Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Extrakce sociálních sítí ze zpravodajských textů

Plzeň, 2014 Lukáš Witz

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 7. května 2014

Lukáš Witz

Abstract

Social Network Extraction from News

This bachelor's thesis deals with an analytic application usable for social network extraction from preprocessed news articles, and today's trends in graph visualization. It also describes graph algorithms and selected problems being solved by Natural Language Processing.

In the second part, the developed application called SoNEx is introduced. This application is intended to analyze news articles and vizualize obtained results. Using of particular algorithms in this case is also explained. The last part presents the results that were achieved and compares them with expectations.

Abstrakt

Extrakce sociálních sítí ze zpravodajských textů

Tato práce se zabývá analýzou novinových článků předzpracovaných pomocí metod z oblasti zpracování přirozeného jazyka. Jejím tématem je extrakce sociální sítě založené na pojmenovaných entitách nacházejících se ve vstupních datech. Dalším předmětem zájmu jsou možnosti její následné vizualizace, které poskytují moderní nástroje pro vykreslování grafů a práci s nimi. Populární a často používané knihovny a frameworky jsou porovnány a na jednoduchých příkladech jsou ukázány jejich výstupy. Je též předvedena základní práce s knihovnou použitou při implementaci výsledné aplikace. Ve stručnosti jsou ukázány metody pro práci s grafy a základní grafové algoritmy.

Poděkování

Rád bych poděkoval především svému vedoucímu práce doc. Ing. Josefu Steinbergerovi, Ph.D. za jeho ochotu, nekonečnou trpělivost a výborné rady. Díky patří i mé rodině za podporu při studiu a Kateřině Skryjové za korekturu

Obsah

1 Úvod

Pro období posledních 40 let se vžilo pojmenování informační věk. Informace se začaly hromadit a šířit překotným tempem, které stále narůstá. Co naopak rychle klesá, je cena za jejich přenos a uchovávání. S rostoucím množstvím přišla i potřeba třídit je a zpracovávat. Ekonomicky výhodné je však jejich hromadění pouze v případě, že s nimi lze dále nakládat a získávat z nich znalosti. Vyhledávat v takovém množství dat je nad lidské síly a zabývat se pouze dílčími daty znamená ztrátu globálního pohledu a kontextu, a tím i znepřesnění výsledků. Především proto je nezbytně nutná automatizace těchto procesů, která poskytne uživateli všechny informace sumarizované, v přijatelné a pochopitelné podobě a především ty, o které žádá.

Práce se zabývá získáváním znalostí z množiny prostých textů a jejich zprostředkováním a klade si za cíl provést analýzu předzpracovaných zpravodajských textů, jejímž výstupem bude sociální síť pojmenovaných entit.

Navazující částí bude vytvoření grafické reprezentace vygenerované struktury. Vizualizaci obdobného charakteru využívají projekty, jako je například obchodní rejstřík, pro zobrazení relací účastnických entit, což umožní velmi rychle a intuitivně získat pro firmy i osoby cenné informace a znalosti za vyvinutí minimálního úsilí. Použitelnost závisí na způsobu a efektivitě zobrazení dat. Úkolem tak nebude pouze vymyslet způsob jejich analýzy, ale také prozkoumat možnosti generování jejich vizuální reprezentace.

2 Analýza textu

2.1 Základní charakteristika NLP

Tato práce spojuje výsledky získané pomocí NLP s vizuální reprezentací dat. *Termín NLP* (z anglického Natural Language Processing, Zpracování přirozeného jazyka) *se obvykle používá pro popis funkce softwarových či hardwarových komponent počítačového systému, který analyzuje či syntetizuje mluvenou nebo psanou řeč. Slovo „přirozený“ zdůrazňuje, že jazyk běžné komunikace se odlišuje od formálnějších jazyků, jako je matematický zápis či programovací jazyky, kde jsou slovník a syntaxe značně omezeny* (definice přeložena z [1]). NLP lze využít především v oblastech jako jsou sumarizace textu, strojový překlad, umělá inteligence a rozpoznávání řeči. Svoje uplatnění najde i při vyhledávání informací ve vícejazyčných zdrojích.

2.2 Úlohy NLP

Každý jazyk má svá specifika, což velmi ztěžuje vytvoření univerzálního nástroje pro analýzu, neboť ta pro každou řeč a její zápis vyžaduje jiný přístup (velmi univerzální jsou pak samozřejmě metody, které jsou založeny na předem daných slovnících, jež stačí při změně jazyka vstupních dat pouze vyměnit za jiné, pokud jsou takto obecně implementovány). Těžko budeme moci bez rozsáhlých úprav použít pro angličtinu vyvinutý program k rozboru textu psaného v češtině, tím spíše textu například korejského původu. Úlohy stejného typu tak pro každý jazyk vyžadují odlišný postup. Některé problémy řešené pro český jazyk jsou nastíněny níže. Jsou zde popsány hlavně ty oblasti NLP, které se týkají vstupních dat této práce.

2.2.1 Předzpracování dat

Segmentace

Úkolem segmentace je nalezení začátku a konce každého jednotlivého úseku (slova, věty apod.) řetězce znaků tak, aby vznikl seznam smysluplných celků.

Ruční označování textů je ve větším množství velmi pracné a tím i velmi nákladné. Mnohem levnější cestou je automatizace, která ale přináší větší míru chybovosti.

Při implementaci programu určeného k analýze psané češtině narazíme na problémy jako například nalezení konce věty v řetězci typu „Na konferenci vystoupí Ing. Pavel Novák.“, kde jej logicky umístíme až za poslední slovo „Novák“, avšak při sekvenčním průchodu narazíme na první tečku a na následující velké písmeno, což by jako posloupnost jinak ukazovalo na konec jedné a začátek druhé věty. Ještě větším problémem jsou jazyky, kde se tečka (nebo její alternativa) na konci věty nepoužívá.

Jak již bylo řečeno, úkolem segmentace je nalezení slov v řetězci, ve kterém od sebe nejsou specificky oddělena (například URL). Pokud se vrátíme k běžnému psanému textu, velmi běžné jsou i jazyky, kde jsou odděleny pouze celé věty, ale slova se od sebe nikterak nedělí (některé jazyky sou psány formou scriptio continua a nedělí ani slova ani věty), či jsou do slov vkládány speciální znaky.

Pro český jazyk lze použít například metodu založenou na pravidlech. Aby nedocházelo k výše popsanému problému („Ing. Pavel Novák“), lze přidat navíc katalog zkratek, podle nějž můžeme zjistit, že token „Ing“ většinou nepředchází tečce, která ukončuje větu. Při použití na anglické texty se úspěšnost pohybuje okolo 95 % [2].

Jiným způsobem, který se často používá, je učení programu pomocí dat, kde jsou konce vět ručně označeny. Automat se tak naučí, pravidla říkající která jsou slova (skupiny slov, písmen), za kterými se často vyskytuje tečka, kterou věta nekončí. Tato metoda dává přesnější výsledky (pro angličtinu až 99,5 %[3]).

Tokenizace

Úkolem, který na segmentaci navazuje, je tokenizace. Ta řetězec rozdělí na jednotlivá významová slova či fráze nazývané tokeny. Problémem je například určování čísel, jež se mohou vyskytovat v různých podobách s různými oddělovači a je zapotřebí je vždy správně rozeznat (123 456 nebo 123,456 nebo 123456, stejně tak 1.2 nebo 1,2). Jako další příklady lze zmínit konkatenace slov přes pomlčku (červeno-bílé dresy), různé formy zápisu datumů a v neposlední řadě stažené tvary, které se hojně využívají spíše v cizích jazycích.

Jedním možným způsobem, jak jednotlivé tokeny hledat, je považovat za samostatný token každý řetězec obklopený mezerami (a samozřejmě zbavený interpunkčních znamének). Problémem tohoto přístupu je, že vznikají celky, které samostatně nemají význam. Například spojení *de facto*, *ad hoc*, ale také *a tak*, *i kdyby* a podobně. Navíc dochází k rozdělení několikaslovných názvů (*Ministerstvo práce a sociálních věcí*).

Možností, jak se takovému problému vyhnout, je použití seznamu výjimek aplikovatelného již při tokenizaci, nebo rozpoznávat taková spojení až dodatečně a označovat je speciálními značkami.

Podobnou otázkou je i to, zda posuzovat předložku a slovo po ní jdoucí jako jeden token. Výhodou může být usnadnění další analýzy. Předložka většinou indikuje pád slova po něm následujícího. Informaci o pádu lze pak s výhodou použít pro převedení slova do jeho prvního pádu.

Lemmatizace a stemming

Čeština používá, jakožto jazyk mající ohebné slovní druhy, mnoho slov odvozených od společného základu (skloňování, rody, časování...). Pro účely analýzy textu (a obzvláště při vyhledávání relací mezi pojmenovanými entitami, které je náplní této práce) je však nepřijatelné rozlišovat mezi entitami *Pavlu Novákovi* a *Pavel Novák*. Proto je potřeba nalézt společný tvar a identifikátor různých pádů, časů a zkrátka flexí jednoho slova. V českých textech zabývajících se touto problematikou lze často narazit na poněkud zmatenou terminologii co se týče nástrojů pro hledání takového společného řetězce. Velmi často se vychází z odborných článků anglického původu, což zapříčiňuje splývání termínů lemmatizátor a stemmer. Zatímco lemma je definováno jako reprezentativní forma slovníkové jednotky, stem v překladu znamená kmen slova. V angličtině tyto termíny splývají, nebot' tento jazyk v naprosté většině případů tyto pojmy nerozlišuje. Vhodným příkladem rozdílnosti je například tvorba množného čísla, kdy si angličtina ve většině případů vystačí s přidáním písmene *-s* (či koncovky *-es*). V češtině naopak vytvoření množného čísla představuje změnu kmenu (dítě – děti, pes – psi, oko – oči atd.).

Vrat'me se nyní k lemmatizaci a pokusme se tuto metodu blíže popsat. Tento postup využívá slovníkové a morfologické analýzy slov. To znamená, že pokud chceme dosahovat dobrých výsledků, je zapotřebí využít dostatečně obsáhlého slovníku zahrnujícího všechny myslitelné formy daného lemmatu.

Jak bylo výše řečeno, český lemmatizátor musí hledat základní slovníkový tvar a nikoliv kmen slova. Takového tvaru je pak využito například i při fulltextovém vyhledávání na webových stránkách. Zadavatele dotazu zajímají všechny stránky, kde se o klíčových slovech mluví a proto je důležité nebrat v potaz pád a tvar vloženého klíčového slova.

Existují dva hlavní způsoby vytváření nových tvarů slova (resp. nových slov) [5]. První možností je odvodit nové slovo od jiného, pocházejícího ze stejného slovního druhu (různé koncovky sloves pro různé osoby a časy: *chodit – chodí – chodil – chodila* atd.). Nové formy tak vznikají ohýbáním (neboli skloňováním, časováním). Druhým způsobem vzniku je odvozování, kde nové slovo vychází ze slova jiného slovního druhu (jako například zpodstatnělá přídavná jména nebo slovesa: *chůze – chození – chodit*). U obou těchto způsobů je vznik nových nových slov zařízen aplikováním jistých pravidel. Vždy se jedná o odebírání a přidávání předpon a přípon. Lemmatizaci lze tak považovat za inverzní aplikaci těchto pravidel.

Nutnou součástí lemmatizátoru je proto nejen slovník, ale i soubor pravidel, která budou aplikována na lemmatizovaná slova. Vždy se tedy aplikuje jedno z pravidel popisující změnu přípony (koncovky) a následně je nutno ověřit, zda se nově vytvořený tvar vyskytuje ve slovníku jako základní tvar a pokud ano, zda dané reverzní aplikace takového pravidla poskytuje původní tvar slova.

Jinou možností nalezení jednotného tvaru je zmíněný stemming (v češtině někdy nazývaný stemmatizace). Ten místo slovníku používá heuristiku, takže není zapotřebí sestavovat velké slovníky jako v předchozím případě. Pomocí heuristiky metoda nalezne příponu, kterou odstraní a, pokud je to nutné, doplní zbytek slova tak, aby byl získán jeho správný tvar. Všechna slova se stejným kořenem (nebo lépe řečeno kmenem) jsou převedena na stejný tvar. Nevýhodou tohoto přístupu je ztráta gramatického kontextu. Navíc odtržení posloupnosti znaků, která je posouzena jako přípona, může vyústit v nesmyslný tvar, který není slovotvorným základem. Podle [4] jsou dvěma hlavními principy oddělování přípon iterace a nejdelší shoda. Obě metody trpí specifickými neduhy, které se dají odstranit jejich kombinací.

Metoda iterace je založena na myšlence, že na kořen se přípony napojují postupně v určitém pořadí, což znamená, že lze rozlišit typy přípon. V češtině lze očekávat, že posledním suffixem sloves tak například budou koncovky. Při tvorbě kmenu slova jsou tak postupně odzadu odstraňovány jednotlivé podřetězce, až zbude pouze hledaný stem.

Metoda nejdelší shody používá seznam vygenerovaných vzorů (což je vlastně kombinace všech přípustných přípon). Každé slovo je pak porovnáno s těmito vzory, které jsou seřazeny od nejdelšího k nejkratšímu. Odstraněna je vždy první přípona, s níž nastane shoda. Nevýhoda tohoto postupu je nasnadě. Potřebuje nejprve vygenerovat všechny kombinace a mít je po celou dobu analýzy uloženy v paměti.

2.2.2 Rozpoznávání pojmenovaných entit

Další oblastí z oboru NLP, která bezprostředně souvisí s touto prací a která je pro ni v podstatě nejdůležitější, je Named entity recognition (NER), česky Rozpoznávání pojmenovaných entit. Pojmenovaná entita je slovo či sousloví, které jednoznačně identifikuje například osobu, společnost, geografický celek, politické uskupení, produkt apod.

Rozpoznání takového výrazu v textu lze provádět několika způsoby. Prvním přístupem je slovníková metoda. Pokud správně určíme lemma daného úseku textu, lze jej snadno porovnat s entitami obsaženými ve slovníku. Takto lze získat velmi přesné výsledky (tato metoda neidentifikuje jako pojmenovanou entitu výraz, který se ve slovníku nenachází, neboli bude mít málo falešně pozitivních výsledků), avšak podobně jako u metody lemmatizace zde narazíme na problém vytvoření dostačujícího slovníku pro identifikaci všech pojmenovaných entit v textu. Těžko vytvoříme katalog obsahující jména všech světových jmen, měst, řek apod. Jediným nástrojem, který by tak mohl odstínit tvorbu slovníku, je encyklopedie podobného typu jako projekt Wikipedia, který by měl navíc tu výhodu, že by umožňoval velmi snadno spojit lingvisticky nesouvisející ale logicky totožné entity jako například „Miloš Zeman“ a „prezident České republiky“, neboť informace o jejich relaci je již zaznamenána. V tomto ohledu lze poukázat na existující projekt jménem DBpedia.org

Jiným přístupem může být použití strojového učení (machine learning), které používá k analýze tzv. klasifikační model, který předem naučíme za použití korpusových textů, jaké řetězce si má uložit do kolekce výsledků, neboli databáze znalostí. Rozlišují se dva způsoby učení klasifikátoru. *Učení s učitelem* spočívající v ručním označení textu správnými tagy (vytvoření korpusu) a *učení bez učitele*, kde se používá sestavených pravidel. Učením s učitelem získáváme pak funkci reprezentovanou rozhodovacím stromem nebo pravděpodobnostním modelem, jejímž vstupem je řetězec znaků a výstupem klasi-

fikační hodnota.

Při učení bez učitele pak nemáme žádná trénovací data, ale pouze data testovací. Cílem je vytvořit vzory, podle kterých se později budou entity rozpoznávat. Na základě schodných vlastností jsou entity shlukovány do klastrů, které jsou již jen označeny typem pojmenovaných entit. Shluky jsou tvořeny pomocí shodných rysů zahrnutých entit, které dopředu definujeme. Pokud tak před slovem začínajícím velkým písmenem najdeme slova jako *pan*, *Mgr.* a podobně, lze předpokládat, že se jedná o vlastní jméno. Stejně tak pokud začíná název zkratkou typu *SK*, *AC*, *HC*, lze usuzovat, že se jedná o sportovní kluby. Výstupem vytvořené funkce, které zadáme prostý text, je v tomto případě klastr, do kterého entita patří.

Další součástí potřebnou k učení klasifikátoru je nástroj, který mu bude dodávat data ke zpracování. Většinou je používán sekvenční přístup. Proto je zapotřebí částečného parsování, například často používané metody zvané *chunking* (kouskování, viz [8]).

V průběhu učení vzniká znalostní databáze, která je většinou založena na statistických předpokladech. Například v kolika procentech případů začíná pojmenovaná entita velkým písmenem (tzv. rys slova) apod. Stejně tak musí obsahovat informace o víceslovných entitách, která může být doplněna slovníkem entit.

Ve chvíli, kdy již máme pojmenovanou entitu rozpoznánu, nastává problém jejího zařazení. Mějme například slovo Mars. Čtenář z kontextu snadno rozezná, zda se jedná o planetu, výrobce cukrovinek, nebo osobní jméno, avšak ze samotného slova jeho kategorii nelze určit. Proto při strojovém určení musí být vhodně zvolen kontext tak, aby byl sice dostatečně velký, ale zároveň se nesmí informace ztratit v záplavě nepodstatných dat.

2.2.3 Analýza polarity názoru

Původní anglický termín *sentiment analysis* se do češtiny překládá dosti nepřesně jako *analýza sentimentu*, přestože slovo sentiment má v původním významu trochu jiný smysl. V našem jazyce ale těžko najdeme přesný překlad. Nejblíže se původnímu významu blíží polarita názoru. Pojem jako takový pochází z oblasti burz a akciových trhů. Jde o odhad postoje, nálady skupiny či jednotlivce ke konkrétnímu produktu, osobě, jiné skupině a tak podobně.

Rozmach tohoto odvětví nastal s příchodem sociálních sítí, diskuzních fór, ale i běžných jednoduchých komentářů na stránkách výrobců a poskytovatelů služeb. Na internetu na takovýchto portálech získáme mnohem větší vzorek uživatelů než pokud bychom nechali provádět nákladné průzkumy veřejného mínění používané pro cílený marketing, což samozřejmě firmám ušetří nemalé peníze.

Problémem je ale anonymita lidí na internetu. Pokud do diskusí vstupují pouze pod přezdívkou bez dodatečných informací, můžeme sice určit globální postoj uživatelů, posluchačů atd., ale nelze v takovém případě mluvit o cílových skupinách. Velmi jednoduchým řešením je nutnost registrace před vstupem, kde ale lidi mohou poskytnout nepravdivé údaje. Firmy dobře vědí, že pro člověka mnohem pohodlnějším způsobem je přihlašování pomocí již existujícího účtu a velmi často poskytují přihlašování pomocí Twitteru a Facebooku. Stačí jedno kliknutí pro potvrzení přístupu k účtu a uživateli je možno diskutovat. Diskutuje přihlášen účtem, který (pokud nemá nastaveno velmi vysoké zabezpečení) obsahuje všechny informace, které jsou pro řádný výzkum potřeba (věk, místo pobytu) a navíc nám napoví, jaké jsou zájmy daného člověka atd. Navíc můžeme zpracovat statistiku zahrnující jeho přátelé (zda jsou stejná cílová skupina a zda daný produkt/firmu/rádio apod. znají).

Z výše uvedeného plyne, že analýza sentimentu je oblast, v níž je stále co rozvíjet a která má velký potenciál. Posledním krokem této práce tak bude vyzkoušet některý z existujících nástrojů pro určení sentimentu příslušejícímu vztahu mezi nalezenými entitami.

Určovat hodnotu sentimentu lze mnoha způsoby, mezi lze nimiž nejčastěji narazit na slovníkovou metodu, naivní Bayesův klasifikátor, metodu maximální entropie a Support Vector Machine. Nejprve se zaměříme na **slovníkovou metodu**.

Analýza obecně probíhá následovně: text je nejprve rozdělen na n-gramy, neboli sled po sobě jdoucích slov. K určení, neboli kvantifikování polarity se používá několik metod, které všechny označují tyto části pomocí čísel, znamenajících hodnotu sentimentu daného n-gramu, a pomocí značek, které představují zeslabení nebo zesílení ohodnocení následujícího n-gramu.

Triviální možnost, jak následně určit celkovou polaritu, je sekvenční projití ohodnocených unigramů v textu, při kterém se negativní ignorují, zesilující a zeslabující vynásobí následující hodnotu předem daným koefici-

entem a negující obrátí hodnotu následujícího unigramu. Výsledek je pak dán průměrem všech nenulových hodnot. Možnou úpravou je provádět popsané operace nad bigramy a trigramy, což vede ke zpřesnění výsledků při malém počtu neutrálních n-gramů. Nejlepších výsledků dosahuje pro texty obsahujících silně pozitivní a silně negativní n-gramy, naopak velmi často jsou špatně vyhodnoceny výrazy obsahující převážně neutrální části za pozitivní či negativní.

Vylepšení přináší **průměrová metoda**, která bere v potaz i počet, resp. podíl neutrálních výrazů, kterým se násobí aritmetický průměr získaný triviální metodou. Lze tak dosáhnout zjemnění a utlumení chyby popsané výše. Naopak zhoršení výsledku se dosáhne pro sekvence se nízkou hodnotou sentimentu.

Poslední úpravou algoritmu je **rozdílová metoda**, která zvýhodňuje trigramy, které jsou vyhledávány a vyhodnocovány dříve, nad unigramy, které jsou vyhodnoceny naposledy. Následuje výpočet polarity všech částí, určí se průměrné hodnoty pozitivních a negativních koeficientů a ty se nakonec vynásobí jejich poměrem vůči celku. Tato metoda se zaměřuje především na nejednoznačné sekvence. Chybovat může tam, kde se společně vyskytují v podobném poměru pozitivní i negativní n-gramy, neboť se vzájemně vyruší a výsledkem je neutrální ohodnocení.

Metody strojového učení jsou založeny na přiřazení části textu do kategorie (pozitivní, negativní, neutrální) na základě předem definovaných vlastností, které jej mohou popisovat. Tyto způsoby analýzy jsou dále přiblíženy tak, jak je definuje [9]. Velmi účinnou metodou používající tohoto postupu je naivní **Bayesův klasifikátor**, který využívá inkrementálního učení.

Každému ucelenému textu d je přiřazena třída $\hat{c}$ takto:$\hat{c} = \arg\max_c \ P(c|d)$, kde c je množina tříd sentimentu. Dále odvodíme podle Bayesova pravidla:

$$P(c|d) = \frac{P(c)P(d|c)}{P(d)},$$

kde $P(c)$ je apriorní pravděpodobnost výskytu třídy c v dokumentu podle natrénovaného korpusu. Odhad pravděpodobnosti $P(d|c)$, je nahrazen rozkladem, který předpokládá nezávislost příznaků dokumentu.

$$P_{NB}(c|d) := \frac{P(c)(\Pi_{i=1}^{m} P(f_i|c)^{n_i d}}{P(d)}$$

Většinou lepší výsledky poskytující metodou je **maximální entropie**.

Výpočet $P(c|d)$ má v tomto případě tuto podobu:

$$P_{ME}(c|d) := \frac{1}{Z(d)} \exp(\sum_i \lambda_i f_i(d,c)),$$

kde $Z(d)$ je normalizační funkce, λ_i je odhadový parametr a $f_i(d|c)$ je příznakový model. Je důležité, že oproti naivnímu Bayesově klasifikátoru nejsou vytvářeny předpoklady o vztazích mezi vlastnostmi, a pokud takové neexistují, dává maximální entropie lepší výsledky. Důležité je vybrat takový model, který má vytvořeny nejmenší předpoklady o datech a přitom stále zůstává s daty konzistentní.

Poslední metoda, které se budeme věnovat, je **Support Vector Machine**, která oproti výše zmíněným umožňuje lepší ohraničení jednotlivých tříd. Na rozdíl od nich se nejedná o statistickou metodu. V případě, že bychom měli 2 třídy, je základní myšlenkou trénování nalezení nadroviny reprezentované vektorem $\vec{w}$, která nejenže oddělí dvě třídy, ale navíc je oddělí tak, že jejich vzdálenost je co největší. Tento úkol je optimalizační úlohou. Pokud budeme mít množinu tříd $c_j \in \{-1, 1\}$ (kladné a záporné), řešení lze zapsat takto:

$$\vec{w} := \sum_j \alpha_j c_j \vec{d_j},\ \alpha_j \geq 0,$$

kde α_j jsou získány vyřešením binárního optimalizačního problému. Vektor $\vec{d_j}$ je nazýván pomocným vektorem a protože $\vec{\alpha_j}$ je vždy větší než 0, určuje jako jediný vektor dokumentu ovlivňující $\vec{w}$. Klasifikace jednoduše rozhodne, které straně nadroviny náleží zpracovávaná instance.

2.3 Grafové vizualizace

2.3.1 Tvorba grafického výstupu

Aby byly informace získané analýzou dat pro uživatele snadno přístupné a použitelné, je zapotřebí zvolit jejich rozumnou reprezentaci tak, aby byla co nejefektivnější a výsledná vizualizace uživatelsky co možná nejpříznivější. To, co se děje v pozadí aplikace, je uživateli skryto a kromě rychlosti zpracování jej tak zajímá hlavně kvalita grafického výstupu.

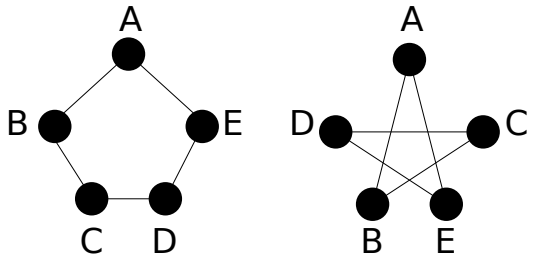
Na místě je srovnat metody, kterými lze dosáhnout nejlépe použitelného výsledku. Zaměřujeme se na efektivitu a způsob reprezentace, kde nás zajímá

výsledný vzhled a rychlost odezvy na uživatele. Následuje stručný výčet možností, které máme pro tvorbu reprezentace grafu, jeho vizualizace a grafové algoritmy, které se používají k procházení grafu. Zadefinujeme také několik základních pojmů z oblasti teorie grafů, které se bezprostředně týkají této práce.

Grafy

Graf jako obecný pojem se vyskytuje v několika významech (diagramy, neboli grafy často používané ve statistice – koláčové, sloupcové – a samozřejmě grafy funkcí). Zde se myslí graf tak, jak jej chápe diskrétní matematika (přesněji tak, jak jej definuje [6]). V celé této práci se pak takto označuje neprázdná množina vrcholů (někdy nazývané uzly) V (z anglického *vertex*) a množina hran E (z anglického *edge*) mezi dvojicemi uzlů, at' už různých, nebo totožných, kde takové hrany nazýváme smyčky. Zapisujeme $G = (V, E)$. Vrcholy představují jednotlivé entity a hrany vztahy mezi nimi.

Vizuální reprezentace grafů se provádí jejich zakreslením do roviny. Totožnou množinu vrcholů $V(G)$ a hran $E(G)$ můžeme zakreslit více než jedním způsobem. Na obrázku 2.1 níže lze vidět, že jeden a ten samý graf lze reprezentovat dvěma zcela odlišnými způsoby pouhým prohozením vrcholů. Záleží pak vždy na účelu použití, ale obecně platí, že se snažíme vyhnout křížení čar (hran), pokud je to možné. Ve schématech popisující elektrické obvody je křížení hran dokonce zcela nepřípustné.

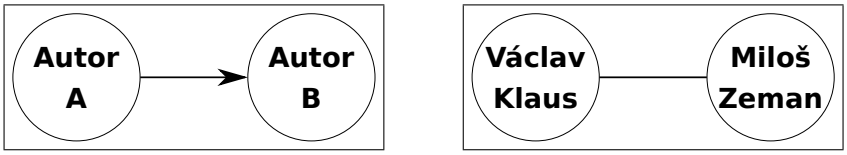


Obrázek 2.1: Různé reprezentace stejného grafu

Hrany mezi vrcholy můžeme rozlišovat orientované a neorientované. **Orientovaný graf** definuje [7] jako dvojici (V, E), kde V je množina vrcholů a $E \subset V \times V$ je množina hran. Hrany tak představují uspořádanou dvojici dvou vrcholů (zobrazení viz 2.2). Pokud by pak uzly například reprezentovaly autory A a B odborných článků a hrany mezi nimi by znamenaly citování

zdroje, hrana AB by pak říkala, že autor A ve svém díle cituje autora B. Naopak hrana BA by znamenala, že B cituje z díla pocházejícího od A.

Neorientované hrany pak pouze informují o existenci propojení (at' už logického, nebo fyzického) mezi dvěma vrcholy. Pokud bychom tak chtěli převést plánek nějakého města do podoby grafu, kde uzly znamenají křižovatky a hrany ulice, využili bychom neorientovaných hran pro obousměrně průjezdné ulice a orientovaných pro ty jednosměrné.



(a) Orientovaná hrana (b) Neorientovaná hrana

Obrázek 2.2: Typy grafů

Dalším pojmem z oblasti teorie grafů je **podgraf**. Tímto termínem je myšlena podmnožina uzlů celého grafu a hrany mezi nimi. Pokud bychom měli najít opět praktické použití, vezměme v potaz dříve zmíněné mapové podklady. Pokud budeme hledat cestu z jedné části města do jiné, bude nás zajímat pouze několik málo křižovat a silnic mezi nimi. Stejně tak čtenář snadno nahlédne, že výsledný graf sociální sítě vytvořený v rámci této práce je tak rozsáhlý, že pro zachování přehlednosti a použitelnosti grafického výstupu je nutné omezit se pouze na jeho podgrafy.

Zapotřebí je ještě zavést termín **ohodnocený graf**. [7] definuje ohodnocené grafy takto:

Ohodnocený orientovaný graf (G, w) *je orientovaný graf* G *spolu s reálnou funkcí* $w : E(G) \to (0, \infty)$. *Je-li* e *hrana grafu* G, *číslo* $w(e)$ *se nazývá její* ohodnocení *nebo* váha.

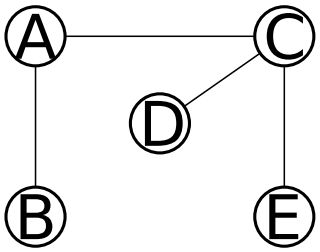
To znamená přiřadit každé spojnici vrcholů číslo, které kvantifikuje konkrétní veličinu. Může se jednat o vzdálenost mezi místy, průtok potrubí, maximální dosažitelnou přenosovou rychlost mezi síť ovými prvky či četnost citací. Ohodnocení hran se dále využívá pro optimalizační úlohy a hledání podgrafů. Například nalezení nejkratší či nejrychlejší cesty mezi dvěma body, kdy každá hrana má několik různých ohodnocení (mimo jiné vzdálenost, maximální povolená rychlost, průjezdnost), která se mohou při optimalizaci vzájemně kombinovat (autonavigace by při průjezdu Prahou měla preferovat podle aktuální dopravní situace průjezdnější trasy před těmi kratšími).

Poslední částí, které se budeme v tomto stručném uvedení do problematiky grafů věnovat, je **reprezentace grafů**. Kromě jejich vizuálního znázornění se využívá algebraických prostředků. V paměti počítače uchováváme nejčastěji graf pomocí matice sousednosti, kterou [6] definuje takto:

Necht' $G = (V, E)$ je graf s n vrcholy. Označme vrcholy $v_1, ..., v_n$ v nějakém libovolném pořadí. Matice sousednosti *grafu G je čtvercová $n \times n$ matice $A_G = (a_{ij})_{i,j=1}^{n}$ definovaná předpisem*

$$a_{i,j} = \begin{cases} 1 & pro\ \{v_i, v_j\} \in E \\ 0 & jinak. \end{cases}$$

Pokud se jedná o neorientovaný graf (viz 2.3), získáme podle tohoto přepisu symetrickou čtvercovou matici, která obsahuje duplicitní data. Naopak matice orientovaného grafu (viz 2.4) symetrická být nemusí (a obecně není – pouze v případech, kdy platí, že pokud existuje hrana $A \to B$, pak také existuje hrana $B \to A$).

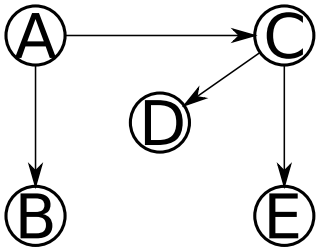


(a) Vizualizace

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

(b) Matice sousednosti

Obrázek 2.3: Neorientovaný graf



(a) Vizualizace

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

(b) Matice sousednosti

Obrázek 2.4: Orientovaný graf

Druhou možností, jak reprezentovat graf v paměti počítače, je **seznam sousednosti**. Každý uzel V má pak přiřazenu strukturu (většinou implementovanou pomocí spojového seznamu) obsahující všechny vrcholy, které s ním sousedí, lépe řečeno vrcholy, do nichž z V vede hrana. Grafy z obrázků 2.2b a 2.2a bychom popsali takto:

$A \to B, C$	$A \to B, C$
$B \to A$	$B \to$
$C \to A, D, E$	$C \to D, E$
$D \to C$	$D \to$
$E \to C$	$E \to$
Neorientovaný graf	Orientovaný graf

Grafové algoritmy

Od chvíle, kdy máme vytvořenu reprezentaci grafu, můžeme jej začít procházet. To znamená putovat z každého vrcholu do jeho sousedů za účelem nalezení řešení. Řešením může být nalezení nejlevnější cesty mezi dvěma uzly, nalezení nejbližšího vrcholu (případně hrany) mající nějakou vlastnost či nalezení všech řešení daného problému (v takovýchto úlohách se spíše než obecné grafy prohledávají stromy – neorientované grafy, které jsou souvislé a neobsahují kružnice – populární úlohy Osm dam, Rubikova kostka, apod.).

Při prohledávání se musí postupovat přísně systematicky, aby byla zaručena co největší efektivita, nedocházelo k tomu, že se na nějaké řešení zapomene a že se algoritmus na konečném grafu zacyklí a nikdy neskončí. Výběr správného přístupu je pak vždy otázkou toho, jaké požadavky na řešení klademe. Základní dva přístupy k prohledávání grafu jsou pak tyto: **Depth First Search** (DFS, neboli Prohledávání do hloubky) a **Breadth First Search** (BFS, Prohledávání do šířky).

Prohledávat graf do hloubky znamená expandovat vždy jen jednoho souseda (následníka), který zatím nebyl navštíven, označit jej jako navštívený, přesunout se do daného uzlu a znovu se zaměřit na jednoho souseda. Algoritmus se zastaví až ve vrcholu, ze kterého nelze dále pokračovat. Pak přichází na řadu backtracking, neboli vracení se vlastních stopách. Vrátíme se o krok zpět, kde se opět pokusíme expandovat prvního nenavštíveného následníka.

Tento postup se opakuje, dokud nejsou všechny uzly grafu označeny jako navštívené.

```
1: root v in graph G;
2: stack S;
3: for each u in G.nodes do
4:    u.visited ← FALSE;
5: end for
6: S.push(v);
7: while S is not empty do
8:    u ← S.pop;
9:    if ¬u.visited then
10:      u.visited ← TRUE;
11:      for each w in u.neighbours do
12:         S.push(w);
13:      end for
14:   end if
15: end while
```

Oproti tomu prohledávání do šířky expanduje nejprve všechny sousedy zpracovávaného uzlu. Změna v algoritmu je tedy evidentně v tom, že do zásobníku (prioritní fronty) jsou nejprve vloženi všichni sousedi, kteří jsou po jednom expandováni, a teprve tehdy, když jsou ze zásobníku vyjmuti, pokračuje se expanzí dalších vrcholů. Tento algoritmus použijeme spíše u takových problémů, kde chceme najít nejkratší způsob, jak dospět k řešení, neboť celý stavový prostor je procházen v případě stromu po patrech, v případě obecného grafu po kružnicích, čili první správné řešení, které je nalezeno, je tím nejbližším (proveditelným za nejmenší počet kroků).

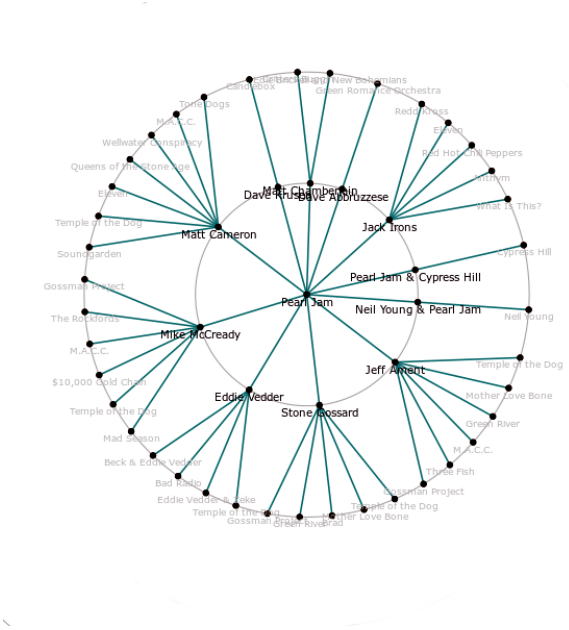
2.3.2 Knihovny a frameworky pro vizualizaci grafů

Pro vykreslení výsledku získaného z analýzy dat je zapotřebí použít nástroj pro vizualizaci grafů. Je možno buď využít existujících řešení, nebo vytvořit svoje vlastní. Vzhledem k tomu, že lze najít mnoho již existujících a vyzkoušených knihoven, jeví se jako lepší řešení je porovnat a zjistit, zda jsou pro tento projekt použitelné a vhodné, nebo zda je lepší přijít s vlastním novým způsobem.

Dracula Graph Library a JIT

Jako první si představme knihovnu představující robustní nástroj použitelný nejen pro vizualizaci, ale i pro další zpracování dat pomocí základních algoritmů, které jsou jeho součástí. Tato knihovna se nazývá Dracula Graph Library (dále jen Dracula). Projekt Dracula je založen na skriptovacím jazyce JavaScript. Jeho výstupem jsou moderně působící výstupy. Hlavní předností jest velmi jednoduchý způsob použití dohromady s rychlostí spojenou s tímto skriptovacím jazykem.

Knihovna je stále intenzivně vyvíjena (pod MIT licencí, která umožňuje i komerční použití) a není zcela stabilní. Veškeré závěry vyvozené z jejího testování jsou tak platné pro verze vydané v posledním kvartálu roku 2013. Jak bude dále vysvětleno, v době jejího testování nebyla vhodná pro větší grafy, ale svou jednoduchostí poslouží menším projektům, které potřebují rychlé a funkční řešení.



Obrázek 2.5: Příklad výstupu knihovny JIT

Velmi často je srovnávána s jiným nástrojem taktéž založeném na JavaScriptu, který se jmenuje JavaScript InfoVis Toolkit (častěji se objevuje pouze zkratka JIT). Obě knihovny podporují AJAX. I v případě JIT lze hovořit o vysoké kvalitě výstupu, který působí velmi moderně a uhlazeně. Stejně tak platí, že JIT umožňuje rychlé vytvoření výstupu za vyvinutí mini-

málního úsilí, ale pro úplného začátečníka je méně vhodný, neboť je o něco komplikovanější. Na druhou stranu pro větší projekty je vhodnější, neboť vývoj v něm je rychlejší a lze dosáhnout stejného výsledku úspornějším kódem. Další výhodou je lepší dokumentace a větší uživatelská základna. Lze tak najít i mnoho příkladů a řešených problémů na internetových fórech. Kód opět podléhá licenci MIT.

Java Universal Network/Graph Framework

Další testovaný framework nese název Java Universal Network/Graph Framework (známý spíše pod zkratkou JUNG). Je napsán v programovacím jazyce Java, ale je většina kódu je maximálně optimalizována. I tato knihovna kromě možnosti vykreslení grafu poskytuje velké množství funkcí umožňujících prohledávání grafu, klastrování, prořezávání stromu a mnoho dalších pokročilých algoritmů z oboru teorie grafů, data miningu a analýzy sociálních sítí. Navíc nabízí možnost výběru z mnoha druhů rozvržení grafu, které jsou vysoce pokročilé a framework se tak pomocí vestavěných algoritmů sám postará o návrh vizualizace. Velkou výhodou této knihovny je rychlost a stabilita.

Vytvoření jednoduchých grafů je snadné i pro neznalého uživatele. Navíc lze najít na internetu velké množství krátkých návodů zabývajících se základy práce s knihovnou. Vzhledem k minimalistické dokumentaci se programátor musí při problémech spoléhat spíše na uživatelskou základnu. Kód je šířen pod BSD licencí.

Vykreslování je velmi variabilní a lze uživatelsky nadefinovat podobu uzlů, hran i popisků. Takové zásahy jsou již však věcí vyžadující větší zkušenosti s frameworkem, neboť jsou řešeny dosti neintuitivně a zlepšení bohužel čekat v dohledné době nelze, neboť poslední verze 2.0.1 byla vydána již v roce 2010. Lze ale stále rozšiřovat funkcionalitu pomocí novějších knihoven třetích stran, které knihovna používá, především pak Cern Colt Scientific Library.

Tento framework je od samotného začátku zaměřen na zpracování většího množství dat než Dracula Graph. Přesto i zde může při velkém množství uzlů dojít k nešťastnému rozvržení layoutu, které vyústí v překřížené hrany a překrývající se popisky uzlů. Nezbývá pak než takový problém vyřešit ručně vhodným přiblížením či oddálením či přeskupením uzlů.

Poslední knihovna, o které se zmíníme, je šířena pod GNU General Pub-

lic License a jmenuje se GraphStream. Zaměřuje se především na vizualizaci a její dynamiku. Použitým programovacím jazykem je opět Java. Hlavní myšlenkou je automatický převod dat uložených v souboru na graf a dále jeho zobrazení (odtud název GraphStream).

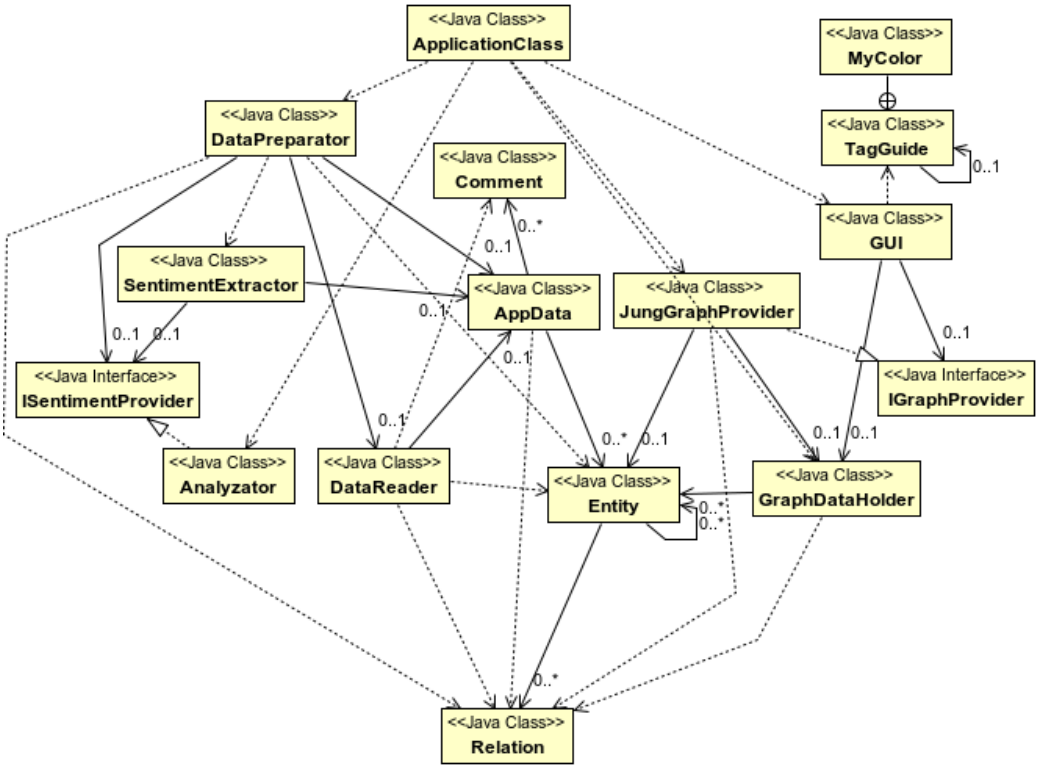
Zajímavostí této knihovny nejsou jen působivé výstupy, ale hlavně zcela odlišný přístup k tvorbě vzhledu grafu pomocí kaskádových stylů. Díky tomu má programátor obrovské možnosti co se týče designování výsledného produktu. Bohužel pro začátečníka to znamená zhoršení přehlednosti a čitelnosti kódu. Dalším problémem této knihovny je obtížnější implementace grafů vznikajících dynamicky, neboť to vždy přináší i automatické generování nových CSS souborů.

Na obrázku 2.5 si lze prohlédnout výstup knihovny JIT. Produkt knihovny JUNG a Dracula si lze prohlédnout na ukázkách z aplikace (viz například 4.2, 3.4 a 3.5).

3 SoNEx

3.1 Architektura aplikace

Architektura aplikace SoNEx je znázorněna pomocí UML digramu na obrázku 3.1. Na tomto diagramu jsou zakresleny všechny původní třídy aplikace (balíček `cz.zcu.fav.kiv.witz.bp`) a navíc třída `Analyzator`, která přísluší použitému analyzátoru polarity názoru (viz níže). Bližší informace o algoritmech a strukturách použitých v jednotlivých částech lze nalézt dále.



Obrázek 3.1: UML diagram aplikace

- **ApplicationClass** – Hlavní třída, která obsahuje metodu `main`. Definuje umístění vstupních dat a vytváří instance tříd potřebných pro jejich zpracování a zobrazení. V této třídě je možné definovat, které konkrétní implementace definovaných rozhraní budou použity.

- **AppData** – Tato třída je využívána pro uchovávání informací v průběhu prvotní analýzy dat. Obsahuje mapu, resp. slovník, entit (třída `Entity`), které zároveň propojuje s větami označenými jako komentáře, jejichž zpracování následuje. Jsou zde tak obsažena všechna data včetně vytvářeného grafu, neboť pouze zde jsou dostupné všechny existující instance třídy `Entity` představující položky seznamu sousednosti grafu.

- **Comment** – Třída je přepravkou uchovávající informace o každém komentáři. To znamená jeho umístění a dodatečné informace o jeho okolí.

- **DataPreparator** – Má za úkol řídit prvotní analýzu dat. Předává třídě `DataReader` postupně všechny soubory s nimi a vytváří instanci třídy `SentimentExtractor`, která provádí analýzu polarity názoru. Nakonec spojuje všechny mezivýsledky, které jsou ukládány do samostatných souborů algoritmem popsaným v 3.3.

- **DataReader** – Provádí analýzu vstupních souborů. Vyhledává pojmenované entity, určuje jejich vzájemnou polohu a tím i vztah. Pro každý soubor je vytvořena daty nenaplněná instance třídy `AppData`, do níž jsou ukládány výsledky každého úseku (článku, věty, komentáře). Stejně tak provazuje komentáře s příslušnými entitami.

- **Entity** – Tato třída udržuje všechny informace o uzlu grafu: název uzlu, seznam sousednosti a dodatečné informace o hranách (typ a míra vztahu, sentiment). Protože je použito pro uchovávání instancí kolekce `HashMap`, musí obsahovat metodu `hashCode()` a `equals()`.

- **GraphDataHolder** – Slouží k načtení a udržování dat ve 2. fázi, což je tvorba GUI. V 1. fázi byla data zpracována a výsledky uloženy do jednoho souboru jako modifikovaný seznam sousednosti, který je načten, a podle něj jsou vygenerovány seznam typů entit a barevná paleta použitá pro jejich vizuální reprezentaci. Samozřejmostí je slovník entit, ve které je klíčem název entity a hodnotou instance třídy `Entity`.

- **GUI** – Třída oddděná od třídy `JFrame`. Jedná se o jednoduché okno s komponentami zprostředkovávajícími uživatelský vstup, neboli filtry pro prohledávání grafu. Využívá dat poskytovaných třídou `GraphDataHolder` a stejně tak jí generovanou paletu barev přiřazených typům entit. Samotný graf dodává třída implementující rozhraní `IGraphProvider` jako `JPanel`, který je vsazen do hlavního okna. Podrobnější informace viz 3.6.2.

- **IGraphProvider** – Rozhraní, které musí splňovat třída vykreslující graf vsazovaný do hlavního okna aplikace. Taková třída musí implementovat veřejnou metodu `JPanel getGraph(String selectedItem, int selectedRelType, boolean[] selectedTypes, int userDepth, int userRelLvl)`. Parametry jsou uživatelsky navolené složky filtru (výchozí entita, typ vazby, typ entit, hloubka hledání, minimální četnost společného výskytu).

- **ISentimentProvider** – Toto rozhraní musejí implementovat třídy, které provádějí analýzu sentimentu. Vzhledem k tomu, že byl použit již existující nástroj (viz 3.5.3), byl zachován český název metod v něm použitý. Jde o metody `getSentimentVety(String sentence)` a `naplnSlovnik(File dict, ETypSlova typ)`.

- **JungGraphProvider** – Tato implementace rozhraní `IGraphProvider` využívá knihovny Jung popsané v 2.3.2 a 3.6.5. Poskytuje `JPanel`, který je možno přidat do `contentPane` jiné komponenty. Navíc poskytuje rozbalovací roletu s možností přepínání mezi editačním a prohlížecím módem.

- **Relation** – Třída slouží jako přepravka, do níž se ukládají informace o vazbě mezi dvěma entitami. Síla jednotlivých typů vazby a zjištěný sentiment.

- **SentimentExtractor** – Nalezení přesného rozsahu textu, komentáře, který je relevantní pro analýzu polarity názoru je úkolem této třídy. Podle dat uložených v instanci třídy `Comment` je určena poloha uvozovek. Algoritmus v jejich okolí hledá uvozovací větu, výsledek předá k analýze sentimentu (třída implementující `ISentimentProvider`) a výsledek uloží do příslušných instancí třídy `Relation`.

- **TagGuide** – Podle vzoru *Singleton* navržený jednoduchý dialog pro výběr typů entit k analýze a zobrazení. Využívá paletu barev a seznam typů entit generované třídou `GraphDataHolder`.

3.2 Vstupní data

3.2.1 Vznik a podoba

Po teoretickém uvedení do problematiky se podívejme na aplikaci výše zmíněných metod v praxi a jejich využití pro účely cílové aplikace nazvané SoNEx (Social Network EXtractor). Základním stavebním kamenem této bakalářské práce jsou vstupní data, která vznikla na základě zpracování novinových článků, jejichž původcem je Česká tisková kancelář (ČTK). Původní souvislé texty byly tokenizovány 2.2.1, lemmatizovány 2.2.1 a, což je pro cílovou aplikaci nejdůležitější, nad těmito daty bylo provedeno rozpoznávání pojmenovaných entit, jehož produktem se budeme především zabývat.

3.2.2 Formát

Textový soubor, který je výsledkem výše popsané sekvence úkonů, sestává ze 4 sloupců. První sloupec obsahuje původní slova a znaky původního analyzovaného textu v původním pořadí. Čtením prvního sloupce řádek po řádku tak získáme původní nezměněné články. Vzhledem k tomu, že všechny potřebné informace následují až v dalších sloupcích, není tento pro účely aplikace použit.

Druhý sloupec každého řádku uchovává lemmatizovaný tvar slova ze sloupce prvního. Na lemmatizovaný tvar se dále více zaměříme, neboť právě z něj budeme čerpat jména entit.

Třetí sloupec obsahuje značky poskytující informace o tvaru slova v původním textu (mimo jiné pád, slovní druh atd.). Takové informace jsou pro výslednou aplikaci nadbytečné a nebudeme si jimi proto dále zabývat.

Zajímavý pro nás bude opět až sloupec čtvrtý, poslední. V něm totiž nalezneme tag, který říká, zda se jedná o pojmenovanou entitu. Pokud ano, obsahuje i informaci o jaký konkrétní typ se jedná. Tagy pomohou nejen pro barevné rozlišení jednotlivých typů, které pomůže čitelnosti dat, ale zároveň poslouží k snadné filtraci. Každý druh textů bude používat pravděpodobně jinou sadu tagů. V obecných novinových článcích nás bude zajímat, zda je daná entita osobou, firmou, státem, městem, apod., ale pokud bychom měli specializovanější texty, bude nás spíše zajímat, ke kterému fotbalovému klubu

zmíněný hráč přísluší, kdo je autorem díla atd. Můžeme si pak snadno vybrat, že nás například zajímají pouze hráči plzeňské Viktorky či příslušníci některé strany. V případě předložených vstupních dat můžeme filtrovat dle skupin (tagů) zmíněných níže.

3.2.3 Ukázka

```
Česká  český AAFS1----1A----B-country
republika  republika NNFS1-----A----I-country
převzala  převzít VpQW---XR-AA---0
s  s-1 RR--7----------0
příchodem  příchod NNIS7-----A----0
nového  nový AAIS2----1A----0
roku  rok NNIS2-----A---1 0
předsednictví  předsednictví NNNS4-----A----0
v  v-1 RR--6----------0
Evropské  evropský AAFS6----1A----B-organization
unii  unie NNFS6-----A----I-organization
.  . Z:-------------0
```

Na vybraném vzorku je vidět, jak jsou původní texty transformovány a o jaké informace jsou doplněny. Jak již bylo řečeno, pro analýzu byl upřednostněn druhý sloupec před sloupcem prvním. Hlavním důvodem je ten fakt, že se jedná o lemmatizovaný tvar, který sjednocuje všechny vyskloňované formy daného výrazu. Nevýhoda tohoto přístupu je ovšem zjevná i z výtahu vstupních dat, kde v prvním sloupci můžeme vidět správný tvar „Česká republika“, který lemmatizací přešel v nesprávný tvar „český republika“. Pokud tak bude celý název takto deformován, získáme takovým způsobem tedy správně všechny výskyty dané entity, ale u všech bude uveden stejný nesprávný název.

Nástroj, který byl použit pro rozbor vstupních dat, navíc v mnoha případech neurčuje správně lemma některých entit, a tak například vzniknou dvě různé verze téhož jména (např. „Mirek Topolánek“ a „Mirek Topolánka“). Nejdůležitějším poznatkem, který se týká vstupních dat, je tak ten, že by bylo pro výslednou aplikaci velmi užitečné jejich rozšíření o další položku, která by vznikla na základě další analýzy – například pomocí vyhledáváni v rozsáhlejší databázi (viz 2.2.2), jenž by v tomto triviálním případě odhalilo různé pojmenování jedné entity.

Jak již bylo řečeno výše, důležitý je pro tuto práci poslední sloupec dat obsahující typ pojmenované entity. V datech, která byla dodána pro ověření funkčnosti výsledné aplikace, můžeme nalézt tyto tagy: `city`, `country`, `e-subject`, `figure`, `geography`, `nationality` `organization`, `problematic`, `region`, `religion`, `sport`, `sport-club`

Vzhledem k povaze původních textů je zvolen velmi zobecňující způsob rozčleňování do jednotlivých skupin, což sice neumožní příliš jemné filtrování, ale na druhou stranu nezpůsobuje nepřehlednost samotného jejich nastavování, která by nutně vznikla při větším množství definovaných skupin.

3.3 Předzpracování vstupních dat

Pro tvorbu té části aplikace, která bude mít na starosti analýzu dat, byl vybrán programovací jazyk Java, což předurčilo i způsob, jakým se musí k datům jako takovým přistupovat. Jedním z důvodu je ten, že Java je objektovým jazykem, ve kterém je objektem úplně vše, což má nepříznivý dopad na množství použité operační paměti. Zatímco testovací vstupní data pro aplikaci dodaná pouze pro testovací účely a pro seznámení se s jejich formátem měla velikost v řádu jednotek megabajtů, což samozřejmě neznamená při dnešní velikosti paměti žádný problém, finální data již představoval textový soubor o velikosti několika gigabajtů. Mějme na paměti, že si musíme uchovat reprezentaci grafu a ke každé entitě množství dodatečných dat. Ač tedy aplikaci analyzující testovací data stačila výchozí přidělená velikost 2 GB RAM, pro finální data nestačila již ani rozšířená kapacita 4 GB. Vzhledem k tomuto a k faktu, že lze očekávat i data mnohem větší, je zapotřebí si vstupní soubor rozdělit na úseky, které se budou zpracovávat zvlášť, výsledky odkládat na disk a částečné výsledky nakonec spojit.

Tvorba grafu z textu vyžaduje nejvíce strojového času, ale naštěstí se jedná o jednorázovou operaci. Stačí ji tak provézt pouze při změně dat. Pokud dat přibude, velmi jednoduše lze přidat další soubory, které budou samostatně zpracovány a výsledky budou spojeny s výstupy předchozích analýz.

Jednotlivé mezivýsledky mají stejný formát jako výsledný soubor, který představuje reprezentaci grafu pomocí seznamu sousednosti doplněného o informace o relaci mezi entitami. Každá entita je tak uložena v takovémto formátu:

```
 Air France organization:: Paříž: A:52 S:35 Q:0 L:0;; ČTK: A:59
S:47 Q:0 L:0;; Brazílie: A:30 S:22 Q:0 L:0;; Atlantik: A:52 S:43
Q:0 L:0;;...
```

Entita *Air France* byla označena za organizaci a má sousedy *Paříž*, *ČTK*, *Brazílie*, *Atlantik* atd. S entitou *Paříž* se vyskytovala v 52 článcích, 35 větách a nevyskytovaly se spolu v žádné větě obsahující přímou řeč nebo citaci, což znamená, že i sentiment mezi nimi je neutrální (0).

Předzpracování je součástí třídy `DataPreparator`, která se postará o správu souborů a správný postup jejich analýzy. Po dokončení všech částečných analýz je vytvořena skupina souborů o počtu odpovídajícím vstupním souborům. Data jsou ve výše popsaném formátu seřazena v abecedním pořadí a při jejich spojování je této vlastnosti využito. Soubory jsou procházeny všechny zároveň řádek po řádku a mezi všemi je nalezen „minimální“ řádek, tj. takový, který obsahuje entitu ležící v abecedě před ostatními, který je zapsán do výstupu, nebo entitu shodnou s entitou v jiném souboru, která je spojena a zapsána. Tímto způsobem jsou všechny entity spojeny.

3.4 Problémy při analýze dat

Vzhledem k charakteru vstupních dat nastává několik problémů. Prvním z nich jsou záhlaví a zápatí uvozující všechny novinové články, kde se lze dozvědět více o zdroji a autorovi článku. Data poskytnutá zadavatelem pocházejí ze zdrojů České tiskové kanceláře (ČTK). Po nahlédnutí do poskytnutých dat si na první pohled lze všimnout, že každý článek je uvozen příslušnou hlavičkou poukazující na to, že právě ČTK je původcem tohoto textu. Taková informace je zajisté nezanedbatelná, ovšem pokud by měla být zahrnuta do sociální sítě, stává se dosti matoucí, neboť by veškeré přítomné entity byly ve vztahu s ČTK alespoň na úrovni společného výskytu ve článku. Nejenom že by tak mohla vzniknout nesmyslná vazba, ale navíc (což je mnohem závažnější) by se vytvořily tranzitivní vztahy (A $\rightarrow$ ČTK$\rightarrow$ B) mezi všemi entitami.

V každém článku nalezneme vždy minimálně jeden výskyt této zkratky, ať už v záhlaví nebo ve větách typu „... zjistila ČTK“ nebo „... sdělil ČTK zdroj blízký XY“. Zmírnit tento nepříznivý vliv se dá samozřejmě nezapočítáním výskytu v záhlaví. Zda je nežádaný i výskyt v rámci dalšího textu, je

diskutabilní. Můžeme z něj alespoň vyčíst, kde ve světě a jakém oboru působí která zpravodajská agentura podle jejích sousedů v grafu. Bohužel v dnešním zpravodajství není časté zveřejňovat původní zdroj, ze kterého je informace převzata, a často se používá fráze „jak se AB podařilo zjistit...“ apod., kde AB je označení až posledního článku zpravodajského řetězce.

Zde je popsán problém pouze pro jednu konkrétní entitu, ale je samozřejmě obecný a záleží vždy na typu vstupních dat. Jednou z možností je ignorovat vazby, které se výrazně vymykají běžné četnosti, což by ale mohlo vézt ke ztrátě těch nejvýznamnějších dat. Jako lepší možnost se proto jeví možnost potlačení některých entit ručně. Pokud uživatel zpozoruje nežádoucí výskyt některé entity, stačí ji přidat do seznamu a aplikace se již automaticky postará o její vymaskování.

3.5 Tvorba sociální sítě

3.5.1 Graf a jeho reprezentace

Metod pro reprezentaci grafu v paměti počítače je mnoho (některé viz 2.3.1). Pro uchování grafu tvořeného ze vstupních dat pojmenovanými entitami byl po zralé úvaze zvolen seznam sousednosti. Aby mohla být použita matice, musela by buď to dynamicky zvětšovat svou velikost, nebo by muselo být zpracování vstupního souboru provedeno ve dvou fázích: spočítání všech unikátních entit a teprve potom vytvoření matice o správné velikosti a její naplnění při druhém průchodu daty. Dynamické zvětšování matice by byla operace velmi drahá a druhá zmíněná varianta představuje jeden průchod souboru navíc s tím, že hledání správné buňky v matici (relace mezi dvěma entitami) by již od prvního momentu znamenala hledání ve všech entitách v daném souboru. Jejich hledání by tak i při všech optimalizacích znamenalo vysokou časovou náročnost již od první nalezené entity.

Použití seznamu sousednosti umožní projít soubor pouze jednou a navíc, což je důležitější, vyhledávání v množině entit na začátku tvorby grafu je velmi rychlé, neboť obsahuje malý počet položek, který se zvětšuje s množstvím analyzovaných dat.

Optimalizace v tomto případě hraje významnou roli. Je sice pravda, že prvotní analýza dat je jednorázová operace, ale je časově a paměťově velmi

náročná. Řešení paměťové náročnosti bylo již popsáno. Jednou z věcí, která tyto zvýšené požadavky způsobuje je právě snaha o urychlení. Všechny entity jsou tak uloženy ve slovníku, kde klíčem je název jednotlivých entit a hodnotou odkaz na k ní příslušící objekt. Pro jeho implementaci je použita standardní kolekce jazyka Java `HashMap`, která umožňuje velmi rychlé vyhledávání pomocí klíče, neboť je využito hashovací tabulky.

Je logické, že graf neobsahuje orientované hrany. Jedinými úseky textu, ze kterých by bylo možno vyčíst směr hrany, jsou přímé a nepřímé řeči. To by ale znamenalo určovat vlastnosti hran v obou směrech a též vykreslování tří různých hran mezi entitami (oba směry a neorientovaná hrana pro ostatní případy), což by vytvořilo velmi nepřehlednou změť čar ve výsledné vizualizaci. Navíc nebyl spatřen smysl v uchovávání takových informací.

Tvorbu sociální sítě má na starosti třída `DataReader`, která přijímá postupně soubory ke zpracování. Veškeré datové struktury použité při analýze jsou udržovány a spravovány v rámci třídy `AppData`, která dále pracuje se třídami `Entity`, což je de facto přepravka uchovávající informaci o každé entitě a zároveň s tím seznam sousedů daného uzlu grafu, a `Relation`, což je přepravka s údaji popisujícími vazbu mezi vrcholy (četnost společného výskytu a celkový sentiment). Poslední třídou je `Comment`, která se používá pro určení polohy vět zvolených k analýze sentimentu.

3.5.2 Relace

Aplikace na základě tagů obsažených ve vstupních datech rozpoznává pojmenované entity a vytváří relace mezi nimi. Lze rozlišit 3 druhy vztahů na základě jejich vzájemné pozice. Dvě entity se mohou vyskytovat ve velmi volném vztahu, kterým je společný výskyt v rámci celého jednoho novinového článku. Takový typ vazby je nejslabší a měl by se na výsledku projevit nejmenším dílem. Je však zdaleka nejčetnějším.

Silnější spojení značí koexistence dvou entit v jedné větě. Nijak nepřekvapí, že z výsledků analýzy lze vyčíst, že již není tak častá jako společný výskyt v celém článku, který může být složitě větvený a jednotlivé části spolu nemusejí souviset, nebo může být jejich souvislost jen vzdálená.

Za nejsilnější vazbu bychom měli považovat na sociální síti vazbu typu přímá řeč, citace a všechny výskyty typu „A“B (*„Orco Group se potácí v problémech,“ varují Hospodářské noviny.*), „A,B“(*„Oběda u prezidenta Klause se*

zúčastní jak premiér Nečas, tak předseda Senátu", informovala kancelář prezidenta.) a „..."A,B (*„Počkejte si na tiskové prohlášení," prohlásili svorně Paroubek i Hašek.*). Nejenom že lze z těchto vět usuzovat na silné spojení mezi entitami, ale navíc lze ve větách tohoto typu najít další upřesňující informace, které by danou vazbu přesněji popsaly. V tomto případě budeme považovat věty tohoto typu za texty stejného charakteru jakého jsou komentáře či příspěvky na sociálních sítích, jak je známe. Jak již bylo řečeno (2.2.3), velmi zajímavým odvětvím je analýza sentimentu takovýchto příspěvků. Pokusíme se analyzovat i tyto texty na výskyt slov lépe popisujících danou vazbu.

3.5.3 Polarita vztahu

Člověk je při pohledu na nějaký komentář, diskusi či jakýkoliv jiný text schopen rozhodnout, zda je obsah tohoto sdělení negativní, pozitivní, či neutrální, byť se při určování této vlastnosti může stát, že se dva lidé ve svém názoru neshodnou. Kromě strohého údaje o četnosti a síle vazby tak lze navíc brát v potaz tuto hůře kvantifikovatelnou informaci.

Analýza sentimentu (polarity) je prováděna ve druhé fázi, neboli při druhém průchodu dat. Při prvním je pouze zaznamenána poloha vět určených ke zpracování. Důvodem je skutečnost, že ve zdrojových datech (především v původních testovacích) se často vyskytují uvozovky postrádající párovou značku. Při druhém průchodu už se tak stačí zaměřit na okolí takových značek a snažit se zrekonstruovat uvozený text.

V textu jsou vytypovány úseky, u kterých lze předpokládat, že jejich analýzou získáme polaritu názoru či vztahu mezi entitami. Ta je do programu dodána pomocí třídy `SentimentExtractor`, která je zjednodušující derivací bakalářské práce Karla Zíbara (Katedra informatiky a výpočetní techniky, Fakulta aplikovaných věd, Západočeská univerzita v Plzni) z roku 2014. Výsledky její aplikace se budeme zabývat dále.

Použití tohoto externího nástroje umožňuje jednoduchou výměnu za jiný, který by byl založen na jiných metodách analýzy a poskytoval jiné výsledky, které by spíše odpovídaly povaze dat, neboť jeho původní použití se zcela neschoduje s jeho aplikací v této práci.

3.6 Grafický výstup a GUI

Data grafu jsou po předchozí analýze uložena v jednom souboru, jehož vznik a podoba jsou popsány výše. To umožňuje při spuštění programu pouze je načíst a uložit do paměti. Aby se zmenšily nároky na paměť, nejsou použity stejné datové struktury jako při tvorbě grafu. Jediná entita, kterou je zapotřebí nalézt mezi všemi, je první, centrální, ze které se k dalším přistupuje ze seznamu sousednosti se složitostí $O(1)$.

3.6.1 Filtrace dat

Zobrazení celého grafu obsahujícího všechny uzly a hrany by znamenalo vytvořit nepřehlednou změť hran a čar, ve které by se veškeré informace ztratily, a taková vizualizace by byla k nepoužití. Proto je zapotřebí získat od uživatele získat informace umožňující omezení počtu entit, neboli zadání filtru.

Pro komunikaci s uživatelem bylo vytvořeno jednoduché uživatelské rozhraní, které umožňuje zadat jméno entity, která se stane centrem vytvářeného grafu. Další parametry jsou samozřejmě hloubka prohledávání, což znamená, že lze například vyhledat pouze ty entity, které jsou přímo spojeny s daným centrem apod. Dalším užitečným parametrem je typ relace, podle kterého se má vyhledávat. Lze volit mezi společným výskytem v článku, větě či citaci.

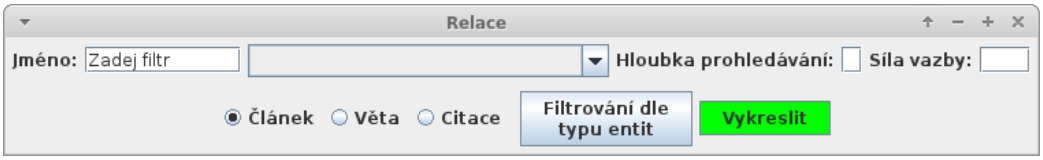
Další možnosti filtrace jsou velmi užitečné, protože jejich použitím se lze zaměřit na velmi specifické podmnožiny grafu. Prvním filtrem je omezení podle četnosti společného výskytu. Lze tak ignorovat nevýznamné relace/entity pro hledaný centrální bod. Stejně tak účinným způsobem omezení množství hran a uzlů je nastavení množiny tagů, které chceme brát v potaz. Lze tak vyhledávat pouze mezi osobami, městy a podobně podle charakteru vstupních dat (seznam tagů je generován automaticky).

Pro větší přehlednost a lepší orientaci v grafu jsou uzly obarveny. Centrální uzel, ze kterého je započato prohledávání grafu, je vždy obarven černě. Ostatní uzly mají barvu podle typu entity. Každému tagu je automaticky jedna přiřazena. Lze si pak nechat zobrazit legendu a vyselektovat si ty, které chce uživatel skrýt. Stejně tak jsou obarveny i hrany podle sentimentu určeného externím nástrojem.

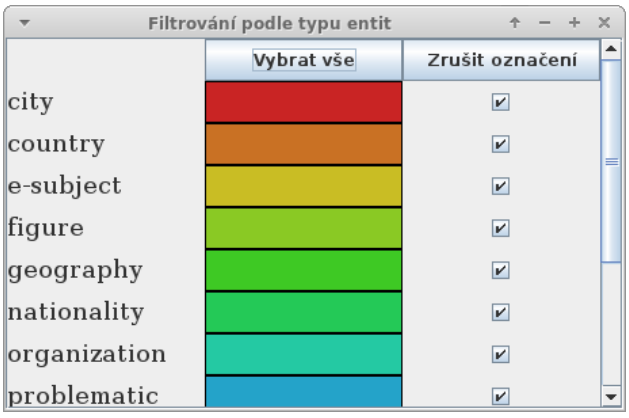
3.6.2 Tvorba GUI

GUI bylo vytvořeno za použití knihovny `Swing`, která je sice pomalejší než například `SWT`, které používá nativní knihovny operačního systému, protože vše je vykreslováno pomocí `Java2D`. Výhodou je lepší přenositelnost za použití méně kódu. Výkonnostní nedostatky se navíc při takto minimalistickém uživatelském rozhraní výrazněji neprojevují. Veškerý kód byl psán ručně v prostředí Eclipse, neboli nebylo využito například designeru prostředí NetBeans, které by generovalo kód poskytující za malého úsilí možná lepší výsledek než ručně tvořený, ale kód neoptimalizovaný, a tím pádem pomalejší.

Základem této části je třída `GUI`, která obsahuje vytvoření jednoduchého okna s několika ovládacími prvky, kteréžto jsou popsány výše. Výstup si lze prohlédnout na obrázku 3.2 Možnost filtrování podle tagů je poskytována další třídou `TagGuide` představující jednoduché okno. Poslední použitou je třída `MyColor` dědící od třídy `JLabel`. Ta slouží ke správnému vykreslování legendy (barev použitých pro označování typů entit podle tagů, viz obrázek 3.3).



Obrázek 3.2: Výsledné GUI



Obrázek 3.3: Dialog pro filtrování podle typu entit

3.6.3 Vyhledávání a podgrafy

Již bylo řečeno, že pro vykreslení grafu je vyžádán startovací uzel (říkejme mu kořen). Z něj je započato vyhledávání omezené zadanými parametry (hloubka, typy entit, typ vazeb, síla vazby). Pro tyto potřeby bylo zvoleno BFS, neboli prohledávání do šířky (pro více informací viz 2.3.1).

Tento algoritmus byl vybrán proto, že pro tento případ je algoritmus snáze pochopitelný a lehce implementovatelný. Na začátku je do prázdné fronty vložen kořen. Dále se postupuje tak, že je vybrán jeden vrchol z fronty a jeho následovníci přidáni do fronty, pokud zatím nebyli zpracováni, mají typ, který není uživatelsky ignorován, a nejsou vzdáleni od kořene více, než je požadovaná maximální hloubka grafu. Do třídy `Entity` tak musela být dodána položka, která znamená minimální vzdálenost od kořene.

Pomocí zadaných filtrů tak vznikají podgrafy daných vlastností. Lze též nastavit hodnotu maximální hloubky grafu tak vysokou, že by byl obsažen celý graf, ale takový požadavek se neočekává, neboť by výsledný graf byl příliš složitý.

3.6.4 Vizualizace grafu

V části 2.3.2 jsme se seznámili pouze s těmi knihovnami, se kterými byly prováděny pokusy v rámci aplikace SoNEx. V úvahu byly vzaty samozřejmě i jiné možnosti, které ovšem většinou byly zavrženy z důvodu nesrozumitelné dokumentace, nelíbivého vzhledu výstupu nebo nadmíru složitého propojení na existující kód.

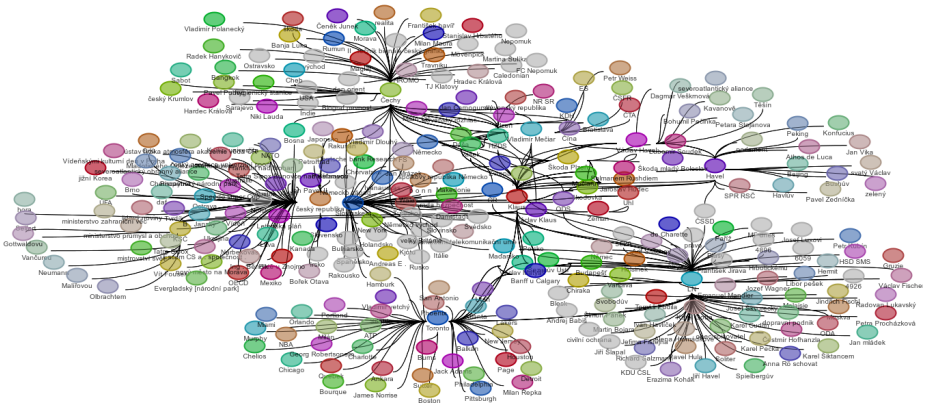
První pokusy byly učiněny s moderní JavaScriptovou knihovnou Dracula Graph Library (2.3.2). Propojení s částí psanou v programovacím jazyce Java proběhlo jednoduše tak, že část skriptu obsahující tvorbu grafu byla dynamicky generována samotnou aplikací (zároveň s procházením grafu byl tvořen skript).

Bohužel byly po vytvoření funkční kostry odhaleny chyby menší závažnosti, které by nebylo těžké odstranit (chybné zobrazování českých znaků apod.), ale i chyba, která je těžko odstranitelná a svou závažností znemožnila pokračovat dále za použití této knihovny. Přesto, že byla funkčnost testována na poměrně malých datech a výsledný skript tak nebyl příliš rozsáhlý,

byla rychlost JavaScriptové části neúnosně nízká a pro real-timové využití se ukázala být knihovna naprosto nepoužitelná. Zobrazení v prohlížeči pak doprovázelo zobrazování varovných hlášek poukazujících na abnormálně dlouhý běh skriptu na stránce. Zásadním nedostatkem použité verze byla i absence možnosti zoomování.

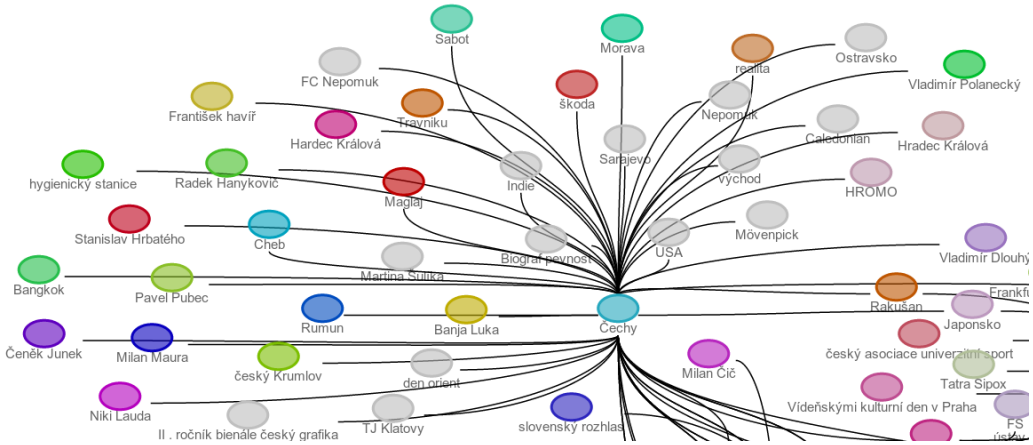
Lze ji však na základě zkušeností doporučit pro malé grafy, neboť je její základní použití opravdu triviální. Ukázky jednoduchého kódu i výstupu je možno nalézt na: `http://www.graphdracula.net/`. Zde je ukázka nejjednodušššího grafu, který lze pro potřeby této práce vytvořit.

```
$(document).ready(function() {
        var width = $(document).width();
        var height = $(document).height();
        var g = new Graph();

        g.addEdge("Klaus", "Jaroslav Hudec");
        g.addEdge("Klaus", "LN");
        g.addEdge("Klaus", "Toronto");
        ...

        var layouter = new Graph.Layout.Spring(g);
        var renderer = new Graph.Renderer.Raphael(
        'canvas', g, width, height);
});
```



Obrázek 3.4: Dracula Graph Library

Mnohem lepších výsledků bylo dosaženo pomocí frameworku JUNG psaného v jazyce Java, který splnil všechny požadavky, ačkoliv jeho dokumentace



Obrázek 3.5: Dracula Graph Library – detail

je velmi stručná a výstup neupraveného vizualizéru nemá tak moderní vzhled jako v případě některých jiných testovaných, ale díky tomu, že je velmi variabilní, lze jeho vzhled měnit téměř bez omezení. Knihovna obsahuje velké množství předpřipravených grafových algoritmů, které ale vzhledem k požadavkům na filtrování dat musejí být implementovány samostatně, a funkcí pro práci s grafy. Výsledný kód je dostatečně rychlý a umožňuje nejsnazší možný způsob propojení s předchozím modulem pro analýzu dat.

Velmi snadno lze propojit výstup frameworku i s grafickým uživatelským rozhraním, protože samotný JUNG vykresluje svůj výstup do komponenty `JFrame` či `JPanel` knihovny `Swing`. V aplikaci je použito vykreslování do panelu, který je zasazen do okna GUI společně s ovládacími prvky. JUNG poskytuje i další komponenty, které lze použít na ovládání a volání některých funkcí grafu, jako například přepínání mezi režimy zobrazení, zapínání automatického klastrování apod. Zajistit správné fungování všech těchto funkcí je náročné, ale pro SoNEx bylo použito pouze přepínání režimů zobrazení, neboť pro ostatní funkcionalitu nebylo nalezeno využití.

3.6.5 Použití knihovny JUNG

Na internetu lze najít mnoho dílčích návodů, jak pracovat s frameworkem JUNG. Spíše se jedná o úseky kódu prezentované na diskusních fórech a podobně. Z tohoto důvodu lze dále nalézt alespoň některé z nejdůležitějších metod a jejich použití pro tvorbu grafu a úpravu jeho designu.

V případě této aplikace byl použit neorientovaný graf (třída `UndirectedGraph`). Do něj lze přidávat nové vrcholy metodou `addVertex(String)` a nové hrany metodou `addEdge(String, String, String)`, kde první argument je název hrany, podle kterého lze v množině hran vyhledávat. V našem případě se do textu ukládají též informace o hraně, které lze pak snadno načítat při vykreslování pro určení barvy a síly hrany.

Po vytvoření grafu pak již jen stačí předat jej třídě, která se bude sama automaticky starat o rozvržení uzlů při vykreslení. Volba layoutu je otázkou použití a jejich seznam i s ukázkou výstupu je k dispozici na stránkách knihovny. Dalším krokem je předání layoutu třídě, která zajišťuje veškerou funkcionalitu zobrazování. Zde byla použita třída `VisualizationViewer`, který lze podobně jako komponenty knihovny swing přidat do obsahu okna či panelu. Základem úpravy vzhledu je pak jeho metoda `getRenderContext()`, nad kterou můžeme volat všechny potřebné settery. Jednoduché metody typu `setLabelOffset(int)` či `setPrefferedSize(Dimension)` nepotřebují komentovat, ale všechny metody nastavující `Transformer` (třídu popisující způsob vykreslení dané komponenty překrytím metody `Paint transform(String)`) jsou obecně složitější a definování vzhledu za jejich použití není příliš intuitivní. Nicméně názvy metod jsou vhodně zvoleny, a programátor se tak rychle zorientuje.

Například barva hran se nastavuje pomocí metody `setEdgeDrawPaint-Transformer(Transformer<String, Paint>)`, která podle předdefinované palety nastaví hraně barvu odpovídající stupni sentimentu (odstíny zelené a červené), pomocí `setVertexFillPaintTransformer(Transformer<String, Paint>)` se nastavuje barva uzlu podle tagu apod.

Na závěr tohoto krátkého exkurzu do používání knihovny JUNG je zapotřebí upozornit, že graf je překreslován při každé změně grafu, tedy i při posouvání či přibližování. Z toho plyne, že všechny funkce použité k definování vzhledu by měly poskytovat výsledek za co nejmenší možnou dobu, neboť pro větší graf by neoptimalizovaný kód znamenal velmi dlouhou odezvu aplikace.

3.7 Testování

Při rozpoznávání vzájemné polohy entit není problémem rozpoznání společného výskytu ve stejném článku (alespoň při dodržení vhodného formátu

dat), ale rozpoznávání vzájemné polohy ve větě je ztíženo stejným problémem jako rozpoznávání hranic věty. Problémem je tak například tečka za titulem. V předložených datech není titul součástí pojmenované entity a proto nelze z těchto tagů usoudit, že se jedná o její součást.

V dodaných datech se vyskytuje velmi málo akademických titulů, proto se úspěšnost pohybuje velmi vysoko (ve vybraném vzorku 1000 vět se dokonce tento problém neobjevil). Například titul *Mgr.* se na celkovém počtu 41644557 řádek vyskytuje dvanáctkrát, *Ing.* devětadvacetkrát a *Bc.* dokonce jen dvakrát. Pokud by byla tato chyba závažnější, bylo by na místě přidat slovník slov, po kterých se má tečka jako větný oddělovač ignorovat.

Zajímavější z hlediska testování je tak tvorba relací typu citace a analýza sentimentu. Prvním úkolem testování je stanovit, zda je vůbec daná věta relevantní pro vyhledávání vzájemného vztahu entit a následně zda je sentiment správně určen. Opět byl sestaven vzorek o velikosti 1000 úryvků, které byly určeny algoritmem k analýze sentimentu popisujícího vztah mezi entitami. Výsledky popisuje následující tabulka (3.1).

Relevantní pro analýzu sentimentu		Určený sentiment	
Ano	Ne	Správně	Nesprávně
64,3 %	35,7 %	57,8 %	42,2 %

Tabulka 3.1: Výsledky testování

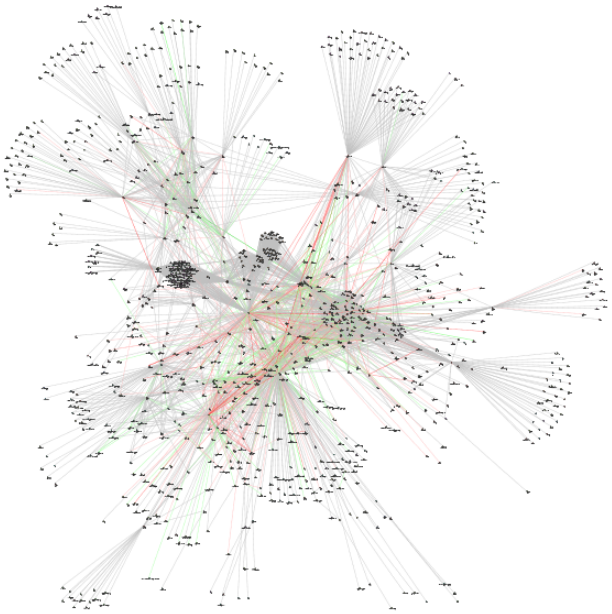
Pojem relevance komentářů pro analýzu sentimentu znamená, že v komentáři se vyskytují alespoň dvě entity, jejichž vztah lze na základě úryvku z hlediska sentimentu kvantifikovat. Příkladem (tento a další uvedené nejsou v doslovném znění, nejsou citací a jsou určeny pouze pro názornou demonstraci problematiky) takové věty je *„Neumím si představit lepšího zaměstnavatele," libuje si vývojář Google, Jon Smith*, ve které člověk i určí pozitivní hodnotu sentimentu. Mezi úryvky, které jsou a nejsou relevantní je mnohdy tenká hranice. Na příkladu *„Očekáváme, že cena zlata bude v příštím kvartálu mírné růst", sdělil deníku mluvčí ČNB Marek Petruš* vidíme, že při velmi podobné větné skladbě a stejné pozici entit se situace zcela změnila. Relevantnost pro vyhodnocení sentimentu je diskutabilní a v případě, že bychom usoudili, že je vhodné sentiment určovat, jeho hodnota by měla být neutrální. Výsledek analýzy je tak velmi často právě neutrální hodnota, konkrétně v 46 % z celkového počtu správně anotovaných úryvků.

4 Experimenty

Výstupem práce je aplikace zaměřená na vizuální získávání informací z vykresleného grafu. Je proto vhodné zde ukázat výsledky, kterých lze pomocí aplikace SoNEx dosáhnout. Navíc nám důkladný pohled na prezentované výsledky a zamyšlení se nad nimi poskytne představu o úspěchu či neúspěchu každé součásti programu.

4.1 Layout velkých grafů

Problémem je vždy nadimenzovat velikost plátna, na které se má kreslit. Byla proto použita dynamická velikost podle počtu uzlů obsažených v daném podgrafu. Na obrázku 4.1 lze vidět výstup vyhledávání pro parametry: *Václav Klaus*, hloubka prohledávání: *3*, minimální společný výskyt: *100*, typ společného výskytu: *článek*, filtr entit: *všechny*. Hrany jsou vzhledem k celkové velikost možná zbytečně dlouhé, ale nevznikají nesmyslné změti čar a uzlů. Použití `FRLayout` je však pro tyto účely použitelné (na rozdíl od jiných designérů obsažených v JUNG), ale je zde prostor pro zlepšení.

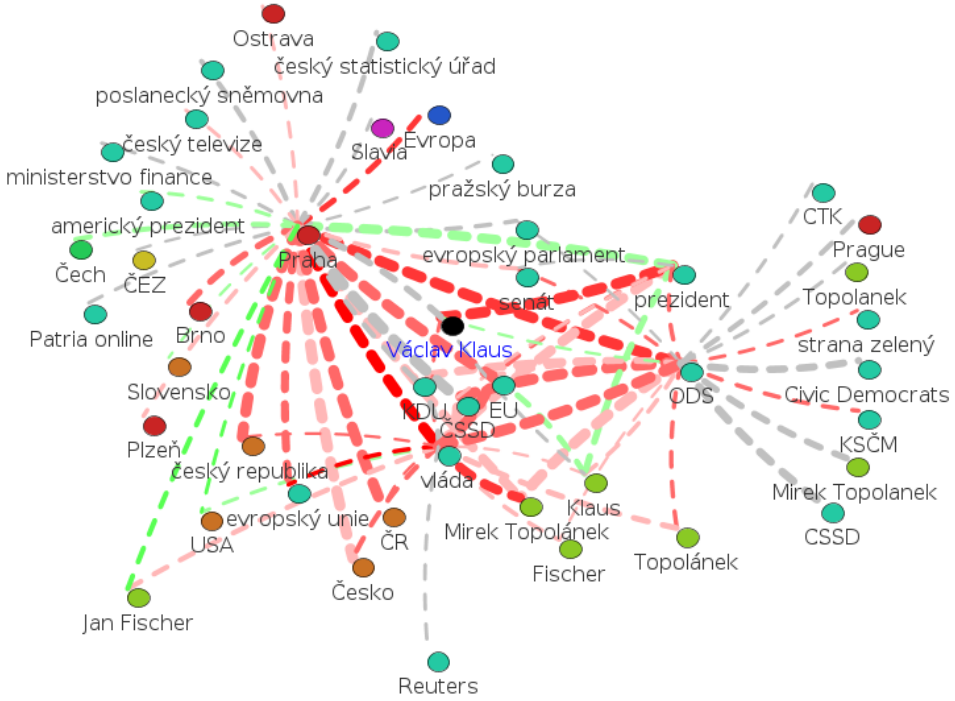


Obrázek 4.1: Rozvržení většího grafu

4.2 Konkrétní příklady

Nyní se podívejme na několik konkrétních případů, na nichž se zaměříme na zhodnocení správnosti a relevance výsledků analýzy.

Obrázek 4.2 ukazuje graf relací pro entitu *Václav Klaus* s hloubkou prohledávání *2* a limitem pro minimální společný výskyt *450*, což znamená již vysoce filtrovaná data.



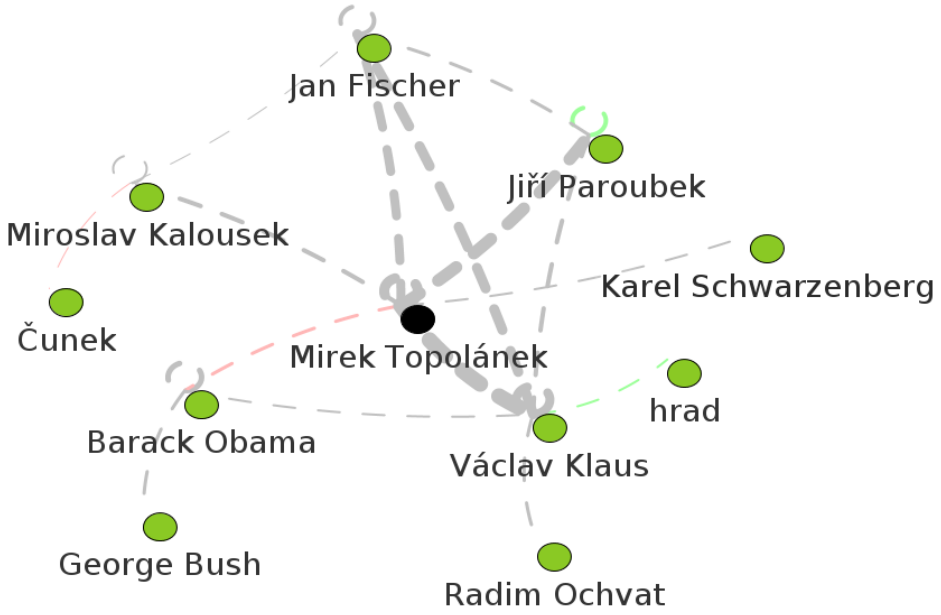
Obrázek 4.2: Příklad grafu se startovacím vrcholem *Václav Klaus*

Na tomto příkladu je vidět, že entita Václav Klaus je nejsilněji propojena s entitami *prezident*, *Praha*, *vláda* a *ODS*, což je logické a očekávané. Více překvapuje obarvení hran. Zatímco s *ODS* by zelená spojnice představovala pozitivní polaritu vztahu (pravděpodobně by takto byla označena i lidmi) a s entitou *Praha* šedivá barva znamená neutrální výsledný vztah poukazující zřejmě na nepřítomnost takových vět, jejichž charakter by umožňoval analýzu sentimentu, červená čára vedoucí k entitě *prezident* značně překvapí. Podobnou nezvyklost můžeme pozorovat i mezi entitami *Praha* a *ČR*, *Praha* a *Česko* a *Praha* a *český republika*. Lze se tak domnívat, že by bylo vhodné použít k analýze sentimentu nástroj, který by více odpovídal povaze dat,

nebo tuto část zcela pominout. Její výsledky jsou očividně diskutabilní a její užitečnost pro aplikaci taktéž.

Také si zde můžeme všimnout nedokonalostí ve stupních datech zjevných z vyjmenovaných relací. První je samozřejmě ta, že tyto dva názvy (*ČR* a *český republika*) jsou zástupci jedné a téže reálné entity. Druhou je chybný tvar názvu státu *Česká republika* způsobenou lemmatizací.

Abychom si demonstrovali užitečnost aplikování filtrů, zkusíme nyní vyhledat entitu *Mirek Topolánek* a její relace pouze s osobami. Snížíme minimální hranici společného výskytu na 150. Kdybychom takový limit nastavili bez jakéhokoliv filtru, získali bychom rozsáhlý graf, v němž by se dalo jen stěží orientovat. Přitom omezovat se pouze četností společného výskytu by bylo nepraktické z důvodu ignorování některých informací. Například investigativní reportér by tak byl ochuzen o informaci, že se nějaký lobbista v několika málo případech setkal s některým vysoce postaveným politikem a podobně, reportér rubriky sport by se nedozvěděl, že hráč párkrát nastoupil za jiný tým a podobně. Na obrázku 4.3 lze vidět účinnost filtru.



Obrázek 4.3: Filtrovaná data

5 Závěr

Automatické zpracování textu se v době internetu stalo zajímavou a lukrativní oblastí informačních technologií. Získávání dat je fenoménem dnešní doby a stále se hledají nové metody jejich nalezení a zpracování. Cílem projektu proto bylo, mimo seznámení se s problematikou rozpoznávání pojmenovaných entit v textu, navrhnout aplikaci, která analyzuje předzpracované zpravodajské texty a na základě tohoto rozboru vytvoří sociální síť parametrizovanou četností společného výskytu entit a tuto síť graficky znázorní. V souvislosti s tím bylo úkolem porovnat možnosti, které pro tento účel jsou k dispozici.

Sociální síť je bezchybně vytvořena a reprezentována pomocí seznamu sousednosti v paměti počítače bez ohledu na jazyk vstupních dat. Perzistence analyzovaných dat je zajištěna ukládáním do souboru na disk, aby bylo urychleno spuštění aplikace a zároveň aby bylo umožněno použití externích nástrojů pro vizualizaci grafu.

Prostředky pro vykreslování grafů byly porovnány a na základě toho byl vybrán nástroj JUNG použitý vytvořenou aplikací, který může být modulárně nahrazen. Vizualizace umožňuje efektivní zprostředkování informací uživateli, který má možnost ve vytvořené síti vyhledávat a tvořit její požadované podmnožiny.

Výsledný program splňuje zadání, ačkoliv aby se stal v praxi použitelnějším, bylo by velmi vhodné jej doplnit o některé další funkce. Jednou z nich by mohla být možnost odkazovat přímo z vykresleného grafu na související texty.

Literatura

[1] *Jackson, P.*; *Moulinier, I.*: **Natural Language Processing for Online Applications**. John Benjamins B.V., 2007. ISBN: 978-90-272-4993-7

[2] *O'Neil, J.*: **Doing Things with Words, Part Two: Sentence Boundary Detection**. Citováno 14.4.2014. Dostupné z: `http://www.attivio.com/blog/57-unified-information-access/263-doing-things-with-words-part-two-sentence-boundary-detection.html`.

[3] *Palátová, H.*; *Grác, M.*: **Segmentace textu na věty**. Hradec Králové: Gaudeamus, 2012. 8 pp. ISBN 978-80-7435-243-0

[4] *Lovins, J. B.*: **Development of a Stemming Algorithm**. Mechanical Translation and Computational Linguistics 11, 1968, str. 22-31.

[5] *Kanis, J.*; *Skorkovská, L.*: **Comparison of different lemmatization approaches through the means of information retrieval performance**. In Text, Speech and Dialogue (pp. 93-100). Springer Berlin Heidelberg, 2010.

[6] *Matoušek, J.*; *Nešetřil, J.*: **Kapitoly z diskrétní matematiky**. 4., upr. a dopl. vyd. V Praze: Karolinum, 2009. ISBN 978-80-246-1740-4.

[7] *Čada, R.*, *Kaiser, T.*; *Ryjáček, Z.*: **Diskrétní matematika**. Katedra matematiky FAV, Západočeská univerzita v Plzni, 2004. ISBN: 80-7082-939-7.

[8] *Sang, E. F. T. K.*; *Meulder, F. D.*: **Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition**, 2003. Citováno 28.4.2014. Dostupné z: `http://acl.ldc.upenn.edu/W/W03/W03-0419.pdf`

[9] *Pang, B.*; *Lee, L.*; *Vaithyanathan, S.*: **Thumbs up?: sentiment classification using machine learning techniques.** In Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10, str. 79-86, Association for Computational Linguistics, 2002.

A Uživatelská dokumentace

A.1 Požadavky aplikace

Tento program je určen pro OS Linux / Windows (doporučené minimum Windows XP) / MacOS. Pro běh programu SoNEx je zapotřebí Java ve verzi alespoň 1.6, nejlépe však 1.7.

Minimální operační paměť 2 GB, doporučeno 4 GB. Požadavky na pevný disk jsou dané analyzovanými daty, maximálně však vždy 1,5násobek jejich velikosti.

A.2 Data

Pro správný běh aplikace je nutno dodržet formát dat popsaný výše. Velikost jednoho souboru je bez navyšování přidělené paměti virtuálnímu stroji doporučena maximálně 250 MB a musí být vložena ve složce `data`. Aplikace po první analýze do této složky vkládá prázdný soubor pojmenovaný processed_data.true, který indikuje, že data není potřeba opět analyzovat a lze tento krok přeskočit. Při dodání nových dat je tak zapotřebí tento soubor odstranit.

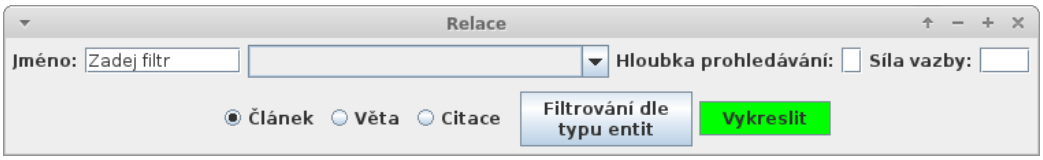
A.3 Spuštění

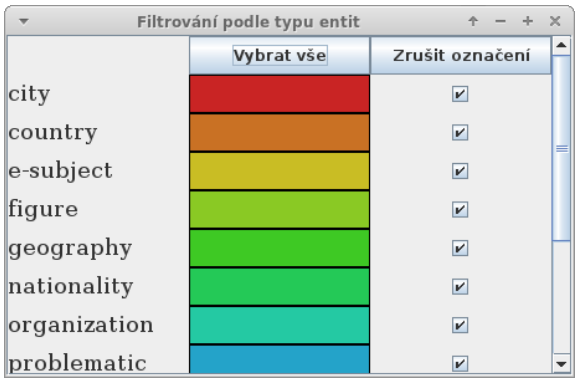
Aplikaci spustíme zadáním příkazu `java -jar SoNEx.jar` do příkazové řádky. Pokud nebyla provedena první analýza, lze v konzoli sledovat postup analýzy. Poté se otevře hlavní okno aplikace, ve kterém lze snadno nadefinovat hledané informace.

A.4 Ovládání

Do pole Jméno se zadávají počáteční písmena jména (nebo celý název) hledané pojmenované entity. V roletce napravo se vyfiltrují odpovídající entity. Ta která je vybraná se stane centrem vyhledávání. Další pole vyžaduje zadání hloubky, do které se bude prohledávat vytvořený graf. Do posledního pole patří minimální síla vazby, do které se budou vazby mezi entitami ignorovat.

Výběrem mezi tlačítky *Článek – Věta – Citace* lze zvolit požadovaný typ relace. Poslední možností filtru je vybrat zaškrtnutím typy entit, které budou brány v potaz. Po vyplnění všech polí stačí stisknout tlačítko *Vykreslit*.





V samotném grafu se lze pohybovat pomocí myši. Kolečko slouží pro přiblížení či oddálení. Funkce levého tlačítka se různí podle nastavení. V roletce nad grafem lze vybrat mezi dvěma módy plátna – *Trasforming* a *Picking*. Po výběru prvního zmíněného se lze stisknutím a táhnutím pohybovat po plátně, při zvolené druhé možnosti lze vybírat uzly grafu a táhnutím je přesouvat.