku dále stará klasické prostředí ASP.NET nebo IIS server.

Výsledkem zdárného nalezení cesty je výběr třídy implementující *RouteBase*, která je typicky instancí třídy *Route* (ale nemusí tomu tak být). Zdpovědností *Route* objektu je pak navrátit *UrlRoutingModulu* objekt implementující rozhraní *IRouteHandler*, který je spjat

s objektem *Route*, jenž ho vrátil. Typickým případem je, že *IRouteHandler* je přímo instancí *MvcRouteHandler*.

Proces nalezení správné cesty na serveru se blíží ke svému konci, neboť *IRouteHandler* již jen vytváří objekt respektující kontrakt *IHttpHandler*, kterému je poslán *IHttpContext*, což je kontext aktuálního požadavku na server. Ve výchozím zpracování je *IHttpHandler* reprezentován třídou *MvcHandler*, která nakonec je zodpovědná za výběr správného controlleru, jenž zpracuje došlý požadavek.

Routing modul a handler tedy zajišťují nalezení správného vstupního bodu do MVC frameworku použitého v naší aplikaci. Jejich posledním úkolem je získat instanci controlleru a zavolat jeho metodu *Execute* realizující požadové chování serveru.

Pro snazší pochopení slouží diagram 6.2

6.7 Styly prezentační vrstvy – Less CSS

V prostředí webových aplikací je zpravidla značkovací jazyk (markup) doplněn o nějakou formu vyjádření vizuálního stylování prvků na stránce. Pakliže dbáme na striktní oddělení sémantické části od vzhledu, nevyhneme se použití externích stylových předpisů známých jako Cascading Style Sheets (CSS). Nemá smysl se zde rozepisovat o CSS do hloubky, neboť obecný princip je těm, kdo přišli do styku s webovými technologiemi, většinou znám.

Zajímavým rozšířením je však preprocesor zvaný Less CSS [2]. Základem CSS jsou selektory s blokem pravidel. Hovořit o znovupoužitelnosti kódu napsaného v CSS lze jen do té míry, že nějakému bloku pravidel přiřadíme více selektorů, čímž pravidla uplatníme vícekrát. V tom přináší Less CSS takřka revoluční změnu.

Preprocesor Less CSS vznikl v roce 2009, jedná se tedy o poměrně novou záležitost, kterou považuji za přínosné zde představit jako jedno z vylepšení prezentační vrstvy aplikace. Díky Less CSS můžeme deklarovat a definovat vlastní proměnné, které poté použijeme uvnitř jednotlivých pravidel. Na začátku souboru tak lze například definovat barvy pozadí nebo okrajů stránky a použitím proměnné mít zajištěno, že při případné změně se provede jedna centrální úprava s dopadem na všechny výskyty.

6.7.1 Mixins a zanořená pravidla

Užitečnou vlastností je podpora tzv. *mixins*, což jsou znovupoužitelné bloky pravidel. Uvažujme třeba situaci, kdy potřebujeme více verzí pravidla pro různé prohlížeče:

```
selector {
  opacity: 0.6;
  filter: alpha(opacity=0.6);
}
```

Na všech místech, kde budeme chtít nastavovat průhlednost, musíme vždy zopakovat uvedené. Nebo vytvořit a použít *mixin*:

```
.opacity(@howMuch) {
  opacity: @howMuch;
  filter: alpha(opacity=@howMuch)
}
selector {
  .opacity(0.6); }
```

Preprocesor pak provede odpovídající nahrazení v místě všech použití. Další typickou situací je potřeba zanořování selektorů, např.:

```
div.menu {
  margin: 10px;
}
div.menu ul {
  list-style-type: circle;
}
div.menu ul li {
  ...
}
```

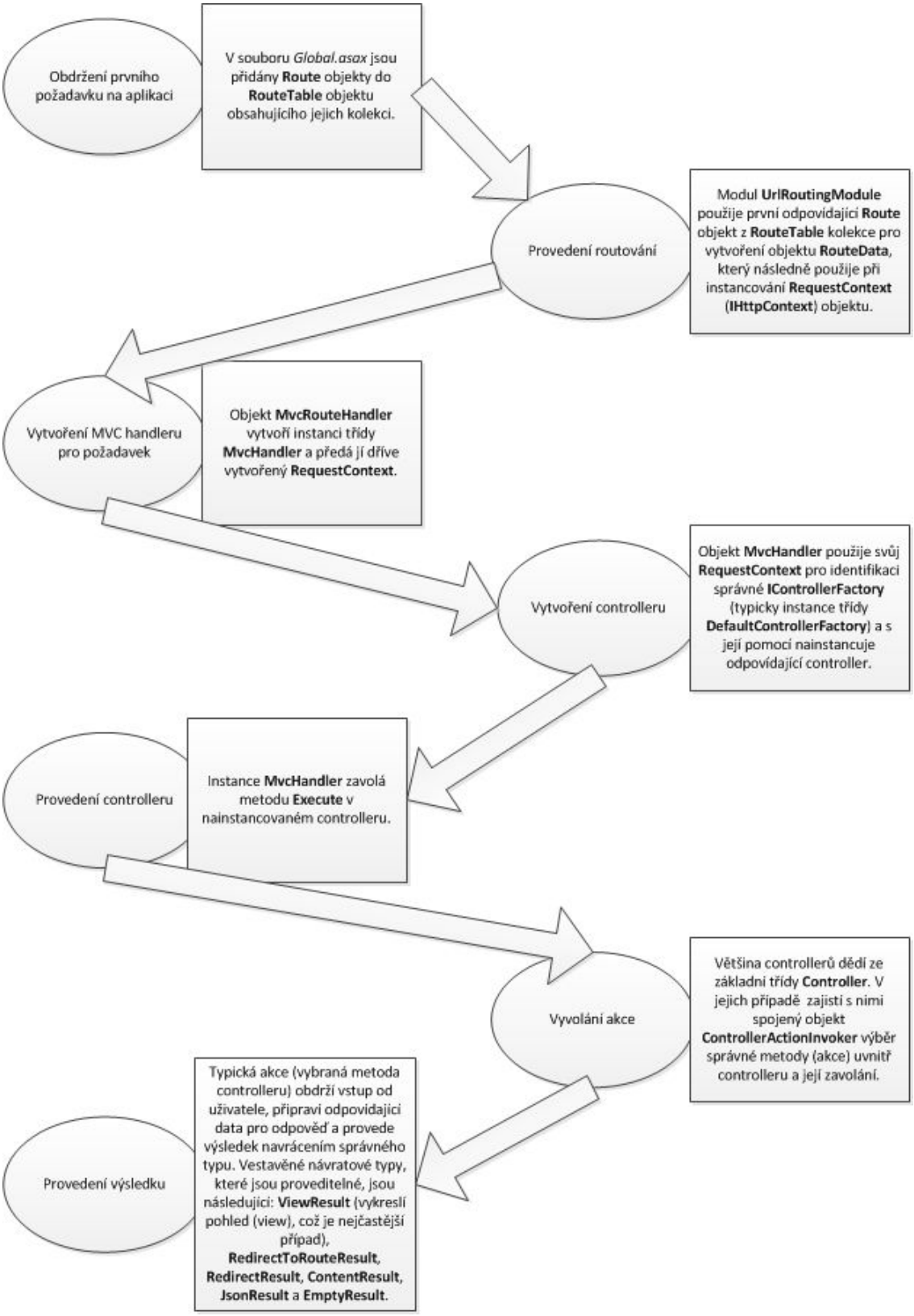
Můžeme nahradit následujícím zápisem:

```
div.menu {
  margin: 10px;

  ul {
    list-style-type: circle;

    li {
      ...
    }
  }
}
```

Vyhneme se tak nepříjemnému opakování částí selektorů, které za nás může vygenerovat preprocesor. Vítaným vylepšením je také možnost importu souborů, kdy můžeme CSS rozdělit do více modulů a ty následně spojit, aniž bychom museli do výsledného HTML generovat tag pro vkládání stylového předpisu modul po modulu. V praktické situaci je běžné vytvářet moduly znovupoužitelných pravidel, které se následně importují do jednoho souboru a v něm užijí dle potřeby.

Less nabízí také podporu pro aritmetické operace, se kterými můžeme dopočítávat velikosti okrajů nebo odstíny barev (světlejší/tmavší než zadaná). Uvedené umí i jedna z vestavěných funkcí nazvaná `lighten(color, howMuch)`. Nechybí podpora pro jmenné prostory mající zamezit konfliktům v modulech různých autorů nebo podmíněné použití volaného *mixin* v závislosti na hodnotě některého parametru.

Užitečné mohou být i řádkové komentáře, které klasické CSS postrádá. V Less se jedná o druh komentáře, který je určen pouze vývojářům a do výsledného CSS souboru se nezahrnuje. Stejně tak nechybí vestavěná podpora pro přímé vytváření miniaturizované verze CSS zbavené přebytečných bílých znaků, ať už za účelem ušetření přenosového pásma nebo zhoršení čitelnosti souboru (jednoduchá obfuskace).



Obrázek 6.2: Schéma zpracování příchozího požadavku na server v MVC (vytvořeno na základě [17])

Kapitola 7

Návrh aplikace

Stěžejní částí pro správnou implementaci dolovacího algoritmu je promyšlený návrh aplikace, která bude realizovat jak samotné dolování dat ze sociální sítě, tak prezentaci pro uživatele. Návrh respektuje architektonický vzor MVC blíže popsaný v kapitole 6. V této kapitole se blíže zaměříme na dekompozici problému na jednotlivé třídy, zajišťující požadovanou funkčnost v jednotlivých částech architektury.

7.1 Model

Na základě analýzy dolovacího algoritmu vznikl návrh business modelu aplikace. Ten sestává z několika podčástí, jež jsou spolu vzájemně provázány. Patří sem část zajišťující persistenci datových struktur, část pro realizaci samotného dolování, dotazování API sociální sítě Twitter, správa uživatelský účtů pro dolování a pomocné třídy. Všem postupně dáme větší či menší prostor pro vysvětlení jejich účelu.

7.1.1 Persistence

Pro objekty, u nichž je potřeba, aby přežily čas běhu samotné aplikace, je určena relační databáze Microsoft SQL Server ve verzi 2012. Mapování objektů na n-tice relace má na starosti Entity Framework (EF). Jedná se o open-source řešení určené pro platformu .NET. EF nabízí možnost nejen modelování samotných entit business logiky aplikace a vztahů mezi nimi, nýbrž i z programátorského hlediska pohodlný způsob dotazování nad daty prostřednictvím Language Integrated Query (LINQ).

LINQ je vrstva abstrakce umožňující dotazovat takřka libovolný zdroj dat, pro který existuje implementace (objekty, XML dokumenty, SQL, DataSety), nebo pro kterou si vlastní implementaci vytvoříme. LINQ je úzce spjaté s platformou .NET, ačkoli existují i verze pro Javu, PHP či JavaScript.

Jedna z hlavní entit je reprezentace dolovací úlohy pomocí třídy *MiningTask*. Jedna dolovací úloha stojí vždy za jedním iniciálním hashtagem zadaným uživatelem. O úloze je potřeba zaznamenat si především její aktuální stav (vytvořená, běžící, dokončená, zrušená, chyba) a pro lepší informovanost uživatele také čas vzniku.

S úlohou *MiningTask* je přímo svázána třída *TwitterStatus*, jež se stará o reprezentaci příspěvků ze sociální sítě Twitter (tweetů). Každá dolovací úloha s sebou asociuje množství takovýchto příspěvků získaných při procesu dotazování. *TwitterStatus* pak sestává z objektů *TwitterStatusHashtag*, mezi kterými je vztah kompozice. Každý status obsahuje nějaké

hashtagy (principielně to není podmínka, nicméně tweety bez hashtagů nemají pro proces dolování žádný význam).

Třída *ThreadProgress* reprezentuje informaci o procesu dolování vzniklou v určitém časovém okamžiku. Z toho důvodu obsahuje časové razítko a dále zprávu, popisující nastalou událost a její významnost. Při běhu asynchronního dolovacího vlákna může toto díky tomu nahlašovat svůj postup a případné neočekávané situace.

Nezbytnou pro dotazování rozhraní sociální sítě se jeví entita *User*, která nereprezentuje uživatele dolovací aplikace jako takové, ale uživatelský účet na Twitteru. Na vrub uživatelského účtu se pak vždy počítá aktuální limit dotazů na programové rozhraní. O uživateli se zaznamenává jeho jméno, které slouží jen pro snadnou orientaci ve více účtech, dále přístupové *tokeny*, vydáváné Twitterem jednotlivým uživatelů, identifikátor uživatele a příznak, zda je účet *aktivní*. Takovýto účet je pak používán asynchronním vláknem při dotazování. Aktivní může být v jednu chvíle pouze jeden účet.

Z důvodu nutnosti počítat a řídit počet dotazů na API Twitteru je potřebná entita *QueryHistory*. Jejím úkolem je představovat dotaz na určitý zdroj na síti. Právě ke zdrojům se váží jednotlivé limity. Kromě identifikace zdroje se pak uchovává informace o čase provedení dotazu, a zda se jedná o tzv. *otevírací dotaz* (*window opener*). Twitter pracuje s časovým oknem, které je vždy zahájeno prvním dotazem a skončí typicky po 15 minutách. Od otevíracího dotazu se pak počítají další dotazy vstříc limitu.

Poslední významnou entitou je *MiningResult*, což je třída zpodobňující uložený výsledek dolování. Vždy, když si uživatel poznamená aktuální parametry při prezentaci výsledků, vytvoří se persistentní reprezentace této entity. Kromě identifikace dolovací úlohy je obsaženo jméno (umožňuje uživateli přehledně si výsledek označit), aktuální parametry, čas vzniku a serializovaná reprezentace výsledku (ve formátu XML).

7.1.2 Optimalizace čtení persistentních dat

Vše výše uvedené dovoluje správnou funkčnost aplikace po stránce přetrvání dat v čase. Nicméně co je z hlediska relací vhodné (normalizované), nemusí být již výhodné v procesu zpracování nasbíraných dat. Při dolování v datech se velice často dotazuje nad relacemi *TwitterStatus* a *TwitterStatusHashtag*, mezi nimiž se provádí operace spojení. Vzhledem k tomu, že MSSQL nepodporuje uložení dvou tabulek do jednoho clusteru (jako to umožňuje například Oracle), jeví se zde výhodné vytvořit materializovaný pohled. V terminologii MSSQL se jedná o pohled, nad kterým je definovaný clusterovaný index, neboť nic jako materializovaný pohled (známý právě např. z Oracle) vytvářený příkazem jazyka SQL zde není.

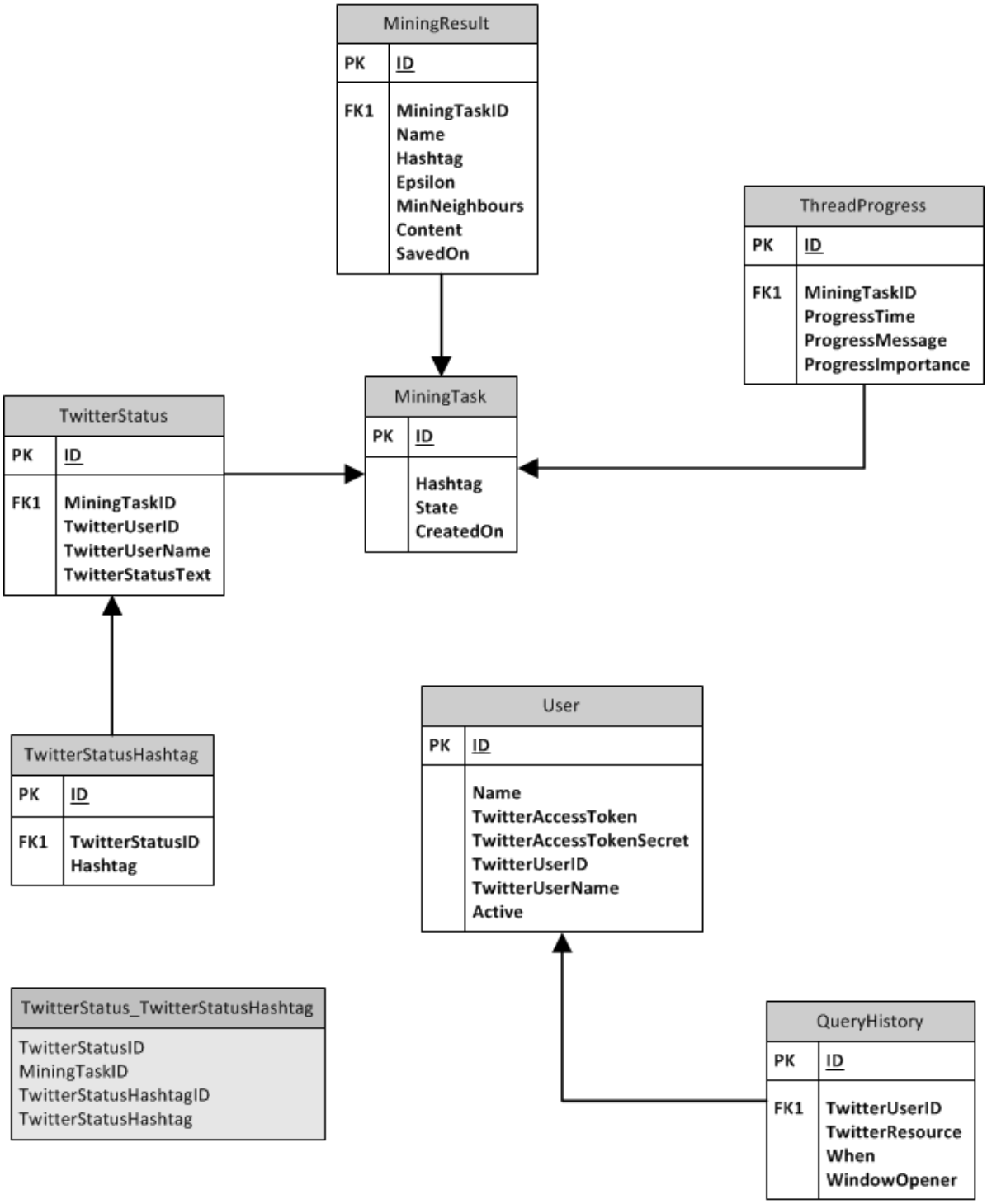
Pohled zahrnuje sloupec identifikující n-tici *TwitterStatus*, identifikátor dolovací úlohy *MiningTask* a dále přes cizí klíč vazbu na *TwitterStatusHashtag*. Materializaci pohledu zajišťuje clusterovaný index nad všemi sloupci, který je navíc doplněn o další neclusterovaný index nad sloupci obsahujícími identifikaci *TwitterStatus* a *Hashtag*. Oba indexy jsou z hlediska funkčnosti aplikace klíčové, co se týká výkonnosti (odezvy při práci uživatele s aplikací) a byly navrhovány v těsné vazbě na konkrétní SQL dotaz zajišťující dolování. Jemu bude prostor věnován později.

7.1.3 Schéma databáze

Všechny popsané entity odpovídají databázové reprezentaci schématu, neboť principem Entity Frameworku je vygenerovat popisné třídy odpovídající 1:1 existující databázi. Stejně

tak lze z vygenerovaných tříd vytvořit novou databázi, například v případě nasazení aplikace na jiném vývojovém počítači.

Pro názornější ilustraci navrženého schématu slouží diagram 7.1, ze kterého je lépe patrná provázanost jednotlivých relací v databázi a jejich domény.



Obrázek 7.1: Schéma relační databáze

7.1.4 Dolovací úloha a dotazování API

Každá dolovací úloha je za běhu reprezentována samostatným vláknem, které běží asynchronně vůči webové aplikaci. Dolovací úlohu musí být možné sledovat (jak je daleko ve své činnosti, zda došlo k chybě) a z pohledu uživatele musí být možné ji kdykoli bezpečně ukončit. V aplikaci představuje dolovací úlohu třída *MiningTaskThread*. Po vytvoření své instance konstruktorem má k dispozici všechny potřebné informace a její práce se odehrává především v metodách *ThreadWork* a *CollectTweetsFromTwitter*. První jmenovaná zajistí nahlášení stavu vlákna po jeho začátku i při jakémkoli způsobu skončení, vždy voláním metody *ReportProgress*, jež se postará o zanesení informace do databáze. Druhá potom zajistí nastartování procesu dotazování programového rozhraní Twitteru za účelem sběru tweetů.

Pro poskytnutí možnosti ukončení vlákna slouží metoda *Cancel*, doplněná o *Join*, díky čemuž se k asynchronně běžícímu vláknu lze připojit a počkat na něj, aby mohl být ohlášen výsledek ukončení vlákna.

7.1.5 Správa vláken

S ohledem na množství vláken, které mohou na serveru běžet, je potřeba mít nad nimi kontrolu a přehled. O správu vláken se pro tento účel stará třída *MiningTaskThreadPool*, jež obsahuje kolekci všech dolovacích vláken. Ke kolekci lze přes vymezené rozhraní přistupovat metodami, které zajišťují vláknovou bezpečnost (jsou tzv. *thread-safe*). Potřeba zajistit bezpečný přístup vyvstává ze skutečnosti, že webová aplikace může mít více uživatelů, kteří mohou chtít v jeden okamžik přidat nové vlákno nebo ukončit běh již existujícího vlákna a za žádných okolností by nemělo dojít k tomu, že nesynchronizovaným přístupem do kolekce se reference na některé vlákno řádně nezaznamená.

Zmíněný účel naplňují metody *AddMiningTask* a *CancelMiningTask*. Druhá jmenovaná po svém zavolání počká nejvýše po vymezený časový interval na skončení asynchronně běžícího vlákna. Pouze pro účely statistiky je poskytována metoda *Count* sdělující přesný počet aktuálně existujících vláken na serveru.

V případě vláken není nikdy jistota, že jejich činnost bude probíhat podle očekávání. U vytvořeného vlákna může dojít k situaci, že bude jeho činnost z nějakého důvodu nepředvídatelně ukončena (například nedostatek paměti na serveru, neočekávané ukončení serveru). V takovém případě nepomůže ani pečlivé nahlášení stavu vlákna v metodě *MiningTaskThread.ThreadWork*, neboť její kód se vůbec nemusí vykonat. Předestřený problém řeší metoda *CleanupUnexpectedlyTerminatedTasks*, jejímž úkolem je konfrontovat aktuálně běžící vlákna na serveru se stavem vláken v databázi a všechny rozdíly mezi skutečně a domněle běžícími vlákny prohlásit za neočekávanou chybu v daném vlákně.

7.1.6 Crawler

V metodě MiningTaskThread.CollectTweetsFromTwitter se vytvoří instance třídy *Crawler*, na kterou je následně delegována zodpovědnost za sběr dat napříč sociální sítí. Třída pro svou činnost potřebuje znát, ke které dolovací úloze náleží, aby se usnadnil proces hlášení aktuálního postupu v dolování.

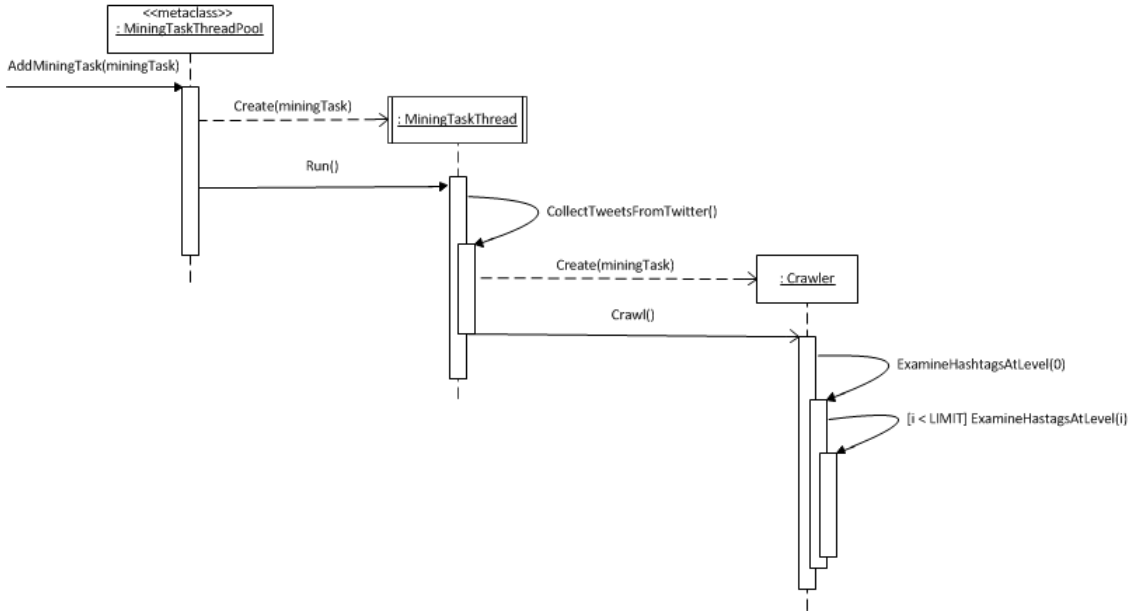
Práce začíná v metodě *Crawl*, která vloží iniciální hashtag z dolovací úlohy do seznamu dosud nezpracovaných hashtagů. Celý proces dolování pracuje v úrovních, kdy v každé úrovni dolování se sbírají hashtagy pro tu následující. Dotazování se započne voláním *ExamineHashtagsAtLevel*.

Metoda *ExamineHashtagsAtLevel* pracuje vnitřně s několika kolekcemi, pomocí nichž řídí svůj postup. Jednou z nich je právě kolekce seznamů organizovaných podle úrovně, ze kterých se bere vždy hashtag pro další dotaz na API. Další je pracovní kolekce hashtagů obsažených v aktuálně vyšetřovaném tweetu, významnou je kolekce již zpracovaných hashtagů a poslední je kolekce již vyšetřených tweetů.

Crawler za žádných okolností nedotazuje dvakrát na jeden hashtag, proto vždy dbá na to, aby do hashtagů pro další úroveň nevložil nějaký již zpracovaný. Není však v jeho moci ovlivnit, jaké tweety se vrátí na ten který dotaz na API. Proto při iteraci přes navrácené statusy z Twitteru konfrontuje identifikátor každého s již zpracovanými statusy a duplikáty znovu nezahrnuje do nasbíraných dat. Pro každý vyšetřený status se do persistentního úložiště poznačí, jaké hashtagy se k němu váží. Hashtagy lze vyčíst ze samotného obsahu tweetu, nicméně vzhledem k jejich významnosti pro aplikační model je každý hashtag modelován jako entita.

Sběr dat v každé úrovni skončí s vyprázdněním seznamu hashtagů pro zpracování. V takový okamžik se rekurzivně volá metoda *ExamineHashtagsAtLevel* s čítačem úrovně o jedničku větším. Každá rekurze musí mít svou ukončující podmínku, v našem případě je jí tvrdý limit pro počet vyšetřovaných úrovní. Alternativou by bylo zvolit nejvyšší možný počet dotazů na API na jednu dolovací úlohu nebo počet nasbíraných hashtagů, všechny tři varianty se jeví jako relevantní. Hodnota limitu úrovně je nastavována s ohledem na restrikce ze strany Twitteru, neboť nemá smysl pro demonstrační účel aplikace čekat několik dnů na sběr tisíců statusů s přihlédnutím k rozkládání dotazů v čase.

Zpracování hashtagů je necitlivé na velikost písmen (*case-insensitive*), protože ani Twitter při dotazování nerozlišuje jejich velikost.



Obrázek 7.2: Sekvence volání metod při vytváření a běhu dolovací úlohy

Posloupnost volání při vytváření a běhu dolovací úlohy (vlákna) demonstruje diagram 7.2. Pro zjednodušení byla vypuštěna volání *ReportProgress*, která se vyskytují po nastartování vlákna *MiningTaskThread* a před jeho skončením.

7.1.7 Řízení počtu dotazů v čase

O řízení počtu položených dotazů na API Twitteru se stará trojice tříd. Jmenovitě *QueryHelper*, *QueryHistoryManager* a *QueryResource*. Úlohou první zmiňované je v metodě *WaitForQuery* čekat na správný okamžik, kdy lze dotaz bezpečně položit. Metoda se pokouší vložit záznam do databáze voláním metody *QueryHistoryManager.CreateRestricted*. Tato metoda vždy atomicky zkontroluje, kolik dotazů na API bylo položeno od posledního dotazu *window opener* na příslušný zdroj (*resource*) a pakliže nebyl dosažen stanovený limit, zaznamená nový dotaz do databáze a vrátí informaci o tom, že samotný dotaz může být realizován. V opačném případě vrátí informaci o tom, kdy bude dotaz moci být položen (čas vypršení platnosti okna od dotazu *window opener*). Právě metoda *WaitForQuery* se stará o rozložení času čekání na dotaz do menších časových úseků, mezi kterými kontroluje, zda nebyl vznesen požadavek na ukončení vlákna ze strany uživatele, a také oznamuje aktuální stav vlákna, tedy že vlákno čeká do stanoveného času.

Poslední zmíněná třída *QueryResource* se stará o výpočty spjaté s počítáním položených dotazů. Jejím úkolem je znát, kolik dotazů v okně může být na zdroj položeno a zároveň zahrnovat do výpočtu určitou bezpečností rezervu. Ta se užívá z důvodu rozdílného chápání času dolovací aplikací a programovým rozhraním Twitteru. V okamžik zaznamenání dotazu do databáze třídou *QueryHistoryManager* vidí aplikace čas položení dotazu jako T_1 a pokud se jedná o první dotaz, je tento *window opener*. Následně je odeslán HTTP dotaz na server, u nějž není garantováno, za jak dlouho na něj dorazí a jak dlouho potrvá jeho zpracování. API Twitter si tak *window opener* poznačí v čase T_2. Ten je pro aplikaci neznámý. Známý je pak čas obdržení odpovědi od serveru v čase T_3. Z uvedeného důvodu aplikace počítá s bezpečnostní rezervou v počtu dotazů (a také s mírně prodlouženou velikostí okna).

7.1.8 Dolování v datech nad MSSQL databází

Poté, co jsou relevantní data pro zadaný iniciální hashtag posbírána do relační databáze, může nad nimi být zahájen proces dolování. Srdcem dolování je třída *MiningQuerier*, jejíž práce začíná typicky voláním metody *GetNeighbourHashtagsAdvanced* (slovo *advanced* je dáno výsledkem experimentování s prototypem dolovací aplikace a respektuje jeho postupnou evoluci).

Metoda přijímá jako parametry iniciální hashtag, od nějž začne dolovat a parametry ϵ a *minNeighbours*, jejichž význam objasňuje kapitola věnující se analýze dolovacího algoritmu. Jejím výsledkem je množina dvojic *(hashtag, sousední hashtagy)*, která popisuje vazby a jejich kardinalitu v rámci získaného shluku. Metoda začíná od předaného hashtagu, který si poznačí do množiny hashtagů ke zpracování. Následně se z množiny vždy odebere jeden hashtag a v metodě *GetNeighbourHashtagsImproved* se dotáže databáze. Zde věnujme prostor znění pokládaného dotazu, jehož existence byla zmíněna v části 7.1.2.

```
SELECT [TwStatusHashtag] AS [Hashtag], COUNT(TwStatusHashtag) as [Count]
FROM [TwitterStatus_TwitterStatusHashtag]
WHERE [TwStatusID] IN (
  SELECT [TwStatusID] FROM [TwitterStatus_TwitterStatusHashtag]
  WHERE [MiningTaskID] = @MiningTaskID AND [TwStatusHashtag] = @Hashtag
)
GROUP BY [TwStatusHashtag]
HAVING COUNT([TwStatusHashtag]) >= @Epsilon
ORDER BY [Count] DESC
```

Nejprve si osvětleme význam dotazu. Úvodní *select* vybírá sousední hashtagy a agreguje jejich počet pro množinu tweetů vymezenou vnořených dotazem. Klauzule *group by* definuje, jak bude probíhat agregace a *having* následně vybere jen ty sousední hashtagy, jejichž násobnost je alespoň *epsilon*. Vnořený dotaz vymezí množinu tweetů na základě toho, zda se v nich vyskytuje hashtag, pro nějž aktuálně vyhledáváme sousední hashtagy a kardinalitu jejich vazeb. Nutnou podmínkou je také příslušnost k dolovací úloze.

Pokud se nad dotazem hlouběji zamyslíme, zjistíme, že první *select* potřebuje pro svou účinnost disponovat indexem nad sloupcem *TwStatusID* (identifikace tweetu). Pouze takový index by však následně vedl na přístup do příslušného bloku dat v tabulce, aby mohla být zjištěna hodnota *TwStatusHashtag*. Proto je index vytvořen nad sloupci *(TwStatusID, TwStatusHashtag)*. Jedná se o index neclusterovaný a postačuje jak k selekci hodnoty sloupce, tak pro výpočet odpovídající agregace nad tímto sloupcem.

Vnořený dotaz vybírá identifikaci tweetu na základě identifikace dolovací úlohy a hashtagu v daném tweetu. Index vytvořený nad těmito sloupci tedy poskytne potřebná data v rámci čtení jednoho bloku indexu bez nutnosti přistupovat do datových bloků tabulky. Pořadí sloupců *(MiningTaskID, TwStatusHashtag, TwStatusID)* odráží posloupnost, v jaké si databáze musí sloupce vybavit. Vzhledem k tomu, že MSSQL neumí přeskočit sloupec v indexu (a později se k němu vrátit), nelze *TwStatusID* specifikovat dříve, neboť by index nebyl příliš použitelný. S ohledem na skutečnost, že chceme pohled materializovat, se musí jednat o clusterovaný index.

Vysvětlený dotaz pokládá tedy metoda *GetNeighbourHashtagsImproved* a vrací odpovídající řádky relace jako seznam. Po návratu do metody *GetNeighbourHashtagsAdvanced* se vyhodnotí, zda počet sousedů odpovídá specifikované podmínce *minNeighbours* a aktuálně zpracovávaný hashtag je tedy jádro, či nikoli. Pouze sousední hastagy jádra jsou dále uvažovány, a proto poznačeny do množiny hashtagů ke zpracování.

Každý takto vyšetřený hashtag je zanesen do množiny již zpracovaných, aby při procházení grafem byla zajištěna orientovanost a nedocházelo k procházení ve smyčkách.

7.1.9 Ukládání vydolovaných dat

Dotazování nad databází je oproti programovému rozhraní Twitteru nepoměrně rychlejší, přesto můžeme uživateli dovolit pracovat s aplikací ještě rychleji díky podpoře pro ukládání vydolovaných výsledků. Za tímto účelem se jeví jako vhodné mít k dispozici serializovatelnou reprezentaci vydolovaných dat, o což se stará trojice tříd *HashtagNeighbourResultRowPair*, *NeighboursResultRow* a *SavedResults*. Jedná se o popis statického uložení dat ve formátu XML s využitím atributů pro definici značek z knihovny .NET *System.Xml.Serialization*. Samotný převod obstarává třída *SavedResultsHelper* s metodami *ResultsToString* a *ResultsFromString*.

Výhodou popsaného mechanismu je zejména odpadávající nutnost opakovaně dotazovat DB při zjišťování sousedů pro aktuálně vyšetřovaný hashtag, což vede vždy na přístup ke dvěma indexům pro každý dotaz. Předpočítaná data v XML podobě se uloží jako jeden z prvků n-tice do *MiningResult* a čtení tak znamená jediný dotaz. Zpracování serializace/deserializace v paměti aplikačního serveru si s rychlostí databáze pochopitelně nezadá.

Výhodou uložení není jen zrychlené znovunačtení zobrazených výsledků, nýbrž také možnost specifikovat odlišnou hodnotu parametru *minNeighbours*, která však musí být větší nebo rovna hodnotě užité pro výpočet persistovaného výsledku. Jiná hodnota *minNeighbours* znamená projít v paměti strukturu popisující vydolovaný výsledek bez nutnosti kontaktovat databázi, práce s takovouto reprezentací je proto rychlejší.

7.1.10 Doplňující třídy modelu

Implementace pracuje s třídou pro správu uživatelů *UserManager* a rozšířením metod entity *User* prostřednictvím *UserExtensions* třídy. Více informací o rozšiřujících metodách, což je v současnosti spíše specialita jazyka C#, je možné nalézt v dokumentaci [18]. Práce s uživateli nezahrnuje nic výjimečného kromě logiky zajišťující, že vždy je aktivní pouze jeden účet.

Zajímavostí je pak třída *ThreadSingleton*, což je implementace návrhového vzoru singleton pro prostředí vícevláknových aplikací. Jejím cílem je poskytovat každému vláknu pro dolování existenci jediné instance třídy v kontextu daného vlákna (ne tedy v celé paměti aplikace). Použití generik dovoluje instancovat libovolnou třídu.

7.2 View a Controller

Části aplikace věnující se pohledům a controllerům si popíšeme ve společné podkapitole. Jejich implementace se nalézá v samostatném projektu z důvodů dříve popsaných v části 6.2.

Ve třídě *UserController* se nachází logika spjatá se zpracováním požadavků a přípravou dat pro pohledy poskytující možnosti správy účtů pro dolování. Podporováno je také zobrazení souhrnných statistik z Twitteru, týkajících se aktuálního stavu vyčerpání limitů na počet dotazů.

MiningController obsluhuje požadavky prezentační části v oblasti řízení dolovacích úloh respektive jejich vláken. Uživatel zde může založit novou úlohu pro zamýšlený hashtag, spustit ji a sledovat její průběh. Případně ji také ukončit, pakliže již není pro něj relevantní.

Pro úspěšně splněné i předčasně ukončené úlohy si lze zobrazit jejich (částečné) výsledky a určovat si parametry ϵ a *minNeighbours* při prohlížení.

Uložené výsledky pak spravuje controller *SavedResultsController*, který v zájmu dosažení uživatelské přívětivosti poskytuje prostředek pro pojmenování/přejmenování uložených výsledků, například pokud si v čase nastřádáme více výsledků pro stejný hashtag a chceme je pak mít možnost od sebe odlišit a porovnávat.

7.2.1 Rozšíření

Je až s podivem, že platforma .NET nenabízí pro webové aplikace vestavěnou možnost zobrazování zpráv po přechodu na jinou stránku (zvané často *flash messages*). Je zde však podpůrný prostředek, který umožňuje jejich snadnou realizaci v podobě vlastnosti každého MVC controlleru *TempData*. Jedná se o instanci třídy *TempDataDictionary*, jejímž jediným úkolem je zajistit přežití persistentních dat mezi dvěma požadavky. Pokud tedy data do něj uložíme, načteme je v dalším požadavku a dále jsou již automaticky odstraněna.

Uvedeného využívají rozšiřující metody (*extension methods*) umístěné ve třídě *ControllerExtensions*, které poskytují rozhraní pro vytváření *flash* zpráv a jejich renderování do HTML markupu stránky.

Stejná třída zapouzdřuje také metody pro práci s *drobečkovou navigací*, což je forma navigace udávající uživateli, kde se na webu nachází z pohledu jeho logického vnímání (neboť všechny *pohledy* v rámci controlleru jsou definované na stejné úrovni).

Přehledné zobrazování ladících výpisů zajišťují rozšiřující metody pro přidávání a renderování výjimek vzniklých při běhu aplikace a zachycených v controlleru. Ladící výpisy

bývají obecně považovány za citlivou informaci, a proto se jejich zobrazování řídí nastavením v konfiguračním souboru webu (*web.config*) direktivou *EnableErrorDebuggingMessages*.

Kapitola 8

Experimenty

V části věnované experimentování s navrženou aplikací se budeme zabývat měřením procesu dolování z hlediska jeho rozsáhlosti a také rychlosti. Zajímavými parametry pro nás bude velikost souboru nadolovaných dat z programového rozhraní sociální sítě Twitter – kolik tweetů se v průměru dá získat, jaké množství hashtagů obsahují či kolik z nich je jádrem.

Zajímavým parametrem také může být průměrná velikost shluku nebo kolik vztahů s dalšími sousedními hashtagy na jeden hashtag připadá.

V části věnované rychlosti aplikace se blíže podíváme na časovou náročnost dolování v získaných datech. Cílem je poskytnout orientační hodnotu uživatelské odezvy aplikace při práci s ní, neboť dolování v získaných datech je z pohledu uživatele důležitá činnost, při které je vhodné, aby práce byla skutečně interaktivní. To je značný rozdíl oproti procesu získávání dat z API, kdy se uživatel může věnovat jakékoli jiné činnosti, neboť data se získávají zcela automaticky.

8.1 Měření rozsahu souboru dat z programového rozhraní Twitteru

Nejprve se podívejme blíže na každou dolovací úlohu, jejíž stěžejní částí je pokládání dotazů na API. V následující metrice se podíváme, kolik dotazů na API je třeba učinit pro získání ucelených podkladů pro následné dolování v datech. (Pozn.: Twitter nerozlišuje velikost písmen a aplikace používá *lower-case* reprezentaci, proto jsou všechny hashtagy uváděny malými písmeny).

8.1.1 Počet dotazů na dolovací úlohu

Z kapitoly věnující se analýze či implementaci zisku dat z programového rozhraní se uvádí, že data se získávají v úrovních. V první úrovni je vždy právě jeden hashtag (iniciální), počet hashtagů v dalších úrovních již závisí na konkrétních podmínkách na síti – jak je téma oblíbené mezi uživateli a jakým způsobem o něm sami informují.

Nejprve pro vytipované hashtagy zjistíme, kolik dotazů je potřeba při dvou úrovních dotazování. Následně ještě srovnáme nárůst počtu dotazů při zvýšení maximálního počtu úrovní o jednu.

Pro experiment byly vybrány hashtagy reprezentující roční období a dále hashtagy s názvy vybraných evropských měst. Důvodem jejich výběru je autorova domněnka, že by se mohlo jednat o relativně zajímavé pojmy z hlediska dolování, v jejichž případě se nejedná

o sezonní záležitost, módní trend či politicky ožehavé téma současné doby, jinými slovy jejich výskyt v čase by měl být zhruba konstantní (u ročních období možná s vlivem aktuálního času v roce). Hodnoty byly naměřeny dne 13. dubna 2014.

Hashtag		autumn	fall	spring	summer	winter
Počet dotazů	**Úroveň 1**	1	1	1	1	1
	Úroveň 2	216	217	259	261	184

Tabulka 8.1: Počet dotazů – roční období

Hashtag		berlin	bern	brno	brussels	helsinki
Počet dotazů	**Úroveň 1**	1	1	1	1	1
	Úroveň 2	210	207	168	193	147
Hashtag		oslo	paris	prague	stockholm	wien
Počet dotazů	**Úroveň 1**	1	1	1	1	1
	Úroveň 2	227	159	206	154	128

Tabulka 8.2: Počet dotazů – evropská města

Zhodnocení pro 2 úrovně

Z tabulky 8.1 vyplývá, že uvedené iniciální hashtagy nevystačí s limitem velikosti okna pro dotazování (180 dotazů za 15 min) a s průměrem 227 dotazů na API tak vyžadují alespoň dvě okna. V případě vybraných evropských měst je situace mírně odlišná, neboť některá města se do velikosti okna vejdou, jiná nikoli. Spíše shodou náhod vychází průměrný počet dotazů na 180.

Z uvedeného experimentu vychází, že pro účely demonstrace jsou dvě úrovně dotazování jakýmsi ideálem, neboť nevyžadují enormní množství času. Pro představu zopakujeme uvedený experiment pro tři úrovně pouze s ročními obdobími. Výsledky shrnuje tabulka 8.3. Výsledky byly naměřeny dne 19. dubna 2014.

Hashtag		autumn	fall	spring	summer	winter
Počet dotazů	**Úroveň 1**	1	1	1	1	1
	Úroveň 2	239	318	261	196	258
	Úroveň 3	17906	23565	21287	16252	18808

Tabulka 8.3: Počet dotazů – roční období, 3 úrovně

Zhodnocení pro 3 úrovně

Tabulka 8.3 ukazuje, jakým způsobem dramaticky narůstá potřebný počet dotazů na programové rozhraní s další úrovní. V průměru je třeba okolo 20000 dotazů, což je o 2 řády více. Z toho vyplývá, že aplikace by se pro takovýto větší soubor neobešla bez neomezeného přístupu k datům skrze Twitter API. Uvedené však není nijak zvlášť překvapující s ohledem na skutečnost, že v praxi se provádí dolování nad daty v řádech GB nebo TB.

Měření pro tabulku 8.3 probíhalo přibližně o týden později oproti prvnímu a jsou tedy patrné drobné změny v počtu dotazů na úrovni 2 u ročních období (průměr 254 dotazů), nicméně se nejedná o nijak zásadní rozdíl.

8.1.2 Počet získaných tweetů a hashtagů na dolovací úlohu

V dalším měření se budeme soustředit na množství získatelných tweetů a hashtagů, které lze z dříve zkoumaného počtu dotazů na programové rozhraní obdržet. Můžeme si tak udělat obrázek, jak moc se s každým dalším dotazem rozrůstá získaná množina dat a také jaké jsou řádově přibližné paměťové nároky na persistenci dat.

Jedná se o stejné dotazování API, k jakému se váží výsledky v tabulkách 8.1 a 8.2 ze dne 13. dubna 2014.

Hashtag	Počet tweetů	Počet hashtagů	Počet jedinečných hashtagů
autumn	17857	77842	16779
fall	19746	82769	17166
spring	21971	98513	21206
summer	20533	87164	19515
winter	16419	77299	15878

Tabulka 8.4: Počet tweetů a hashtagů na dolovací úlohu – roční období

Hashtag	Počet tweetů	Počet hashtagů	Počet jedinečných hashtagů
berlin	16359	70368	14783
bern	14673	59007	12977
brno	11978	51399	12796
brussels	15645	65804	15242
helsinki	10980	40584	10831
oslo	15867	68516	16076
paris	12782	48726	10299
prague	16247	69824	15819
stockholm	11838	47301	11154
wien	9390	35353	9575

Tabulka 8.5: Počet tweetů a hashtagů na dolovací úlohu – evropská města

Zhodnocení počtu tweetů a hashtagů

Z tabulky 8.4 je patrné, kolik tweetů získáme v jednotlivých dolovacích úlohách. V průměru se pak jedná o 19305 tweetů na průměrných 227 dotazů, což značí 85 tweetů na každý dotaz. S ohledem na fakt, že Twitter API vrací vždy nejvýše 100 tweetů na jeden dotaz je vidět, že je kapacita využívána zhruba z 85 %. Zbylých 15 % připadá na tweety, které jsou duplicitní a na dotazy, ke kterým Twitter nenašel ani 100 tweetů. V případě měst se jedná o 13576 tweetů na 180 dotazů v průměru, ve výsledku tedy z jednoho dotazu vzejde 75,5 tweetu.

Počet hashtagů dává u ročních období průměr 84717, v jednom tweetu je tak obsaženo 4,4 hashtagů. Pokud bychom se však zajímali o unikátní hashtagy v rámci dolovací úlohy, dostáváme se k průměrné hodnotě 18109 hashtagů, tedy jedinečných je 0,93 hashtagů na jeden tweet.

U evropských měst je průměrný počet všech hashtagů 55688, což dává 4,1 hashtagů na každý tweet získaný přes programové rozhraní. Jedinečných hodnot se mezi nimi nachází v průměru 12955 na každou dolovací úlohu, tedy 0,95 hashtagů na tweet. Ač tedy evropská města mají menší výtěžnost na jeden dotaz, jsou uživatelé o něco kreativnější při tweetování o nich a dokáží vytvořit pestřejší sociální síť.

V příloze A jsou uvedeny SQL dotazy pro MSSQL server, pomocí nichž lze dojít k výsledkům uvedeným v tabulkách výše.

8.2 Parametry shluků – počet jader

Shluk vzniklý dolováním v získaných datech je charakterizován parametry ϵ a *min neighbours*. Jen pro zopakování dodejme, že první určuje, kolik společných výskytů musejí každé dva hashtagy mít, aby byly uvažovány v dolování, a druhý, s kolika různými jinými hashtagy musí být každý hashtag provázán, aby byl uvažován jako jádro (a jeho okolí bylo dále vyšetřováno).

V následující části si ukážeme, jaký je vliv hodnoty jednotlivých parametrů na početnost jader ve shluku. Za výchozí hodnoty zvolme ϵ rovno 12 a pokládejme *min neighbours* v intervalu 3 až 12.

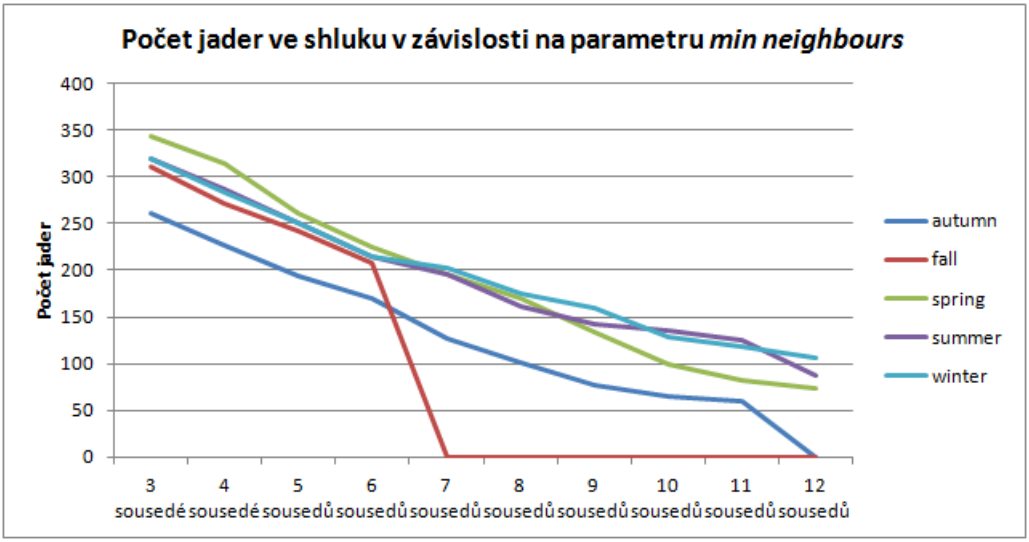
Hashtag	**autumn**	**fall**	**spring**	**summer**	**winter**
3 sousedé	261	310	344	320	320
4 sousedé	226	271	315	286	284
5 sousedů	194	242	261	250	250
6 sousedů	169	208	225	215	214
7 sousedů	127	0	195	195	202
8 sousedů	101	0	169	161	175
9 sousedů	77	0	133	142	159
10 sousedů	65	0	99	135	129
11 sousedů	60	0	82	126	118
12 sousedů	0	0	74	87	106

Tabulka 8.6: Počet jader ve shluku pro $\epsilon = 12$

Zopakujme uvedený postup pro konstantní hodnotu *min neighbours* rovnu 6 a ϵ rostoucí od 6 do 15. Měření zaznamenává tabulka 8.7.

Zhodnocení velikosti shluků

Při sledování velikosti shluku vidíme, že s přibývající hodnotou parametru ϵ či *min neighbours* se shluk zmenšuje lineárně. Při překročení jisté hranice je však velikost shluku najednou nulová. Tento jev je způsoben skutečností, že shluky jsou prozkoumávány a rozšiřovány od iniciálního hashtagu. Pokud již při jeho vyšetřování není splněna hodnota některého z parametrů, shluk dále neroste nezávisle na zbylých jádrech.



Obrázek 8.1: Počet jader ve shluku v závislosti na *min neighbours*

Hashtag	**autumn**	**fall**	**spring**	**summer**	**winter**
$\epsilon = 6$	489	553	572	480	582
$\epsilon = 7$	408	440	455	414	476
$\epsilon = 8$	351	347	398	348	407
$\epsilon = 9$	268	305	332	316	355
$\epsilon = 10$	230	271	280	264	292
$\epsilon = 11$	194	235	238	232	229
$\epsilon = 12$	169	208	225	215	214
$\epsilon = 13$	151	0	207	201	204
$\epsilon = 14$	140	0	165	184	196
$\epsilon = 15$	119	0	132	174	190

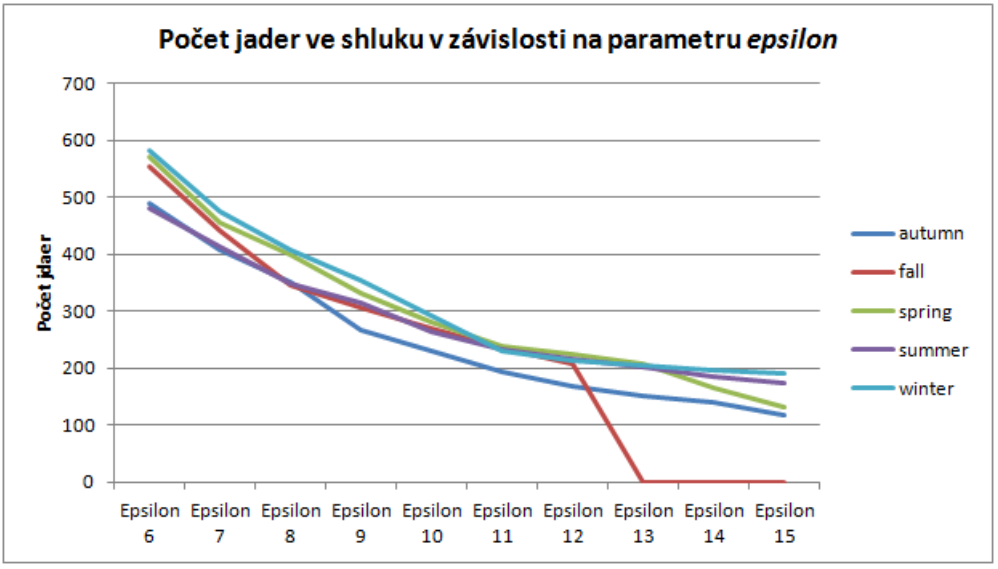
Tabulka 8.7: Počet jader ve shluku pro *min neighbours* = 6

S ohledem na uvedené lze polemizovat, zda je správný přístup prozkoumávat shluk jen od iniciálního hashtagu namísto snahy vytipovat i jiná jádra v rámci shluku a od nich také provést vyšetřování shluku. Účel dolovací metody je však ten, že k zadanému hashtagu hledáme svého druhu synonyma. Není tedy záměrem vyšetřovat okolní jádra, na která iniciální hashtag není dostatečně silně navázán, proto je uvedené chování považováno za žádoucí.

8.3 Rychlost dolování v datech

Pro poskytnutí možnosti srovnání implementace dolovacího algoritmu s případnými jinými implementacemi provedeme otestování časové náročnosti. Její hodnota je důležitá, jak již bylo zmíněno, také pro uživatele, který s aplikací pracuje a interaktivně si prohlíží nadolovaná data.

Využijeme předchozího testu, kdy si ukážeme, jak dlouho trvalo dolování nad daty s proměnnou hodnotou parametru ϵ. Hodnoty uvedené v tabulce 8.8 jsou v sekundách.



Obrázek 8.2: Počet jader ve shluku v závislosti na ϵ

Hashtag	autumn	fall	spring	summer	winter
$\epsilon = 6$	2,061	2,088	3,059	2,081	2,096
$\epsilon = 7$	2,033	2,029	2,097	2,057	2,065
$\epsilon = 8$	2,006	2,005	2,068	2,052	2,044
$\epsilon = 9$	1,088	2,028	2,039	2,014	2,023
$\epsilon = 10$	1,054	1,074	2,016	1,088	2,001
$\epsilon = 11$	1,045	1,059	2,001	1,069	1,070
$\epsilon = 12$	1,030	1,045	1,085	1,060	1,061
$\epsilon = 13$	1,033	0,000	1,071	1,047	1,054
$\epsilon = 14$	1,013	0,000	1,069	1,042	1,046
$\epsilon = 15$	1,006	0,000	1,047	1,049	1,043

Tabulka 8.8: Čas dolování v sekundách

Zhodnocení rychlosti

Na naměřené hodnoty je třeba pohlížet pouze jako na orientační, neboť jsou závislé na konkrétním hardware (*Intel Core 2 Duo, 2.0GHz, 2 jádra, klasický pevný disk*). Poskytují však informaci, že se jedná řádově o jednotky sekund pro vydolování požadované informace.

Průměrný čas dolování dosahuje hodnoty 1,446 sekundy (bez započtení nulových hodnot). Pro zájemce poskytněme informaci, že bez patřičných indexů (a materializovaného pohledu) rozebíraných v kapitole o implementaci se dolování pohybuje v řádu minuty a více.

Kapitola 9

Praktické použití

V následující části věnujme pozornost příkladům prakticky použitelných výsledků, které lze prozkoumáváním shluků získat. Na několika dolovacích úlohách znázorníme, jaké další zajímavé hashtagy lze objevit a jak by je v praxi šlo uplatnit při snaze přilákat uživatele Twitteru ke svým příspěvkům.

9.1 Příklad 1

Začněme hastagem z kolekce reprezentující evropská města – Prahou. Na iniciální hashtag *Prague* obdržíme při následném prozkoumávání shluku s parametry $\epsilon = 4$ a *min neighbours* = 5 celkem 54 přímých sousedů.

Velice silnou vazbu lze pozorovat na hashtag *Praha*, tedy český ekvivalent slova. Dále je silně zastoupeno *party, vampire, czech, Vltava, travel, Europe* či *city*. Pokud zůstaneme nyní jen u těcho přímých sousedů, můžeme ve snaze dělat Praze účinnější reklamu na Twitteru psát tweety v duchu: „Nejlepší #party nabízí jedině #Praha," nebo: „Wanna #party? #Travel to the #city of #Prague, the heart of #Europe."

Nyní se zkusme blíže podívat také na nepřímé vazby. Například skrze *party* je vidět silná vazba na slova *fun, friends, night, instafun, love* či v poslední době oblíbené slovo *selfie*. Z nich lze opět vymyslet příspěvky jako: „#Love having #fun with your #fiends? Visit #Prague."

9.2 Příklad 2

Dalším z evropských měst, které může zaujmout nejednoho cestovatele je Paříž. Na hashtag *Paris* se lze dočíst o sousedních hashtazích *France, Rihanna, summer2014, Seine, french, Eiffel* nebo *streetart*. Se všemi uvedenými pojí *Paris* desítky společných výskytů v získaných tweetech pro dolování, lze je tedy považovat za silně relevantní.

Z uvedeného výčtu lze dojít nápadů jako: „Don't know where to spend your #summer2014 yet? Wanna see #Seine from #Eiffel? Come to #Paris," či „The most amazing #french #streetart can be found only in #Paris."

9.3 Příklad 3

Poslední evropskou destinací, které se zde budeme ještě věnovat, je hlavní město Belgie. Na zadaný iniciální tag *Brussels* nám dolovací nástroj vrátí informaci o silných vaz-

bách na slova *Belgium, art, painting, iloveart*, ale také nepřímo provázané tagy *graffiti* nebo *Banksy* (pseudonym umělce zaměřujícího se na graffiti pocházejícího z Velké Británie).

Pokud nemáme výhrady proti pouličním umělcům, můžeme Brusel zviditelňovat na sociální síti například příspěvkem: „#Iloveart and #graffiti. That's why #Brussels is my choice," nebo „Love to see #Banksy in action? See his #painting in #Belgium, the home of #art."

9.4 Shrnutí

Cílem bylo na několika krátkých příkladech znázornit možnosti využití nadolovaných informací ze sociální sítě. Každý má pochopitelně jiné preference a může tedy dávat přednost odlišné oblasti než evropským městům, na nichž jsou příklady demonstrovány. Někdo také dává přednost příspěvkům s menším množstvím hashtagů, jiný naopak preferuje více. Není známo, že by Twitter nějak zvýhodňoval nebo naopak penalizoval tweety obsahující například více než 2 až 3 hashtagy, proto v jejich použití nebudeme činit žádná doporučení.

Kapitola 10

Závěr

Seznámili jsme se s významem dolování dat a rozebrali principy uplatnění v prostředí sociálních sítí. Rozebrán je celý dolovací proces od počátečního zisku dat až k transformaci a integraci. Zabývali jsme se konkrétními sociálními sítěmi, které mají své nezastupitelné místo v prostředí Internetu a nabízí své služby také českým uživatelům. Jsou diskutovány přednosti i nevýhody vybraných sítí, přičemž sociální síť Twitter je zvolena jako nejvhodnější pro účely dolování.

Vhodnou pro zisk informací se jeví adaptace dolovací metody DBSCAN pracující s hustotou objektů v prostoru. Zmíněna je taktéž její optimalizace v podobě algoritmu OPTICS. Na základě navržené dolovací metody se následně odvíjí realizace implementace, která zohledňuje zmíněnou myšlenku optimalizace.

Celá aplikace je založena na návrhovém vzoru Model-View-Controller, jehož základním principům se práce taktéž věnuje. Po obeznámení s jeho podstatou se v kapitole věnované implementaci rozvádí, jak je celý nástroj do něj zasazen. K dispozici je podrobný postup, jak nástroj vlastními silami naimplementovat a na co je žádoucí si dát při realizaci pozor. Prostor je věnován také návrhu relační databáze pro persistenci dat získaných z programového rozhraní, neboť vhodné uložení dat má zásadní vliv na výkonnost dolovacího procesu.

V závěru práce se lze dočíst o praktickém uplatnění získaných znalostí na příkladech vybraných evropských měst. Cílem je ukázat, jak lze nástroj použít pro jejich zviditelnění na sociální síti pomocí vhodně použitých hashtagů v různých příspěvcích.

10.1 Možná rozšíření

Aplikace se inspirovala rozšířením OPTICS, které bylo zahrnuto do implementace. Nadále však zůstává prostor pro další zlepšení, jako je rotování uživatelských účtů při dotazování API sociální sítě za účelem většího možného počtu dotazů za jednotku času. S vyšším limitem počtu dotazů nabývá na významu také případná podpora distribuovaného získávání dat z Twitteru v rámci jedné dolovací úlohy nebo výpočet shluku v nadolovaných datech ve více výpočetních vláknech. Jejich práce by spočívala v postupném synchronizovaném odebírání hashtagů ze seznamu dosud nevyšetřených a paralelním dotazování relační databáze, což by přineslo urychlení práce uživatele při čekání na výpočet shluku.

Literatura

[1] Acquire Media: Facebook Reports First Quarter 2013 Results [online]. `http://investor.fb.com/releasedetail.cfm?ReleaseID=761090`, 2013-05-01 [cit. 2013-10-24].

[2] Alexis Sellier: An overview of Less, how to download and use, examples and more [online]. `http://lesscss.org`, 2009-03 [cit. 2013-04-06].

[3] Ankerst, M.; Breunig, M. M.; Kriegel, H.-P.; aj.: OPTICS: ordering points to identify the clustering structure. *ACM SIGMOD Record*, ročník 28, č. 2, 1999: s. 49–60.

[4] Codd, E. F.: *The relational model for database management: Version 2*. Boston: Addison-Wesley Longman Publishing Co., 1990, ISBN 0-201-14192-2, 538 s.

[5] eBiz MBA: Top 15 Most Popular Social Networking Sites [online]. `http://www.ebizmba.com/articles/social-networking-websites`, 2013-10 [cit. 2013-10-25].

[6] Ester, M.; Kriegel, H.-P.; Sander, J.; aj.: A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, ročník 96, 1996, s. 226–231.

[7] Facebook: The Graph API [online]. `https://developers.facebook.com/docs/getting-started/graphapi/`, 2013-09-01 [cit. 2013-10-24].

[8] Google: Authorization and Google+ APIs [online]. `https://developers.google.com/+/api/oauth`, 2013-08-16 [cit. 2013-10-25].

[9] Google: Downloads [online]. `https://developers.google.com/+/downloads/`, 2013-10-11 [cit. 2013-10-25].

[10] Gundotra, V.: Google+: Communities and photos [online]. `http://googleblog.blogspot.cz/2012/12/google-communities-and-photos.html`, 2012-12-06 [cit. 2013-10-25].

[11] Han, J.; Kamber, M.: *Data mining: Concepts and techniques*. San Francisco: Morgan Kaufmann Publishers, druhé vydání, 2006, ISBN 978-1-55860-901-3, 770 s.

[12] LinkedIn: Authentication [online]. `http://developer.linkedin.com/documents/authentication#granting`, 2013 [cit. 2013-10-25].

[13] LinkedIn: Libraries and Tools [online].
http://developer.linkedin.com/documents/libraries-and-tools, 2013 [cit. 2013-10-25].

[14] Lunden, I.: Twitter May Have 500M+ Users But Only 170M Are Active, 75Twitter's Own Clients [online].
http://techcrunch.com/2012/07/31/twitter-may-have-500m-users-but-only-170m-are-active-75-on-twitters-own-clients/, 2012-07-31 [cit. 2013-10-24].

[15] Mayo, J.: Celebrating #Twitter7 [online]. http://linqtotwitter.codeplex.com/, 2013-09-23 [cit. 2013-10-24].

[16] Microsoft Corporation: ASP.NET MVC Overview [online].
http://www.asp.net/mvc/tutorials/older-versions/overview/asp-net-mvc-overview, 2009-01-27 [cit. 2013-03-29].

[17] Microsoft Corporation: Understanding the ASP.NET MVC Execution Process [online].
http://www.asp.net/mvc/tutorials/older-versions/overview/understanding-the-asp-net-mvc-execution-process, 2009-01-27 [cit. 2013-03-29].

[18] Microsoft Corporation: Extension Methods (C# Programming Guide) [online].
http://msdn.microsoft.com/en-us/library/bb383977.aspx, 2009-04 [cit. 2013-04-12].

[19] Nishar, D.: 200 Million Members! [online].
http://blog.linkedin.com/2013/01/09/linkedin-200-million/, 2013-01-09 [cit. 2013-10-25].

[20] Olanoff, D.:
For the last time, let's all say it together: "Google+ is NOT a Social Network" [online].
http://thenextweb.com/socialmedia/2012/03/08/for-the-last-time-lets-all-say-it-together-google-is-not-a-social-network/, 2012-03-08 [cit. 2013-10-25].

[21] Twitter: REST API Rate Limiting in v1.1 [online].
https://dev.twitter.com/docs/rate-limiting/1.1, 2013-03-15 [cit. 2013-10-24].

[22] Twitter: Twitter Certified Partners [online].
https://business.twitter.com/twitter-certified-partners, 2013 [cit. 2013-10-24].

[23] Watkins, T.: Suddenly, Google Plus Is Outpacing Twitter To Become The World's Second Largest Social Network [online].
http://www.businessinsider.com/google-plus-is-outpacing-twitter-2013-5, 2013-05-01 [cit. 2013-10-25].

[24] Wickre, K.: Celebrating #Twitter7 [online].
https://blog.twitter.com/2013/celebrating-twitter7, 2013-03-21 [cit. 2013-10-24].

[25] Zendulka, J.; Bartík, V.; Lukáš, R.; aj.: *Získávání znalostí z databází: Studijní opora.* Brno: Vysoké učení technické, 2009, 160 s.

Příloha A

SQL dotazy

Příloha obsahuje příkazy jazyka SQL pro získání tabulek 8.4 a 8.5 z kapitoly věnované testování.

A.1 Počet tweetů na dolovací úlohu

```
WITH Statuses_CTE ([MiningTaskID], [Hashtag], [CreatedOn], [TwStatusID])
AS
(
  SELECT DISTINCT [MiningTaskID], [Hashtag], [CreatedOn], [TwStatusID]
  FROM [miner].[dbo].[TwitterStatus_TwitterStatusHashtag]
  JOIN [miner].[dbo].[MiningTask] ON [MiningTaskID] = ID
  WHERE [State] = N'Finished'
)

SELECT [MiningTaskID], [Hashtag], [CreatedOn],
  COUNT([TwStatusID]) as [TwStatusIDCount]
FROM [Statuses_CTE]
GROUP BY [MiningTaskID], [Hashtag], [CreatedOn]
ORDER BY [Hashtag]
```

A.2 Počet hashtagů na dolovací úlohu

```
WITH StatusHashtags_CTE ([MiningTaskID], [Hashtag], [CreatedOn],
  [TwStatusHashtagID])
AS
(
  SELECT DISTINCT [MiningTaskID], [Hashtag], [CreatedOn],
    [TwStatusHashtagID]
  FROM [miner].[dbo].[TwitterStatus_TwitterStatusHashtag]
  JOIN [miner].[dbo].[MiningTask] ON [MiningTaskID] = ID
  WHERE [State] = N'Finished'
)

SELECT [MiningTaskID], [Hashtag], [CreatedOn],
```

```
  COUNT([TwStatusHashtagID]) as [TwStatusHashtagIDCount]
FROM StatusHashtags_CTE
GROUP BY [MiningTaskID], [Hashtag], [CreatedOn]
ORDER BY [Hashtag]
```

A.3 Počet jedinečných hashtagů na dolovací úlohu

```
WITH StatusUniqueHashtags_CTE ([MiningTaskID], [Hashtag], [CreatedOn],
  [TwStatusHashtag])
AS
(
  SELECT DISTINCT [MiningTaskID], [Hashtag], [CreatedOn],
    [TwStatusHashtag]
  FROM [miner].[dbo].[TwitterStatus_TwitterStatusHashtag]
  JOIN [miner].[dbo].[MiningTask] ON [MiningTaskID] = ID
  WHERE [State] = N'Finished'
)

SELECT [MiningTaskID], [Hashtag], [CreatedOn],
  COUNT([TwStatusHashtag]) as [TwStatusUniqueHashtagCount]
FROM StatusUniqueHashtags_CTE
GROUP BY [MiningTaskID], [Hashtag], [CreatedOn]
ORDER BY [Hashtag]
```