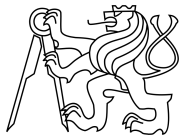
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Vyhledávání ve velkých grafech
Student:	Bc. Petr Šuták
Vedoucí:	Ing. Pavel Kordík, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2016/17

Pokyny pro vypracování

Cílem práce je navrhnout a implementovat graficky zadávané vyhledávání v grafu a podkladových datových zdrojích ve vizualizačním nástroji SVAT, který vznikl z opensourcového nástroje Gephi. Uzly v grafu jsou různých typů, které mají různé atributy, a vyhledávání musí tyto skutečnosti zohledňovat.

Mějme například databázi s daty o klientech, firmách, účtech, transakcích, sídlech firem. Informace jsou popsány jejich semantickým významem, tj. osoba má parametry rodné číslo, jméno, příjmení, účet má číslo účtu, měnu. V grafu vzniklém z těchto dat (v nástroji jsou definovány vazby mezi entitami) budeme využívat těchto semantických informací. Uživatel tedy může chtít vyhledat osoby, které mají účet v cizí měně, bydlí v Praze a posílají peníze na definovaný účet.

Toto hledání půjde provádět jak nad načteným grafem, tak nad datovými zdroji. Algoritmus vyhledávání v grafu bude optimalizován tak, aby byl použitelný i pro grafy nad 100 000 uzlů, a pro uživatele bez znalosti jazyka SQL.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 4. dubna 2016

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

Diplomová práce

Vyhledávání ve velkých grafech

Bc. Petr Šuták

Vedoucí práce: Ing. Pavel Kordík, Ph.D.

10. května 2016

Poděkování

Tímto bych chtěl poděkovat zejména Ing. Marku Sušickému za ochotu a pomoc při realizaci této práce. Dále bych chtěl poděkovat rodině a přátelům za podporu, kterou mi během studia poskytovali.

Prohlášení

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

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

V Praze dne 10. května 2016 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Šuták, Petr. *Vyhledávání ve velkých grafech.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Cílem této práce je prozkoumat metody pro vyhledávání v grafech a na základě jejich analýzy poté vybrat vhodný algoritmus, který bude součástí implementace vizuálního vyhledávání nad grafy. Tento modul bude součástí nástroje SVAT, což je produkt sloužící k vizuální analýze dat. Na základě požadavků na systém je navrženo řešení, které bude dále otestováno. Výstupem práce je funkční modul, připravený pro použití v průmyslu.

Klíčová slova Graf, Podgraf, Vyhledávání, Java, Algoritmus, Vflib

Abstract

The aim of this work is to explore methods for searching in large graphs. Based on their analysis, then select a suitable algorithm to be part of the implementation of a visual search over graphs. Search modul will be part of tool SVAT, which is product used for visual data analysis. Based on the requirements for the system, there will be proposed a solution, which will be tested for its correctness. The outcome of this work will be functional module, ready for use in industry.

Keywords Graph, Subgraph, Searching, Java, Algorithm, Vflib

Obsah

Úvod **1**

1 Popis problému **3**
1.1 Problém hledání vzorů v grafech v praxi 3
1.2 Charakteristika reálných grafů 5

2 Teoretické podklady **9**
2.1 Matematická definice grafu 9
2.2 Podgraf 9
2.3 Stupeň vrcholu 10
2.4 Matice sousednosti 10
2.5 Homomorfismus 10
2.6 Izomorfismus grafů 10
2.7 Problém izomorfismu podgrafu 11
2.8 Indukovaný izomorfismus podgrafů 11
2.9 Epimorfismus grafů 11
2.10 Monomorfismus grafů 12
2.11 Backtracking 13

3 Algoritmy pro vyhledávání v grafech **15**
3.1 Algoritmy pro přesné vyhledávání 15
3.2 Algoritmy pro přibližné mapování 26

4 Analýza a návrh **29**
4.1 Existující řešení 29
4.2 Cílová skupina 33
4.3 Funkční a nefunkční požadavky 34

5 Implementace **41**
5.1 Implementace modulu pro vyhledávání 41

5.2 Zdrojové kódy 52

6 Možnosti rozšíření **53**

Závěr **55**

Literatura **57**

A Seznam použitých zkratek **59**

B Obsah přiloženého CD **61**

Seznam obrázků

1.1 Webový graf znázorňující odkazy na stránky wikipedia.com převzato z [1] 4
1.2 Graf osob zmíněných v článcích deníku The Guardian, převzato z [2] 5
1.3 Ukázka mapy teroristické sítě z 11.9. 2001, převzato z [3] 7

2.1 Příklad indukovaného izomorfismu podgrafů s mapováním $f = \{(1, X), (2, Y), (3, Z)\}$ 11
2.2 Příklad epimorfismu s mapováním $f = \{(1, X), (2, Y), (3, X)\}$ . . . 12
2.3 Příklad částečného izomorfismu s mapováním $f = \{(1, X), (2, Y), (3, Z)\}$ 12

3.1 Rozdělení algoritmů pro vyhledávání v grafech [4] 16
3.2 Příklad vyhledávání podgrafu Ullmanovým algoritmem 20
3.3 Příklad ve kterém je podgraf částečný ale není indukovaný 23
3.4 Příklad ve kterém není splněna druhá prořezávací podmínka 23
3.5 Příklad ve kterém není splněna třetí prořezávací podmínka 23

4.1 Ukázka grafu v Gephi 31
4.2 Úvodní obrazovka programu SVAT 32
4.3 Doménový model 36
4.4 Případy užití modulu 37
4.5 Obrazovka před otevřením nástroje pro vyhledávání 38
4.6 Základní rozložení oken pro vyhledávání 38
4.7 Přidávání atributu k uzlům 39
4.8 Výsledek vyhledávání 39

5.1 Obrazovka po spuštění modulu pro vyhledávání 43
5.2 Ukázka použití neznámé entity 44
5.3 Ukázka okna sloužícího pro vyhledávání 45
5.4 Přiřazení atributu jednotlivým uzlům 45
5.5 Ukázka funkce pravítka 46

5.6 Ukázka vyhledání daného podgrafu 47
5.7 Graf, na kterém byla kontrolována správnost vyhledávacího algoritmu 48
5.8 Skupina jednoduchých vzorů pro testování správnosti algoritmu . . 48
5.9 Složitější vzory pro testování správnosti algoritmu 49
5.10 Příklad vzorů s více řešeními 49

Seznam tabulek

3.1 Porovnání složitostí Ullmanova algoritmu a Vflib (převzato z [5]) . 25

Úvod

Grafy jsou jednou z hlavních datových struktur sloužících k reprezentaci reálných objektů. Objevují se v mnoha odvětvích, například v sociálních sítích, v rozpoznávání vzorů nebo počítačovém vidění. S vývojem grafových modelů se vyhledávání v nich stává jedním z hlavních problémů, které je zapotřebí řešit. Tato práce se tímto problémem zabývá a snaží se nalézt co nejlepší řešení, které umožní toto vyhledávání provádět v přijatelném čase a s přesnými výsledky.

Na začátku práce je představen problém vyhledávání v grafech na několika demonstrativních případech, z nichž jsou odvozeny některé vlastnosti reálných grafů, které je nutné brát v úvahu.

Dále jsou představeny základní pojmy, které se v teorii grafů používají. Na základě těchto teoretických předpokladů jsou představeny algoritmy, které se v současné době pro vyhledávání používají. Z nich je vybrán jeden, který nejlépe odpovídá požadavkům zadavatele této práce.

Následující část práce se zabývá analýzou a návrhem modulu pro vyhledávání, který bude implementován jako součást vizualizačního nástroje SVAT. Jsou zde porovnána již existující řešení a jejich výhody a nevýhody. Dále jsou rozebrány požadavky na vyhledávací modul a návrh uživatelského rozhraní, na který zadavatel práce klade největší důraz.

V implementační části je nakonec popsána samotna implementace a také všechny druhy testů, kterými byl vyhledávací modul testován.

Poslední kapitola se zabývá možnými rozšířeními práce a také shrnuje dosažené výsledky vzhledem ke stanoveným cílům.

Kapitola 1

Popis problému

1.1 Problém hledání vzorů v grafech v praxi

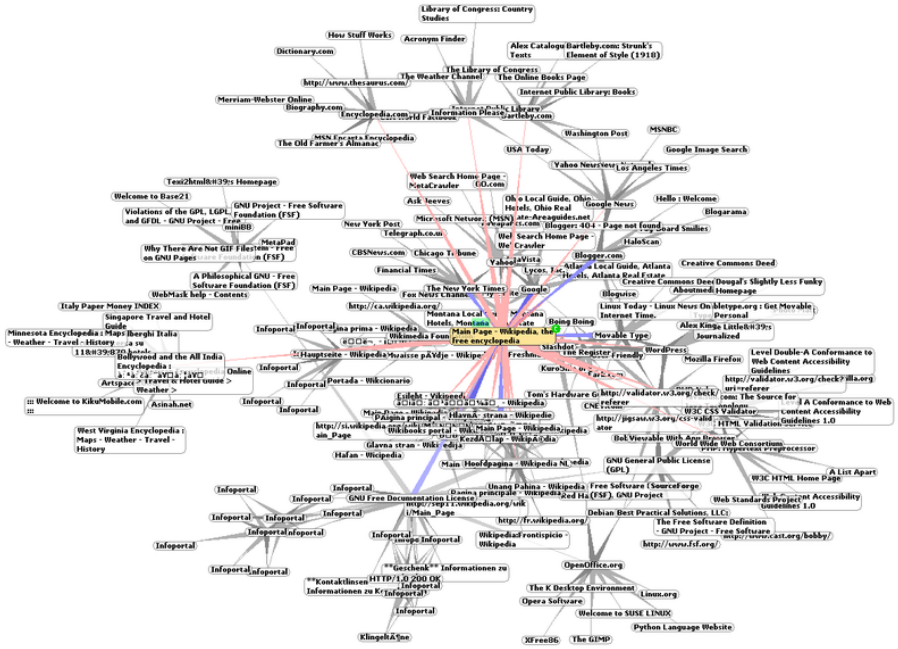
Vyhledávání v grafu je využíváno v mnoha reálných oblastech. Cílem vyhledávání je zjistit, zda jsou dva dané grafy podobné na základě nějaké metriky a pokud jsou, tak nalézt všechny shody jednoho (vzorového) grafu v druhém (cílovém) grafu. Studie [6] uvádí mnoho demonstrativních okruhů, ve kterých je řešení tohoto problému často aplikováno. V následujících podkapitolách jsou uvedené některé z nich.

1.1.1 Detekce zrcadlových webů

odle studie [7] je 30% až 36% obsahu webu složeno z duplicitních stránek. Takové stránky je vhondné detekovat, protože jejich detekce může vést například ke zlepšení page ranku takovým způsobem, že nedává vetší váhu stránkám se stejným obsahem (ale jinou adresou). Pro řešení tohoto problému je web reprezentován jako graf, ve kterém jsou uzly zastoupeny jednotlivými weby a hrany mezi nimi představují odkazy mezi stránkami viz. 1.1. Stránky, které představují plagiatorství, jsou poté jednoduše nalezeny tak, že mají stejné hrany jako originální zdroj. K vyhledávání takových shod mohou být dost dobře použity i algoritmy pro přibližné párování, jelikož někdy může být zkopírována pouze část daného webu a tedy toto mapování může být jen částečné.

1.1.2 Social matching

S rozvojem sociálních sítí (např. Facebook), doporučovacích a mediálních sítí (např. Amazon, Youtube) vznikly sociální grafy, které slouží k jejich reprezentaci. Obecně každý uzel v grafu reprezentuje osobu nebo věc a hrana reprezentuje vztah s ostatními (např. přátelství, doporučení, nákup). Vzhledem k vyhledávacím požadavkům uživatelů, které jsou obecně zastoupeny grafy, social matching [8] je proces, jehož účelem je najít podstruktury v grafu, které



Obrázek 1.1: Webový graf znázorňující odkazy na stránky wikipedia.com převzato z [1]

tyto požadavky splňují. Hledání přátel na sociálních sítích, doporučování skupin nebo výrobků je v podstatě social matching. Příkladem může být graf 1.2, jehož uzly přestavují jména lidí zmíněných v jakémkoli článku, který vyšel v deníku The Guardian v rozmezí 1. - 31. října 2012. Hrany mezi uzly jsou poté vytvořeny za předpokladu, že dvě konkrétní osoby byly zmíněny dohromady alespoň ve dvou různých článcích.

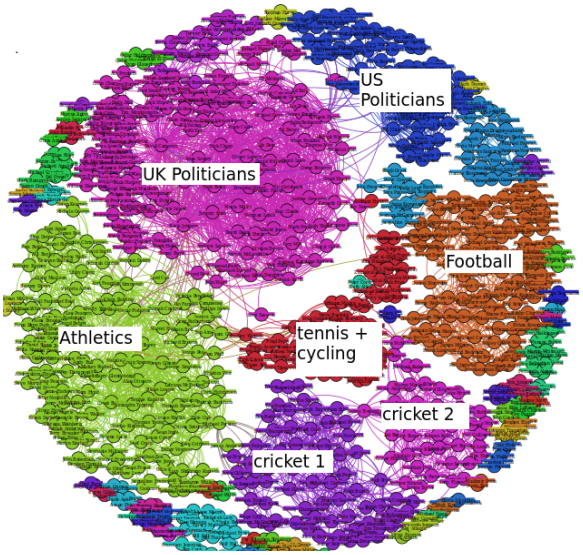
1.1.3 Detekce plagiátorství v kódu

Zdrojový kód je možné reprezentovat jako graf, ve kterém jsou metody, proměnné, přiřazení a další ukazatele reprezentovány jako jeho uzly. Hrany mezi těmito uzly představují závislost mezi nimi. Plagiátorství ja pak detekováno jako podobnost dvou grafů, které reprezentují určitý úsek kódu.

1.1.4 Ostatní možnosti použití

Využití vyhledáváni v grafu je zahrnuto v mnoha dalších odvětvích:

- Porovnávání chemických struktur
- Rozpoznávání objektů v oboru zpracování obrazu
- Detekce podvodného jednání



Obrázek 1.2: Graf osob zmíněných v článcích deníku The Guardian, převzato z [2]

- Odhalování oganizovaného zločinu
- Vyhledávání na základě klíčových slov
- Porovnávání otisku prstů

1.2 Charakteristika reálných grafů

V předchozí části je uvedeno několik demonstrativních případů, na které je možné aplikovat problém vyhledávání či párování grafů. Grafy v těchto oblastech použití mají několik společných charakteristik [6].

1. **Grafy v reálných aplikacích jsou obvykle velké**. Reálné grafy se mohou skládat až ze stovek milionů uzlů a hran (např. graf reprezentující propojení webů). Pro algoritmy, které s nimi pracují, je proto velmi důležitá jejich rychlost. Té se dá docílit například vhodným prořezáváním prohledávaného prostoru nebo hledáním pouze přibližného a nikoli přesného řešení.

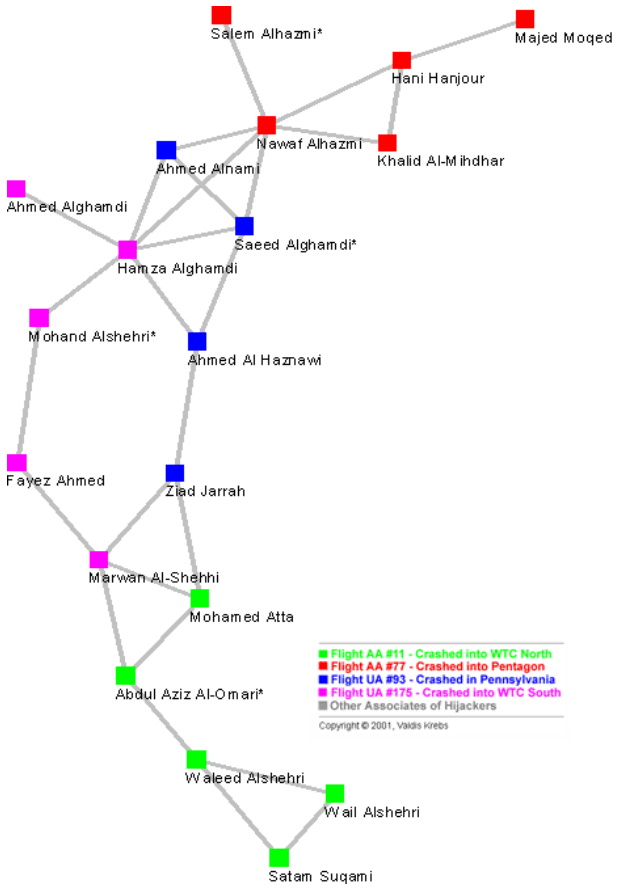
2. **Heterogenost**. To že je graf heterogenní znamená, že se skládá z uzlů a hran mnoha různých typů. Program SVAT, pro který je vyhledávací modul implementován například slouží bankám pro detekci podvodného jednání. Graf, ve kterém je toto jednání detekováno, se může skládat z osob, firem, účtů a dalších entit. Hrany poté mohou reprezentovat vztah mezi osobami nebo třeba peněžní toky mezi účty.

3. **Dynamičnost**. Grafy v reálných aplikacích se neustále mění. Jako příklad je možné opět uvést graf, který reprezentuje zákazníky banky, jejich účty a peněžní toky mezi nimi. Tento graf se mění s každou provedenou transakcí, každým nově vzniklým účtem a nově zaregistrovaným zákazníkem. Pro algoritmy, které s často se menícími grafy pracují, je proto výzva tento problém zvládnout.

4. **Nepřímé vazby** V reálných grafech se velmi často stává, že vztahy mezi jednotlivými uzly nejsou přímé (nejsou spojeny hranou), ale toto spojení existuje přes několik dalších uzlů.

   Například při odhalování korupce, daňových deliktů a praní špinavých peněz jsou často založeny imaginární firmy, které jsou zastupovány tzv. Bílými koňmi, což jsou osoby, které mají zakrýt skutečného pachatele. Vztah mezi skutečnými pachateli této trestné činnosti je tedy nepřímý a je nutné brát v úvahu i takové vazby.

   Jiným příkladem může být studie teroristické sítě útoku z 9.11. [3]]. V této práci ja ukázáno, že dva teroristé, kteří se podíleli na útoku, mezi sebou nekomunikují napřímo, ale přes jednoho nebo více prostředníků. Obrázek 1.3] ukazuje, že mnoho teroristů spolupracujících na jednom útoku (znázorněno stejnými barvami) je od sebe vzdáleno 3 a více kroků. V rámci celé sítě, která měla na starosti všechny útoky, jsou vztahy mezi lidmi plánujícími různé útoky téměř nevystopovatelné. Při hledání jednotlivých útočníků je tedy nutné brát v úvahu, že mezi sebou mohou komunikovat přes několik dalších jedinců.



Obrázek 1.3: Ukázka mapy teroristické sítě z 11.9. 2001, převzato z [3]

Kapitola 2

Teoretické podklady

V této sekci jsou popsány teoretické pojmy, nutné k pochopení algoritmů sloužících pro vyhledávání v grafu. Jedná se o definici grafu a podgrafu, dále vysvětlení některých pojmů souvisejících s teorií grafů a nakonec různé typy morfismu, jejichž určování je základní myšlenkou hledání v grafu. V grafové teorii je graf rozuměn jako reprezentace množiny objektů, ve které jsou některé jejich páry spojeny čarami. Spojované objekty se v teorii grafů nazývají uzly, čáry, které jednotlivé uzly spojují se nazývají hranami. Hrany mezi jednotlivými uzly mohou být buď orientované nebo neorientované, což je v grafu znázorněno šipkami. Takový graf se nazývá orientovaný nebo neorientovaný.

2.1 Matematická definice grafu

Graf (neorientovaný) je uspořádaná dvojice $G = \langle V; E \rangle$, kde V je neprázdná množina vrcholů a E je množina hran - množina vybraných dvouprvkových podmnožiny množiny vrcholů $E \subseteq \{\{u, v\} | u, v \in V, u \neq v\}$ tzv. neorientovaných hran. Graf (orientovaný) je uspořádaná dvojice $G = \langle V; E \rangle$, kde V je neprázdná množina vrcholů a $E \subseteq V \times V$. Hranou se tedy u neorientovaných grafů rozumí dvouprvková množina $\{u, v\}$ vrcholů $u, v \subseteq V$. Poté říkáme, že hrana $\{u, v\}$ spojuje vrcholy u a v. Graf je neorintovaný, tj. na pořadí vrcholů u hrany nezáleží. U orientovaných grafů se hrana chápe jako uspořádaná dvojice $G = \langle V; E \rangle$ vrcholů $u, v \subseteq V$. Poté říkáme, že hrana vede z u do v. Graf je orientovaný, tudíž záleží na pořadí vrcholů u hrany.

2.2 Podgraf

Graf $H = (V_H, E_H)$ je podgrafem grafu $G = (V_G, E_G)$, jestliže jsou splněny tyto podmínky:

1. $V_H \subseteq V_G$

2. $E_H \subseteq E_G$
3. Hrany grafu H mají oba vrcholy v H

2.3 Stupeň vrcholu

Stupeň vrcholu V v neorientovaném grafu je počet hran vycházejících z V.

2.4 Matice sousednosti

Matice sousednosti je matematický způsob reprezentace grafu. Pro graf $G = (V, E)$ s n vrcholy $v_1 ... v_n$ je matice sousednosti čtvercová matice

$$A_G = (A_{ij})_{i,j=1}^{n} \tag{2.1}$$

definovaná předpisem:

$$a_{ij} = \begin{cases} 1 & \text{pro } \{v_i, v_j\} \in E \\ 0 & \text{jinak} \end{cases}$$

2.5 Homomorfismus

Nechť $\circ$ je binární operace na množině X, zatímco $\circ'$ je jiná binární operace na množině X'. $Homomorfismus$ $f : (X, \circ) \to (X', \circ')$ je definován jako funkce z X do X', která zachovává operaci $\circ$ na X do operace $\circ'$ na X'. V tomto smyslu platí, že $f(x \circ y) = f(x) \circ' f(y)$. Homomorfismus je někdy také nazýván morfismem. V párování grafů platí, že operátory $\circ$ a $\circ'$ jsou sousední relace mezi vrcholy grafu. Pokud je funkce f nějakým způsobem omezena, vznikají nové druhy homomorfismu:

- $izomorfismus$ pokud je funkce f bijektivní
- $epimorfismus$ pokud je funkce f surjektivní
- $monomorfismus$ pokud je funkce $f$injektivní

2.6 Izomorfismus grafů

Dva grafy jsou izomorfní, pokud mají stejný počet uzlů a stejnou strukturu. Formálně: Mezi dvěma grafy G_p a G_t je izomorfismus, pokud existuje takové mapování f, které je bijektivní funkcí a platí, že:

$$\{a, b\} \in E_p \Leftrightarrow (f(a), f(b)) \in E_t. \tag{2.2}$$

Zobrazení f poté nazýváme izomorfismus grafů G_p a G_t. Izomorfismus zachovává všechny důležité vlastnosti grafu - zobrazuje každý podgraf na izomorfní podgraf, cestu na cestu a kružnici opět na kružnici.

2.7 Problém izomorfismu podgrafu

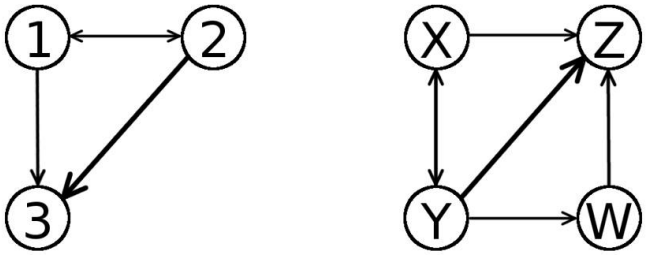
Problém izomorfismu podgrafu je velmi důležitou a zobecněnou formou několika grafových problému zahrnujících nalezení největší kliky a problému, zda graf obsahuje Hamiltonovskou kružnici, což ho řadí mezi NP-úplné problémy. Jedná se o rozhodovací problém, jehož vstupem jsou dva grafy G a H a cílem je zjistit, zda daný graf G obsahuje podgraf izomorfní ke grafu H. Někdy tento problém bývá zaměňován s problémem hledání izomorfního podgrafu, jenž klade důraz na nalezení takového podgrafu na rozdíl od prostého rozhodovacího problému, zda takový podgraf existuje.

2.8 Indukovaný izomorfismus podgrafů

Mezi dvěma grafy G_t a G_t existuje indukovaný izomorfismus podgrafů, pokud existuje takové mapování f, které je celkovou injektivní funkcí a platí, že:

$$\{a, b\} \in E_p \Leftrightarrow (f(a), f(b)) \in E_t. \tag{2.3}$$

Příklad indukovaného izomorfismu grafů je uveden na obrázku 2.1.



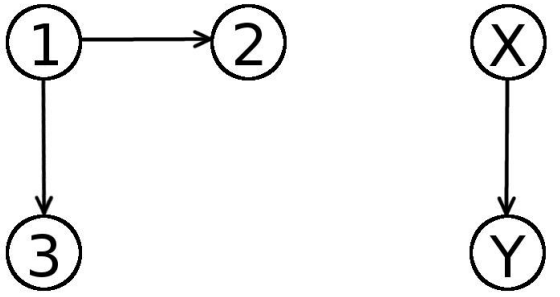
Obrázek 2.1: Příklad indukovaného izomorfismu podgrafů s mapováním $f = \{(1, X), (2, Y), (3, Z)\}$

2.9 Epimorfismus grafů

Dva grafy jsou epimorfní, pokud je možné jeden z nich soustředit (spojováním uzlů) na strukturu grafu druhého. Formálně: Mezi dvěma grafy G_p a G_t je epimorfismus, pokud existuje takové mapování f, které je surjektivní funkcí a platí, že:

$$\{a, b\} \in E_p \Rightarrow (f(a), f(b)) \in E_t. \tag{2.4}$$

Příklad epimorfismu je uveden na obrázku 2.2.



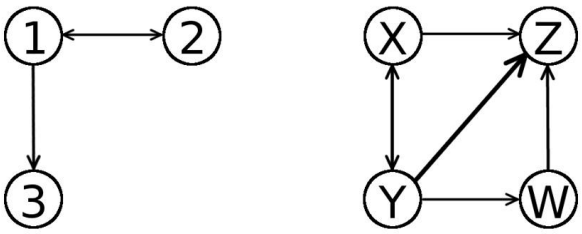
Obrázek 2.2: Příklad epimorfismu s mapováním $f = \{(1, X), (2, Y), (3, X)\}$

2.10 Monomorfismus grafů

Monomorfismus grafů neboli částečný izomorfismus podgrafů je slabší formou izomorfismu podgrafů. Formálně, dva grafy $G = (V_g, E_g)$ $H = (V_h, E_h)$ jsou monomorfní, pokud existuje mapování:

$$\phi : V_g \to V_h, \; pro \;\; které \;\; \forall v, w \in V_g : (v, w) \in E_g \Rightarrow (\phi(v), \phi(w)) \in E_h \quad (2.5)$$

Jelikož tato varianta není výslovně izomorfismus, často se místo něj v literatuře vyskytuje pojem morfismus, čímž se označuje jeho generalizace. Příklad monomorfismu grafů je uveden na obrázku 2.3. Jelikož indukovaný podgraf X, Y, Z má oproti grafu A, B, C navíc ještě jednu hranu mezi Y a Z, nejedná se indukovaný izomorfismus.



Obrázek 2.3: Příklad částečného izomorfismu s mapováním $f = \{(1, X), (2, Y), (3, Z)\}$.

2.11 Backtracking

Backtracking neboli zpětné vyhledávání, je jednou z klíčových metod sloužících pro vyhledávání v grafech. Jedná se o metodu založenou na prohledávání stavového prostoru, který je ve tvaru stromu. Tento algoritmus je vylepšením prohledávání hrubou silou tím způsobem, že je možné ořezat některé větve stavového stromu bez toho, aniž by byly přímo vyzkoušené. Obecně má tento algoritmus exponenciální složitost a jeho použití je vhodné v případech, kdy nejsou pro řešení daného problému známé algoritmy, které mají složitost polynomiální, což je například právě hledání izomorfních podgrafů.

Kapitola 3

Algoritmy pro vyhledávání v grafech

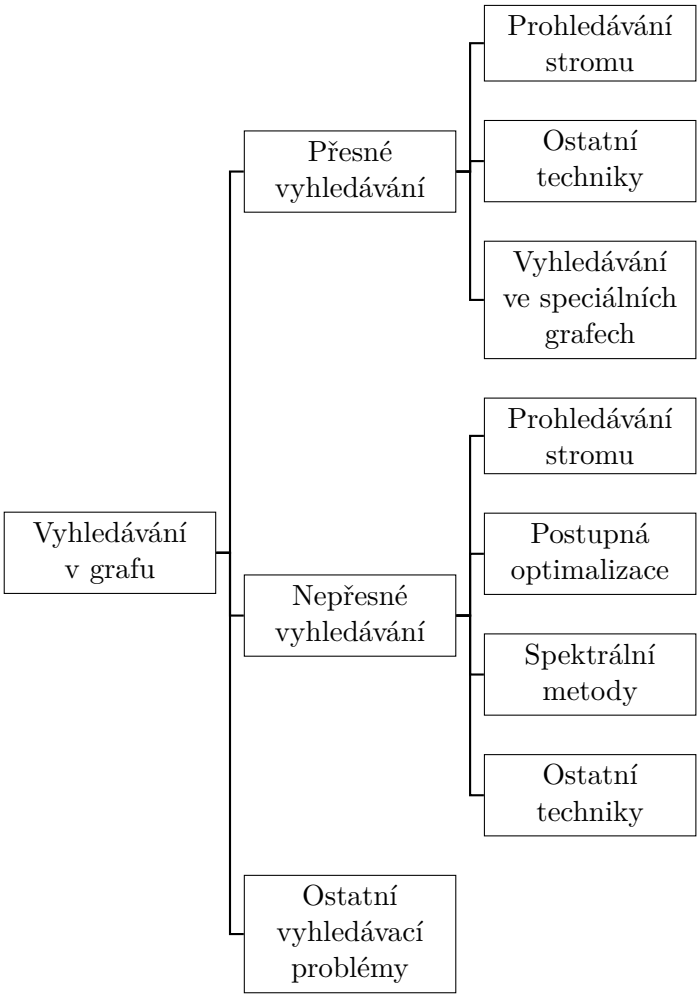
Tato část je věnována samotným algoritmům pro vyhledávání v grafu. V úvodu je uvedeno jejich rozdělení založené na principu jejich fungování a na tom, zda poskytují přesný nebo přibližný výsledek. Studie [4] představuje více jak 100 různých algoritmů pro hledání shod v grafech rozdělených podle toho, do které skupiny patří. V rámci této práce je z každe skupiny jeden nebo více algoritmů, jejichž princip je blíže popsán.

Největší pozornost je věnována grafům pro přesné vyhledávání, jelikož implementace jednoho z nich je součástí implementační části této práce. Konkrétně se jedná o Ullmanův a Vflib algoritmus, které se jeví jako nejvhodnější kandidáti pro implementaci. Tyto algoritmy jsou v této kapitole také porvnány. Princip fungování implementovaného algoritmu vflib je dále také přímo vysvětlen na konkrétních příkladech.

Ostatním skupinám algoritmů je věnován menší prostor z toho důvodu, že nesplňovaly kritéria stanovená zadavatelem této práce. Implementaci například některých metod pro nepřesné vyhledávání si je ale možné velmi dobře představit jako jedno z možných pokračování této práce, jelikož jsou vhodné na některé typy úloh, se kterými by měl vflib algoritmus problémy.

3.1 Algoritmy pro přesné vyhledávání

Tyto algoritmy charakterizuje společný fakt, kterým je to, že v mapování mezi dvěma grafy musí být zachovány hrany v tom smyslu, že pokud jsou dvě hrany v prvním grafu spojené hranou, tak také hrany v druhém grafu, na které jsou namapovány, musí být taktéž spojené hranou. V nejstriktnější formě přesného mapování - izomorfismu grafů, musí být tato podmínka splněna v obou směrech. To znamená, že shoda 1:1 musí být nalezena mezi každým uzlem prvního grafu a každým uzlem druhého grafu.



Obrázek 3.1: Rozdělení algoritmů pro vyhledávání v grafech [4]

Slabší formou přesného mapování je izomorfismus podgrafu, kde musí existovat izomorfismus mezi grafem a nějakým podgrafem druhého grafu. Nalezení právě tohoto typu mapování řeší algoritmus Vflib, který byl vybrán pro implementaci.

Ještě slabší formou je tzv. homomorfismus, který odstraňuje dále podmínku, že každý z uzlů v prvním grafu musí být namapován na různé uzly v druhém grafu, což znamená, že zde může nastat shoda N:1.

Poslední formou přesného vyhledávání podgrafu je problém, ve kterém je hledán podgraf prvního grafu, který zároveň obsahuje graf druhý. Obvykle je cílem této metody najít co největší takový podgraf, který oba grafy obsahují. Existují dvě definice maximálního společného podgrafu - v první se jako maximální podgraf bere ten s největším počtem hran a v druhém ten, který má

největší počet uzlů. Všechny tyto zmíněné problémy jsou NP-úplné.

Existují algoritmy, jejichž složitost je polynomiální, ale pouze nad určitými druhy grafů. Například pro stromy nebo pro planární grafy. Nicméně pro obecný izomorfismus takový algoritmus zatím neexistuje, a proto mají algoritmy pro přesné vyhledávání v nejhorším případě exponenciální složitost. Nicméně v mnoha aplikacích je skutečný čas výpočtů stále přijatelný díky dvěma těmto faktorům:

- Typy grafů používaných v reálných aplikacích jsou obvykle odlišné od těch, které jsou nejhorší pro dané algoritmy
- Jelikož uzly a hrany v reálných grafech mají obvykle nějaké atributy, kterými se liší od ostatních, je na základě tohoto porovnání možné dramaticky snížit čas vyhledávání

3.1.1 Algoritmy založené na prohledávání stromů

Většina algoritmů pro přesné vyhledávání je založena na technice prohledávaní stromu spojené s nějakým druhem zpětného vyhledávání. Základní myšlenkou těchto algoritmů je to, že existuje částečné mapování (na počátku obvykle prázdné), které je iterativně rozšiřováno přidáváním nových párů namapovaných uzlů. Tyto páry jsou určovány pomocí několika nutných podmínek, které zajišťují kompatibilnost vzhledem k omezením mapovaných uzlů a dále pomocí heuristických omezení, jejichž cílem je prořezat prohledávaný strom co nejdříve to je možné a tím zamezit prohledávání cest, které nevedou k žádnému řešení. Tyto algoritmy tedy buď naleznou kompletní mapování mezi grafy nebo dorazí do bodu, ze kterého není možné dané částečné mapování dále rozšířit a v tomto případě se algoritmus vrací zpět do předchozích stavů, dokud nenajde cestu, po které je možné se dále rozšiřovat. Pokud algoritmus takto projde všechny možné cesty a nenajde řešení, končí.

Výhodou těchto algoritmů je to, že mohou být jednoduše přizpůsobeny pro případy, kdy uzly nebo hrany grafu mají nějaké atributy. Tento fakt je velmi důležitý v reálných aplikacích, jelikož hraje klíčovou roli při prořezávání prohledávaného stromu a tím snižuje výpočetní čas algoritmu.

Nejjednodušší strategií prohledávání stavového prostoru je prohledávání do šířky (BFS), které vyžaduje ke svému běhu jen málo paměti a je dobře uzpůsobené pro rekurzivní volání.

3.1.2 Ullmanův algoritmus

Ullmanův algoritmus [9] je založen na metodě zpětného prohledávání spolu s kombinací dopředného testování. Jedná se o rozšíření prohledávání do šířky, které je obohaceno o prořezávání možných výsledků. Tato prořezávací procedura je založena na tom, že pokud jsou v hledaném grafu $G_p = (V_p, E_p)$

dva uzly sousední, mohou být namapovány také pouze na sousední uzly v prohledávaném grafu $G_t = (V_t, E_t)$.

Shoda mezi hledaným podgrafem a prohledávaným grafem je reprezentována maticově. Rozměr této matice je tedy n_p x n_t a tato matice M je binární (obsahuje pouze 0 a 1) a je sestrojena následovně:

$$M^0 = m_{ij}^0 = \begin{cases} 1 & \text{pokud je možné zobrazení } v_i \in V_p \text{ na } v_j \in V_t \\ 0 & \text{jinak} \end{cases}$$

1 se tedy do matice na pozici [i.j] zapíše jen tehdy, pokud pro uzel v_i platí že má menší nebo stejný stupeň jako uzel v_j. Jedničky v této matici tedy označují ty uzly, které splňují podmínku izomorfismu. Aby byl celý graf izomorfní a tím pádem bylo nalezené řešení, je nutné aby každý řádek této matice obsahoval alespoň jednu 1. Pokud ji neobsahuje, o izomorfismus se nejedná algoritmus tedy může skončit. Další podmínkou která je algoritmem kontrolována je mapování sousedních uzlů. Tato podmínka kontroluje, že sousedící uzly v hledaném grafu musí být namapovány na sousedící uzly v prohledávaném grafu. Pseudokód algoritmu je znázorněn v 1. Pro správné pochopení funkce algoritmu je nutné definovat několik proměnných. Vektory H, M_v a proměnná d jsou používány pro zpětné vyhledávání. Vektor F a proměnná k jsou používány k zajištění izomorfního zobrazení [5].

- proměnná d představuje aktuální hloubku v prohledávaném stromu
- proměnná k představuje poslední sloupec vybraný pro aktuální řádek
- matice M, která reprezentuje aktuální matici
- vektor $F =< F_1, ..., F_{n_p} >$ kde $F_i = 1 \Leftrightarrow i$tý sloupec byl vybrán
- vektor $H =< H_1, ..., H_{n_p} >$ kde $H_i = j \Leftrightarrow j$tý sloupec byl vybrán v hloubce i prohledávaného stromu
- vektor $M_v =< m_1, ..., m_{n_p} >$ kde M_i je poslední matice generovaná v hloubce i

Algoritmus (1) představuje pseudokód. Iniciační fáze algoritmu začíná na řádku 3. Řádek 4 říká, že žádný sloupec ještě nebyl vybrán. Smyčka začínající na řádku 5 ma za úkol samotné prohledávání a končí, pokud byla dosažena hloubka prohledávání 0, což znamená, že všechny 1 v prvním řádku M byly nalezeny. Řádek 6 vyhledává sloupec k v řádku d, který zatím nebyl vybrán s tím, že vynechává ty, které nesplňují podmínky zjemňující funkce. Zjemňující funkce zde slouží pro dopředné testování toho, že sousedící uzly hledaného grafu mohou být namapovány pouze na sousedící uzly prohledávaného grafu. Pokud je nalezen sloupec splňující zjemňující podmínku, jsou poté všechny

Algorithm 1 Ullmanův algoritmus (převzato z [5])

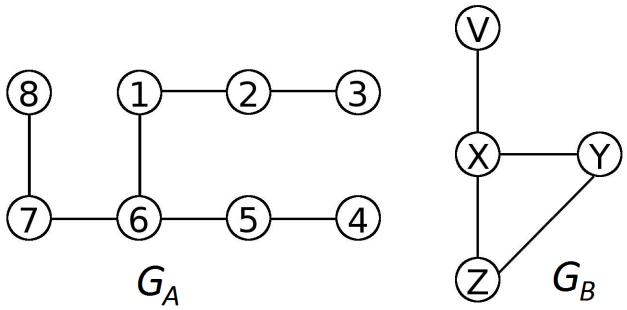
1: **Vstup:**
   Dvě matice sousednosti P a T, počáteční matice M^0
2: **Výstup:**
   $fail$ nebo $|N_p| * |N_t|$ matice M, reprezentující funkci izomorfismu podgrafu
3: $M \leftarrow M^0; d \leftarrow 1; < H_1 \leftarrow 0$;k$\leftarrow 0; backtrack \leftarrow true$;
4: **for** i in [1,$|N_p|$] **do** $F_i \leftarrow 0$
5: **while** $d \neq 0$ **do**
6: **if** *refine(M,P,T)* $\wedge (\exists k : m_{d,k} = 1 \wedge F_k = 0)$ **then**
7: $\forall j \neq k : m_{d,j} \leftarrow 0$
8: **if** $d = |N_p| \wedge$ *zobrazení je možné* **then**
9: $return\ M$
10: $backtrack \leftarrow false$
11: **if** *backtrack* **then**
12: $F_k \leftarrow 0; d \leftarrow d - 1$
13: **if** $d > 0$ **then**
14: $M \leftarrow M_d; k \leftarrow H_d$;
15: **else**
16: $H_d \leftarrow k; F_k \leftarrow 1; M_d \leftarrow M; d \leftarrow d + 1$
return $fail$

ostatní 1 v řádku d nastaveny na nulu. Pokud takový sloupec nalezen není, algoritmus se vrací zpět. Pokud je dosaženo konečného stavu a podmínka izomorfního zobrazení je splněna, algoritmus končí a vrací matici M, neboť našel hledaný podgraf. Řádek 10 pseudokódu zabraňuje zpětnému prohledávání, pokud nebyl v posledním kroku nalezen sloupec k. Pokud je vybráno zpětné prohledávání, výběr aktuálního sloupce k je zrušen, hloubka prohledávání je zmenšena o 1 a matice M předchozí hloubky je obnovena spolu se sloupcem k z předchozího kroku. Pokud zpětné prohledávání vybráno není, aktuální stav je uložen. Nakonec, pokud skončí smyčka while, byly prohledány všechny možné stavy a jelikož algoritmus nenašel řešení, vrací $fail$.

3.1.2.1 Ukázka vyhledávání podgrafu Ullmanovým algoritmem

Na obrázku 3.2 jsou dva grafy G_A a G_B a algoritmus se snaží v grafu G_A najít podgraf izomorfní ke G_B. Na počátku jsou kandidáti pro mapování na uzel $Z = \{1, 2, 5, 6, 7\}$, na uzel $Y = \{1, 2, 5, 6, 7\}$, na uzel $X = \{6\}$ a na uzel V to jsou všechny uzly grafu G_A. Po tomto počátečním mapování se provede zjemňující funkce, která je základní složkou Ullmanova algoritmu. To znamená, že pro jakýkoli uzel A grafu G_A, který je mezi kandidáty pro uzel B grafu G_B, musí mít každý ze sousedních uzlů vrcholu B alespoň jednoho kandidáta mezi uzly vrcholu A. Pokud tato podmínka není splněna, je uzel A odstraněn z

množiny kandidátů uzlu B. Tato podmínka je opakována do té doby, dokud není možné odstranit žádný další uzel. Například uzel 8 je mezi kandidáty na uzel V. Přestože X je sousedním uzlem V, žádný ze sousedů 8(to je pouze 7) není mezi kandidáty X(to je pouze 6). Z tohoto důvodu není možné namapovat uzel V na 8 protože hrana $\{X, V\}$ by byla namapována na neexistující hranu. Proto je možné bezpečně odebrat 8 z kandidátů V. Výsledkem zjemňující procedury jsou tyto kandidáti pro dané uzly: $Z = Y = V = \{1, 5, 7\}$, $X = \{6\}$.



Obrázek 3.2: Příklad vyhledávání podgrafu Ullmanovým algoritmem

V této chvíli začne být prováděno zpětné vyhledávání. Jako první je vyzkoušeno namapování uzlu Z na uzel 1. To znamená, že je nastaven jako kandidát Z na 1 a odstraněn z ostatních množin kandidátů. Zjemňující funkce v tomto případě zjistí, že ani jeden z uzlů 5 a 7 není sousedem uzlu 1 a proto je možné oba odstranit z množiny kandidátů V. Po provedení těchto operací zůstane množina kandidátu uzlu Y prázdná, a proto se algoritmus z této větve vrátí zpět, tím že vrátí všechny změny provedené v množinách kandidátů. V dalším kroku algoritmus zkusí namapovat Z na 5. V tomto případě opět množina kandidátů uzlu Y zůstane prázdná. V posledním kroku je namapován uzel Z na 7 s tím samým výsledkem. Z toho vyplývá že graf G_B není podgrafem G_A, bez toho, aby bylo nutné zkoušet všechny kombinace.

3.1.2.2 Složitost algoritmu

Nejlepším případem je přímá cesta vyhledávání od počátečního uzlu až po výsledný uzel. Nejhorším případem je prozkoumání všech uzlů v prohledávaném prostoru, který obsahuje téměř $N!$ stavů. V nejlepším i nejhorším případě je paměťová náročnost $\Theta(n_p^2 n_t)$, protože jednotlivé vektory nejsou kopírovány, ale pouze aktualizovány. Časová náročnost zjemňovací metody je $O(n_p n_t d)$. V nejlepším případě je časová složitost $O(n_p^2 n_t d)$. Nejhorší časová složitost je $O(n_p! n_p n_t d)$ [5].

3.1.3 Vflib algoritmus

Vflib algoritmus slouží stejně jako Ullmanův k vyhledávání izomorfismu grafů a podgrafů. Rozdíl mezi Ullmanovým a vflib algoritmem je ten, že vflib hledané řešení vytváří inkrementálně. Algoritmus ukazuje pseudokód (2). Pokud je s finální stav, je vráceno přiřazené mapování $M(s)$ (řádek 2). Funkce *genneigh* na řádku 3 nalezene všechny možné kandidáty na další přiřazení k danému stavu s. Pro každého takového kandidáta (n, m) je metodu $feasible$ ověřeno, zda přiřazení n k m je částečný izomorfismus a pokud je, tak je nový stav buď správný výsledek (všechny uzly z G_p jsou namapovány) nebo částečný výsledek a algoritmus porkačuje v řešení (řádek 4 a 5). Pokud může být vrchol n přiřazen k vrcholu m, stav s je rozšířen na stav s' přidáním pravě mapování (n, m) (řádek 6). Celý algoritmus je poté volán rekurzivně na nově vzniklém stavu.

Algorithm 2 vflib2 Algoritmus (převzato z [5])

```
1:  Vstup:
        dva grafy G_p a G_t, stav s
2:  Výstup:
        celkové mapování mezi dvěma grafy pokud existuje
        částečný izomorfismus, jinak fail
3:  procedure VF2(s,G_p,G_t)
4:      if M(s) pokrývá všechny G_p then return M(s)
5:      Candidate ← genneigh(s, G_p, G_t)
6:      for each(n, m) in Candidate do
7:          if feasible(s, n, m) then
8:              vytvoř nový stav s' ze stavu s přidáním mapování(n, m)
9:              vf2(s', G_p, G_t)
10:     obnov datové struktury
```

Funkce *genneigh* definuje, jak má vypadat množina kandidátů na mapování. Generuje kartézský součin sousedů podgrafu M_1 a M_2. Jako první používá kartézský součin sousedů, kteří jejichž směr je hrana vede do podgrafu, poté ty sousedy, jenž jsou spojeny s podgrafem směrem ven. V případě, že jsou tyto dvě množiny prázdné, je vygenerován kartézský součin toho, co zůstalo v obou, jak vzorovém tak prohledávaném grafu [5]:

1. $N^+(s) = N^+(M_1(s) \times N^+(M_2)(s))$
2. $N^-(s) = N^-(M_1(s) \times N^-(M_2)(s))$ pokud $N^+(s)$ je prázdná
3. $N_1 - M_1(s) \times (N_2 - M_2(s))$ pokud $N^+(s)$ a $N^-(s)$ jsou prázdné

Metoda $feasible$ vrací *true* nebo $false$ na základě toho, zda dané rozšíření stavu s o mapování (n, m) splňuje, že nově vzniklý stav s' je stále částečný izomorfismus. Formálně lze tuto podmínku napsat takto:

$$\forall v \in N_1^-(n) \cap N_1(s) \;\; \exists v' \in N_2^-(m) \cap N_2(s) : \\ (v, v') \in M(s) \wedge (v, n) \in E_p \wedge (v', m) \in E_t$$

Pokud je vyhledáván indukovaný podgraf, je přidána další podmínka: každý z uzlů m' jehož hrana vede do m v částečném mapování, musí souhlasit s uzlem n' jehož hrana vede do n. Formálně takto:

$$\forall v' \in N_2^-(n) \cap N_2(s) \;\; \exists v \in N_1^-(m) \cap N_1(s) : \\ (v, v') \in M(s) \wedge (v, n) \in E_p \wedge (v', m) \in E_t$$

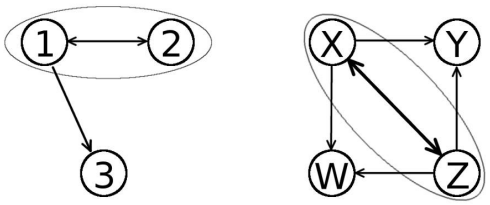
Metoda $feasible$ dále obsahuje další podmínky, které ořezávají prohledávaný prostor. První prořezávací pravidlo kontroluje zda je počet in-sousedů uzlu n, které jsou in-sousedy $M_1(s)$ je menší nebo roven počtu in-sousedů uzlu m, které jsou in-sousedy $M_2(s)$. Druhé prořezávací pravidlo kontroluje, zda počet in-sousedů uzlu n, které nejsou v $M_1(s)$ ani v $N(M_1(s))$ je menší nebo rovno počtu in-sousedů uzlu m, které nejsou v $M_2(s)$ ani v $N(M_2(s))$. Formálně lze tyto pravidla napsat takto [5]:

$$|N_1^-(n) \cap N(M_1(s))| \leq |N_2(m) \cap N(M_2(s))||N_1^-(n) \tag{3.1}$$

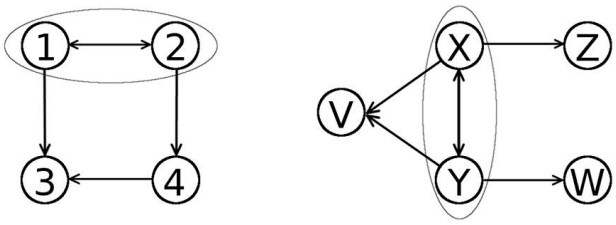
$$|N_1^-(n) \setminus (M_1(s) \cup N(M_1(s)))| \leq |N_2^-(m) \setminus (M_2(s) \cup N(M_2(s)))| \tag{3.2}$$

Tyto podmínky platí také pro out-sousedy, což dohromady dává 6 podmínek. Rozdíl mezi částečným a indukovaným izomorfismem je znázorněna na obrázku 3.3. Na tomto obrázku je $M(s) = \{(1, X), (2, Z)\}$. Metoda $feasible$ testuje kandidáta $(3, W)$. Podmínka částečného izomorfismu je splněna, protože $(1, X) \in M(s)$, $(1, 3) \in E_p$ a $(X, W) \in E_t$. Out-sousední vrchol 3 vrcholu 1 může být namapován na out-sousední vrchol W vrcholu X. Podmínka indukovaného izomorfismu přesto není splněna, protože $(2, Z) \in M(s)$ ale $(2, 3) \notin G_p$. Vrchol W má in-sousední uzel Z, který je namapován na vrchol 2, ale vrchol 2 je in-sousední uzel vrcholu 3.

Nesplnění první prořezávací podmínky je znázorněno na obr. 3.4. V tomto případě neexistuje žádné řešení. Na tomto obrázku je $M(s) = \{(1, X), (2, Y)\}$ a metoda feasibility testuje kandidáta $(3, Z)$. Podmínkačástečného izomorfismu testuje, zda hrana $(1, 3)$ může být namapována na hranu (X, Z). První prořezávací pravidlo není splněno, jelikož jediný in-sousední uzlu 3 je uzel 4. Pro uzel Z takový uzel neexistuje. Třetí prořezávací pravidlo je ověřeno, protože všechny uzly vstupního a cílového grafu jsou obsaženy buď v částečném mapování nebo v jeho přímém sousedství.

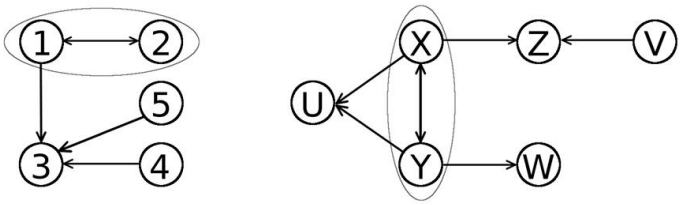


Obrázek 3.3: Příklad ve kterém je podgraf částečný ale není indukovaný



Obrázek 3.4: Příklad ve kterém není splněna druhá prořezávací podmínka

Nesplnění druhé prořezávací podmínky je znázorněna na obr. 3.5. Na tomto obrázku je $M(s) = \{(1, X), (2, Y)\}$ a je testován kandidát $(3, Z)$. Podmínka morfismu je splněna, protože hrana $(1, 3)$ může být namapována na hranu (X, Z). První prořezávací podmínka je také splněna, protože zde nejsou žádné sousední vrcholy vrcholu 3 a Z, které jsou v přímém okolí částečného mapování. Druhá prořezávací podmínka nicméně splněna není, protože zde jsou dva in-sousední uzly 4 a 5 uzlu 3, které jsou mimo částečné mapování, ale v jeho přímém sousedství a je zde pouze jeden in-sousední vrchol V vrcholu Z, který je mimo částečné mapování a zároveň v jeho přímém sousedství. V tomto případě tedy žádné řešení neexistuje.



Obrázek 3.5: Příklad ve kterém není splněna třetí prořezávací podmínka

Funkce feasibility zároveň testuje mapování atributů, pokud je vrcholy obsahují. Pokud atributy vrcholů m a n nebo atributy hran k nim přiřazených nesouhlasí, není takové řešení považováno za správné a algoritmus se vrací zpět. Vflib algoritmus používá následující datové struktury [5]:

- $core_1[n] = m \Leftrightarrow$ vrchol n grafu G_p ja namapován na vrchol m grafu G_t
- $core_2[m] = n \Leftrightarrow$ vrchol m grafu G_t ja namapován na vrchol n grafu G_p
- $in_1[n] = 1 \Leftrightarrow n \in M_1(s) \wedge n \in N^-(M_1(n))$
- $in_2[m] = 1 \Leftrightarrow m \in M_2(s) \wedge m \in N^-(M_2(n))$
- $out_1[n] = 1 \Leftrightarrow n \in M_1(s) \wedge n \in N^+(M_1(n))$
- $in_2[m] = 1 \Leftrightarrow m \in M_2(s) \wedge m \in N^+(M_2(n))$
- d je aktuální hloubka prohledávacího stromu ($|M(s)|$)
- $nbrin_1, nbrin_2, nbrout_1, nbrout_2$ jsou počty vrcholů v $N^-(M_1(s))$, atd.
- (n_1, n_2) jsou páry uzlů, které jsou přidávány do s.

3.1.3.1 Složitost algoritmu

Časová složitost metody feasible je přímo úměrná stupni vrcholů n a m. Otestování podmínky částečného a indukovaného izomorfismu je provedeno v čase $O(d_1(n) + d_2(m))$ stejně jako otestování druhé prořezávací podmínky, protože určení, zda se uzel n vyskytuje v $N^-(M_1(s))$ má časovou složitost $O(1)$ díky datové struktuře $in_1[n]$. Třetí pravidlo má také složitost $O(1)$, protože zkouška, zda $n \in M_1(s)$ je $O(1)$ díky datové struktuře $core_1[n]$. Díky tomu je časová složitost v nejlepším případě $O(n_p d)$ a v nejhorším případě $O(n_p! d)$. Paměťová náročnost je velkou přednosti algoritmu vflib, protože zpětné prohledávání je provedeno bez potřeby kopírování stavů. Zpětné prohledávání ze stavu s je pouze o tom, že se odstraní prvky $core_1[n]$ a $core_2[m]$. Tyto datové struktury zajišťují, že paměťová náročnost je po celou dobu běhu algoritmu $\Theta(n_t)$.

3.1.4 Porovnání Ullmanova algoritmu a Vflib

Co se týče časové a paměťové složitosti, Vflib algoritmus v obou případech vychází lépe. Vflib provádí dopředné zjišťování konzistence, tím že namapované uzly jsou použity pro budoucí ořezávání. Z toho vyplývá, že Vflib předčí Ullmanův algoritmus jak v nejhorším, tak v nejlepším případě. Jeho efektivnost je potvrzena experimentálními pokusy na náhodně generovaných grafech. Další důležitou výhodou vflib algoritmu je jeho nízká paměťová náročnost. Porovnání složitostí je lépe znázorněno v tabulce 3.1.

3.1.5 Související studie

Existuje mnoho algoritmů založených na izomorfismu podgrafů. Ullmanův algoritmus představuje jeden z prvních, který je založený na zpětném prohledávání. Mnoho těchto původních algoritmů bylo formulováno v kontextu hledání

	Ullman		Vflib	
	Nejhorší	Nejlepší	Nejhorší	Nejlepší
Časová složitost	$O(n_p! n_p n_t d)$	$O(n_p^2 n_t d)$	$O(n_p! d)$	$O(n_p d)$
Paměťová složitost	$\Theta(n_p^2 n_t)$	$\Theta(n_p^2 n_t)$	$\Theta(n_t)$	$\Theta(n_t)$

Tabulka 3.1: Porovnání složitostí Ullmanova algoritmu a Vflib (převzato z [5])

podstruktur v chemických složkách. Ullman je nicméně první osoba, která se zabývala problémem izomorfismu (pod)grafů jako takovým. Přestože se jedná o algoritmus starý téměř 40 let, je stále základem mnoha pokročilejších algoritmů.

Studie tákající se vflib algoritmu [10] a [11] demonstrují jeho efektivnost v porovnání s Ullmanovým algoritmem. Jeho další výhodou je veřejná publikace zdrojového kódu a velké množství experimentů, které byly s tímto algoritmem prováděny. Ze všech výše uvedených důvodu byla tedy vybrána právě implementace vflib agoritmu pro vyhledávání v grafu v implementační části této práce.

Někteří autoři navrhují použití algoritmů pro vyhledávání největšího společného podgrafu, nebo algoritmy pro hledání maximální kliky jako algoritmy pro řešení problému izomorfismu podgrafů. Přestože tyto studie jsou zajímavé, bylo dokázáno, že jejich výkon není dostatečný v porovnání s algoritmy účelně navrhnutými na problém izomorfismu podgrafů.

3.1.6 Ostatní techniky pro přesné vyhledávání

V této části je zmíněno několik algoritmů a technik, které slouží k přesnému vyhledávání, ale z různých důvodů se nehodily pro implementaci do vyhledávacího modulu nástroje SVAT.

3.1.6.1 Nauty

Nauty je algoritmus, který slouží pouze k určení toho, zda jsou dva grafy izomorfní. Je založen na teorii grup, kde využívá některých technik z této teorie k vytvoření autoformních grup u každého z grafů. Pro tyto grupy je pak vytvořeno tzv. *canonical labeling* za pomoci kterého je pak ověřen izomorfismus vstupních grafů.

Obrovská výhoda tohoto algoritmu je ta, že *canonical labeling* se počítá separátně pro každý graf. Tato vlastnost činí algoritmus užitečný v těch případech, kdy je například vzor porovnáván oproti množině grafů, jejichž *canonical labeling* byl již spočítán.

Na porovnávání vzoru proti velké množině neměnných grafů je založen také například algoritmus [12]. Tento algoritmus využívá rozkladu grafů na menší části a toho, že některé z těchto podgrafů se objevují ve více grafech z porov-

návané množiny a tím pádem se algoritmus vyhne opakovanému porovnávání hledaného grafu oproti těmto podmnožinám.

Posledním algoritmem pro přesné vyhledávání zmíněným v této práci je [13]. Tento algoritmus vytváří z databáze grafů, na které má být hledaný graf mapován, rozhodovací strom. Díky tomuto stromu je možné namapovat vstupní graf na celou knihovnu hledaných grafů v čase $O(n^2)$, kde N je velikost vstupního grafu.

3.2 Algoritmy pro přibližné mapování

Přibližné vyhledávání nebo mapování grafů je v mnoha případech vhodnější, než mapování přesné. V mnoha datech se například projevuje šum nebo jsou náchylná k chybám. Na algoritmus také může být požadováná rychlejší doba běhu, než je tomu u algoritmu pro přesné vyhledávání, jejichž časová složitost je exponenciální. Všechny tyto důvody vedly k vzniku skupiny algoritmů, které se zabývají přibližným mapováním a vyhledáváním v grafech.

Pro přibližné mapování existují dva přístupy. Častějším z nich je ten, kdy je umožněno algoritmu nenamapovat hranu mezi uzly. Druhým méně častým přístupem je možnost nenamapování uzlů. Obě tyto skupiny mají společnou jednu věc a to, že každé takové nepřesné mapování je určitým způsobem penalizováno. Cílem těchto algoritmu je tuto penalizaci co nejvíce snížit.

Dále lze tyto algoritmy rozdělit do dvou skupin podle toho, jestli dokážou nalézt globálně nebo lokálně optimální řešení.

Algoritmy, které vždy naleznou globální řešení lze z tohoto pohledu tedy považovat za rozšíření algoritmů pro přesné mapování, jelikož pokud existuje správné řešení, dokážou ho vždy naleznout.

Druhá skupina se nazývá aproximační algoritmy a ty narozdíl od první skupiny garantují nalezení pouze řešení s lokální nejnížší chybou. Jejich výhoda je naopak v tom, že bývají o dost rychlejší, než algoritmy z první skupiny, jejichž rychlost bývá dokonce nižší, než je tomu u algoritmů pro přesné vyhledávání.

3.2.1 Algoritmy založené na prohledávání stromů

Stejně jako u technik pro přesné mapování i zde existuje skupina algoritmů založená na prohledávání stromu.

Jeden z rozšířených algoritmů pro nepřesné vyhledávání založený na principu prohledávání stromu je [14]. Tento algoritmus patří do skupin optimálních, tzn. nalezne vždy nejlepší řešení . Založen je na počítání vzdálenosti grafů pomoci A* algoritmu 3.2.1.1. je založen na řešení tzv. *biparitního mapovacího problému*. Základem tohoto problému je nalézt největší mapování mezi dvěma skupinami uzlů, které tvoří bipartitní graf s tím omezením, že každý uzel může být použit maximálně jednou.

3.2.1.1 A* algoritmus

A* je algoritmus, který se používá pro vyhledávání optimálních cest v kladně ohodnocených grafech. Používá stejné principy jako Dijkstrův algoritmus, ale navíc oproti němu přidává heuristický prvek. Tato heuristická funkce odhaduje správnost postupu při hledání optimální cesty za pomoci vzdálenosti z aktuálního uzlu do uzlu konečného a zároveň musí být přípustná, což znamená, že nesmí nadhodnocovat vzdálenost k cíli [15].

3.2.2 Algoritmy založené na postupné optimalizaci

Všechny výše zmíněné algoritmy jsou založené na diskrétní optimalizaci. To znamená, že proměnné, které tyto algoritmy využívají jsou diskrétní (například celá čísla). Opakem tohoto přístupu je právě postupná optimalizace, která místo konkrétních hodnot využívá například intervaly ve kterém se hodnota dané proměnné může libovolně pohybovat. Mezi přednosti algoritmů, které tuto optimalizaci využívají patří například velká volnost při zadávání vstupních parametrů, které ovlivňují hlavní ukazatele algoritmu tedy jeho rychlost a přesnost.

První skupina algoritmů využívajících postupnou optimalizaci je založena na tzv. *relaxation labeling*. Tato myšlenka říká, že každý uzel z jednoho grafu je možné označit nějakým štítkem, který určí, ke kterému uzlu z druhého grafu patří. Postupná optimalizace poté pracuje s tím, že pro každý uzel v cílovém grafu je spočítán vektor pravděpodobností pro každý štítek. Algoritmus poté iterativně tyto hodnoty mění a tím se přibližuje optimálnějšímu řešení. Jako výsledek je poté pro každý uzel brán ten štítek, jehož pravděpodobnost je nejvyšší.

Problém těchto algoritmů spočívá v tom, že zajišťují poze shodu v jednom směru. To znamená, že každému uzlu je přiřazen nějaký nejlepší štítek, ale není zaručeno, že každý štítek je přiřazen právě jednomu uzlu.

Druhá skupina algoritmů je založena na formulaci problému pojmenovaném jako tzv. *Weighted Graph Matching Problem* (WGM), který problém shody v jednom směru odstraňuje. Řešení tohoto problému je založeno na tom, že jsou porovnány dva grafy, jejichž hranám jsou přiřazeny váhy. Toto porovnání spočívá v hledání takové permutace uzlu prvního grafu, že rozdíl mezi vahami hran prvního a druhého grafu je co nejmenší [16]. Vzhledem k tomu, že tento algoritmus nepracuje s uzly, ale hranami, vyplývá z něj omezení, že nedokáže zohlednit atributy u uzlů v grafu, pokud nějaké existují.

3.2.3 Algoritmy založené na spektrálních metodách

Spektrální metody pro porovnávání grafů jsou založené teorii, která se nazývá *Spectral graph theory*. Tato teorie je založena na prozkoumávání vlastností matic, které reprezentuje graf (matice sousednosti a Laplaceova matice).

Vlastní čísla a vektory těchto matic jsou totiž provázené s hranami v tomto grafu a říkají, jakým způsobem je graf propojený [17].

Jedním z algoritmů, který je na tomto principu založený, je algoritmus [18]. V této metodě je vektorový prostor nazývaný *graph eigenspace*, který je definován pomocí vlastních vektoru matic sousedností obou grafů a na body v tomto prostoru jsou promítány jednotlivé uzly. Dále využívá metodu shlukování, která je použita pro nalezení uzlů v grafech, které na sebe lze namapovat.

3.2.4 Ostatní metody

Do této skupiny spadají všechny algoritmy, které nejsou založeny na žádné z metod uvedených v předchozích kapitolách. Jedná se nepříklad o metody, které využívají neuronové sítě [19], genetické algoritmy [20] a další.

Kapitola 4

Analýza a návrh

4.1 Existující řešení

Spolu s tím, jak roste objem dat, která je nutné zpracovat a nějakým způsobem transformovat do uživatelsky přívětivé podoby, stoupá zároveň množství platforem, které toto umožňují. Možností jak zobrazit data je velké množství a jelikož program SVAT, pro který je vyhledávací modul v této práci implementován se zabývá vizualizací pomocí grafu, bylo také vybráno několik již existujících řešení, která data také transformují do grafové podoby. Tato řešení jsou dále vzájemně porovnána.

4.1.1 Linkurious

Linkurious [21] je platforma, která pracuje s Neo4J databázemi. Jedná se o start-up, který vznikl v roce 2013, nicméně mezi jeho klienty v současnosti patří například NASA nebo se také tato firma podílela na zpracování a vizualizaci dat v kauze Panama Papers. Co se týče možností práce se samotným grafem, Linkurious poskytuje velké množství možností, mezi kterými je například shlukování a rozbalování uzlů, filtrování hran i uzlů, vytvoření vlastních ikon, layoutů a další. Zároveň také podporuje vyhledávání uzlů a hran. Oproti funkčnosti implementované v této práci je nicméně vyhledávání omezeno pouze na jednotlivý uzel nebo hranu. Další nevýhoda týkající se Linkuriousu je to, že podporuje pouze Neo4J databáze jako datový zdroj. Naopak navíc umožňuje například vyhledat nejkratší cestu mezi dvěma uzly, což třeba v současné verzi SVATu není možné. Jednou z dalších zajímavých možností programu je tzv. geo mód, který umožňuje zobrazit grafová data na geografické mapě.

4.1.2 Maltego

Maltego [22] je nástroj sloužící pro dolování dat a jejich přehledné zobrazení a analýzu ve formě grafu. Stejně jako SVAT je i tento program zalo-

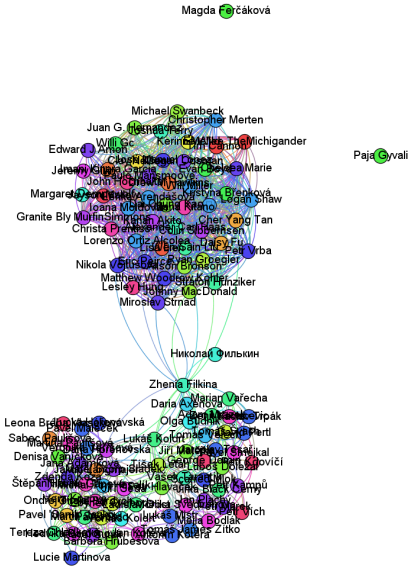
žen na NetBeans platformě. Maltego funguje na principu tzv. transformací, kdy transformuje data od různých poskytovatelů do grafové podoby. Aktuálně podporuje například vizualizaci internetové infrastruktury, Bitcoinového blockchainu, transformaci dat z různých sociálních sítí jako například Twitter, Instagram, Foursquare atd. Existuje několik verzí, z nichž je zdarma dostupná pouze verze Community Edition, která má ale některá omezení a to například zobrazení omezeného množství 10 000 uzlů nebo využití pouze některých transformací. Oproti ostatním nástrojům se Maltego zaměřuje díky svým transformacím na specifické oblasti a proto jeho využití není tak široké jako u dalších nástrojů. Zároveň zde také neexistuje možnost importovat data z různých databází, jako tomu je např. u programu SVAT. Co se týče funkce vyhledávání, stejně jako u předchozího řešení není zde možnost hledat vzor v grafu ale pouze jednu entitu podle zadaných parametrů.

4.1.3 Tulip

Dalším rozšířeným nástrojem pro vizualizaci dat je Tulip [23]. Je napsán v jazyce C++. Mezi jeho hlavní výhody oproti ostatním zde uvedeným nástrojům patří nízká paměťová náročnost i při zobrazení obrovského množství dat najednou (řádově milion uzlů a 5 milionů hran). Podobně i algoritmy, které podporují práci s grafem, jsou v Tulip optimalizovány tak, aby jejich doba běhu byla co nejkratší. Jako ostatní zde uváděné nástroje, k práci s grafem používá metody filtrování na základě hledaných parametrů, clustrování grafu a další vizualizační techniky. Zároveň podporuje import z různých typů souborů - graphviz, CSV, GEFX atd. Bohužel zde chybí možnost napojení přímo na datový zdroj, tedy databázi a stejně jako všechny zde ostatní uváděné nástroje, neposkytuje možnost vyhledávání struktur v grafu.

4.1.4 Gephi

Gephi [24]je interaktivní open-source nástroj pro vizualizaci a prozkoumávání různých druhů sítí, komplexních systémů a dynamických nebo hierarchických grafů. Je kompletně psaný v jazyku Java a založen na NetBeans platformě. Původní verze byla vyvinuta studenty na The University of Technology of Compiègne ve Francii. Jako nástroj byl použit v mnoha výzkumných projektech zahrnujících např. žurnalizmus (Vizualizace globálního propojení obsahu New York Times) nebo třeba při sledování provozu na síti Twitter při sociálních nepokojích, spolu s více tradičními předměty analýzy sítí. Obrázek 4.1 například ukazuje shlukování přátel do skupin na sociální síti Facebook. Konkrétně se jedná o přátele autora této práce, hrany mezi jednotlivými lidmi znamenají, že tito uživatelé jsou přáteli.



Obrázek 4.1: Ukázka grafu v Gephi

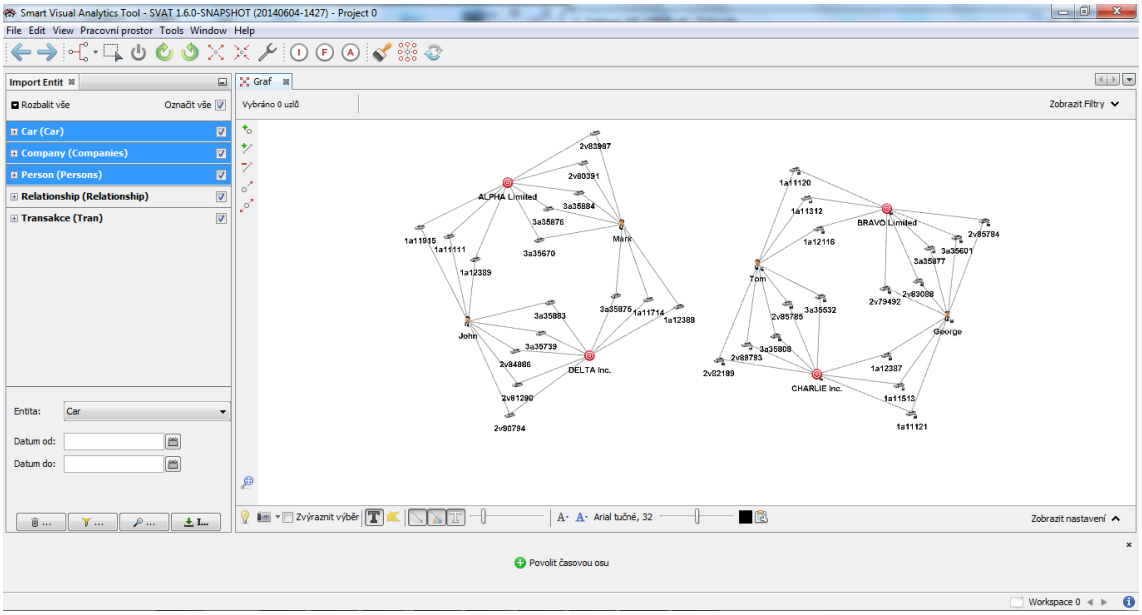
4.1.5 SVAT

SVAT (Smart Visual Analytics Tool) je komerční nástroj založený na platformě Gephi a vyvíjený firmou Profinit. Produkt se chce vyrovnat konkurenčním nástrojům a v posledních měsících prochází vývojem, který hodně ovlivňují zákazníci. Produkt je primárně orientován na podporu šetření a dohledávání ve velkých databázích, sběru dat z více zdrojů a jejich přehledném zobrazení. Podporuje mnoho datových zdrojů, například Oracle, MSSQL, MySQL, PostgreSQL, Firebird nebo Excel. SVAT se snaží konkurovat největším hráčům na trhu vizualizačních nástrojů, klade důraz především na výkon celého řešení.

SVAT je primárně určen pro vyšetřování podvodného a podezřelého jednání. Využití tak najde v bankách, pojišťovnách, leasingových společnostech, u telekomunikačních operátorů, nebo represivních složek. SVAT je možné napojit na neplacené i placené zdroje dat (insolvenční rejstřík atd.), což opět otevírá více možností jeho využití. Na obr. 4.2 je úvodní obrazovka tohoto programu. Největší plochu zabírá samotné plátno s grafem. V levé části je možné na toto plátno importovat entity ze zdrojového souboru nebo databáze. V horní části je k dispozici několik funkcí pro práci s grafem, které ovlivňují například jeho rozložení nebo filtry, díky kterým je možné zobrazit pouze ty části grafu, které splňují filtrovací podmínky.

Jak bylo zmíněno v kapitole 3, jednou z vlastností grafů v reálných aplikacích je jejich změna v průběhu času. Pro práci s takovým typem grafu slouží v nástroji SVAT funkce časové osy, která umožňuje zobrazovat, jak se graf měnil

v určitém časovém intervalu. Nutnou podmínku ke správné funkci časové osy je samozřejmě to, že alespoň některé uzly/hrany musí mít tzv. časové razítko, které ukazuje jejich vznik/zánik.



Obrázek 4.2: Úvodní obrazovka programu SVAT

4.1.6 Porovnání existujících řešení

Jelikož v současné době je kladen čím dál větší důraz na rychlou a přesnou analýzu dat, roste zároveň množství nástrojů, které se touto problematikou zabývají. Několik výše uvedených řešení bylo vybráno jako vzorek, který reprezentuje jak komerční, tak open-source scénu nástrojů pro vizualizaci dat do grafové podoby. Aby byl tento vzorek reprezentativní, byly zároveň vybrány ty platformy, které jsou aktuálně široce používané. V následujícím výčtu jsou uvedeny výhody a nevýhody jednotlivých řešení:

- Linkurious
  - Výhody: Množství možností pro práci s grafem, vlastní layout, Geo mód
  - Nevýhody: Pracuje pouze nad Neo4j databází, nepodporuje vyhledávání struktur v grafu
- Maltego

  - Výhody: Vizualizace velkého množství dat najednou, transformace - předpřipravené algoritmy pro transformaci dat do grafové podoby od různých poskytovatelů (NewsLink, SocialNet atd.)
  - Nevýhody: Nelze napojit na databázové systémy, nepodporuje vyhledávání struktur v grafu

- Tulip
  - Výhody: Opensource, množství pluginů, oproti ostatním nejrychlejší i při práci s velkým množstvím objektů (řádově miliony), možnost importu dat v různém formátu
  - Nevýhody: Nelze napojit na databázové systémy, nepodporuje vyhledávání struktur v grafu

- Gephi
  - Výhody: Opensource, množství pluginů, připojení přímo k datovému zdroji
  - Nevýhody: velké množství chyb, nepodporuje vyhledávání struktur v grafu, chybí uživatelská příručka

- SVAT
  - Výhody: Připojení přímo k datovému zdroji, podpora vyhledávání struktur v grafu
  - Nevýhody: práce s velkým objemem dat je pomalá

Z výše uvedeného nelze určit, který z nástrojů se jevil jako nejlepší. Základní nástroje pro praci s grafem (shlukování uzlů, filtry, grafické přizpůsobení layoutu a uzlů danému problému, algoritmy pro rozložení grafu atd.) poskytoval každý z nich. Rozdíl pak byl v tom, že některé například byly optimalizované na rychlost, další pak na co nejširší možnost použiti a co nelepší grafickou vizualizaci. Zajímavostí je, že ani jeden z porovnávaných nástrojů neobsahoval podporu vyhledávání v grafu podle zadaného vzoru, což je funkčnost, která byla implementována v rámci této práce do programu SVAT. Na závěr této časti tedy nelze říci, který z daných nástrojů je nejlepší, ale každý uživatel musí sám určit, jaký z nich se pro řešení jeho problému hodí nejlépe.

4.2 Cílová skupina

Nástroj SVAT slouží k detekci podvodného jednání. Využití najde například v bankovních nebo pojišťovacích institucích pří detekci falešných pojišťovacích událostí nebo uvedení nepravdivých informací při půjčkách, jelikož dokáže vizualizovat a spojit data z různých registrů, interních dat a dalších zdrojů. Další možností je například pomoc při odhalování organizovaného zločinu, je

zde možné hledat podezřelá spojení přes různé osoby, tok peněz atd. Další zajímavou možností využití programu je investigace kybernetických útoků, kdy je od útočníka posbíráno množství informací, které je následně možné vizualizovat a hledat v nich podezřelé vzory.

Z tohoto výčtu vyplývá, že uživateli mohou byt jak technicky tak netechnicky vzdělaní lidé, drtivá většina ale s vysokoškolským vzděláním. Proto je nutné modul pro vyhledávání přizpůsobit tak, aby byl vhodný pro všechny tyto skupiny.

4.3 Funkční a nefunkční požadavky

Funkční i nefunkční požadavky byly dodány zadavatelem práce v rámci několika prvních setkání. Všechny požadavky se týkají implementovaného vyhledávacího modulu a níže je uveden jejich výčet.

4.3.1 Funkční požadavky

Grafické rozhraní pro vyhledávání

Jelikož SVAT je vizualizační nástroj, tak jedním ze základních požadavků zadavatele bylo, aby i vyhledávání bylo vizualizované. To znamená, že si uživatel hledaný graf bude moci sám nakreslit za pomoci nějakého nástroje.

Vyhledáni vzoru v grafu

Základní účel modulu je právě vyhledání a zobrazení vzorů, který uživatel nakreslil (pokud takový vzor existuje). Po vyhledání by se mělo na výsledek zaostřit

Přepínání mezi řešeními

Jelikož je pravděpodobné, že pro daný vzor může existovat více nalezených řešení, modul musí umožňovat mezi těmito řešeními plynule přepínat.

Mazání uzlů a hran na plátně

Plátno, na které uživatel bude kreslit svůj hledaný vzor, bude pro snadnější ovládání umožňovat mazání libovolných už nakreslených uzlů a hran.

Editace uzlů

Na kreslícím plátně bude uživateli umožněna editace uzlů. Hrany editovat nelze z toho důvodu, že neobsahují žádnou dodatečnou informaci.

Přidání atributů k uzlům

SVAT podporuje u každého uzlu různé atributy, definované datovým zdrojem. Pro vyhledávání je taudíž také potřebné, aby bylo možné k hledanému uzlu přídat atribut a nějakou hodnotu. Například uzel typu *Osoba* má atribut *Jméno* a hodnotu *Petr*.

Editace atributů

Stejně jako u uzlů, uživateli bude umožněno editovat počet atributů a jejich hodnoty.

Funkce Jakýkoli uzel

Tato funkce představuje jeden typ uzlu, který bude dostupný pro každý graf. Jedná se o typ uzlu, který nahrazuje jakýkoli uzel v prohledávaném grafu. Tato funkce je velmi důležitá pro odhalování podvodného jednání, protože pomocí ní je možné například zjistit, jestli jsou dvě osoby v nějakém spojení a přitom nevíme jakém (například posílaly peníze na stejný účet, mají podíl ve stejné firmě, znají stejnou třetí osobu atd.). Funkce *Jakýkoli uzel* nahrazuje právě toto neurčité spojení.

4.3.2 Nefunkční požadavky

Nefunkční požadavky jsou uvedeny pouze jako výčet:

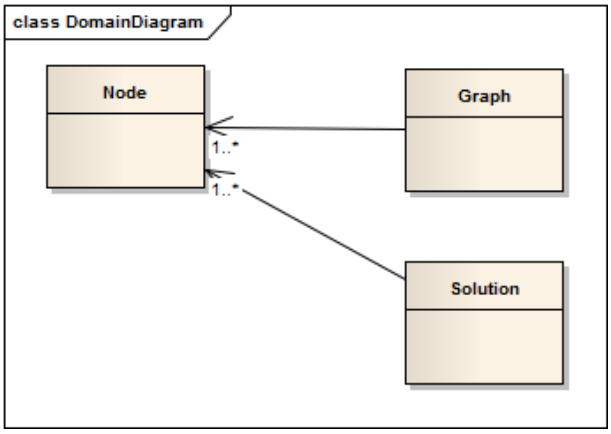
- Jednoduchost
- Uživatelská přívětivost
- Rychlost při hledání ve velkých grafech

4.3.3 Doménový diagram

Doménový model zachycuje obr. 4.3. Jak je vidět, není potřeba více jak 3 základní entity, se kterými bude modul pracovat a to jsou:

- Uzel - základní entita pro celý modul. Uzel představuje vrchol grafu, který hledáme. Entita hrana není potřeba, jelikož uzel obsahuje přímo odkaz na sousední uzly.

- Graf - graf, který se skládá z hledaných uzlů.
- Řešení - skupina výsledků, které odpovídají hledanému grafu.



Obrázek 4.3: Doménový model

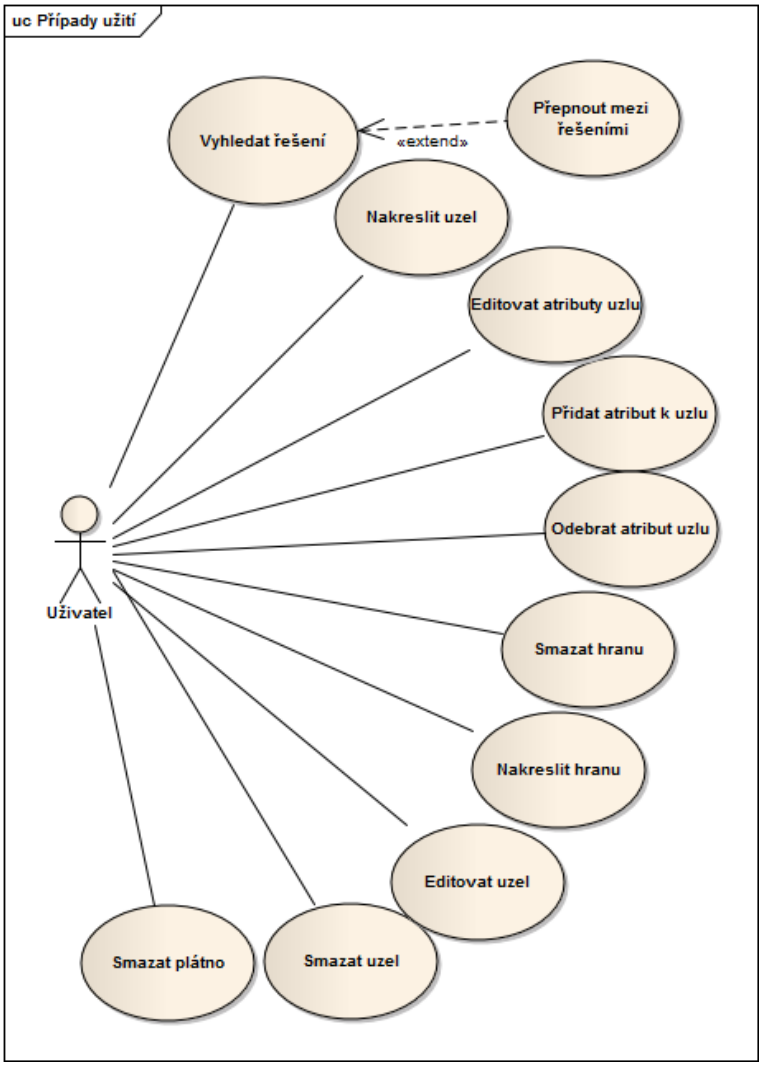
4.3.4 Případy užití

Případy užití vycházejí z funkčních požadavků. Jedná se tedy hlavně o kreslení hledaného grafu, jeho editaci a vyhledávání v cílovém grafu. Všechny případy užití je možné vidět na obr. 4.4. Vzhledem k tomu že SVAT nerozlišuje typy uživatelů, tzn. zde neexistují žádná administrátorská práva a podobně, je v diagramu případů užití pouze jedna role.

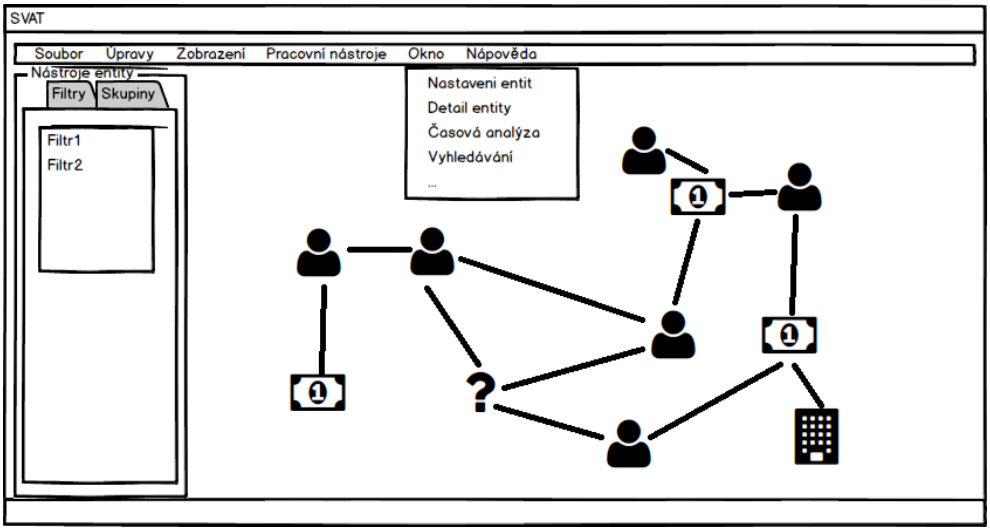
4.3.5 Návrh uživatelského rozhraní

Dva ze tří nefunkčních požadavků se týkají uživatelského rozhraní. Program SVAT obsahuje mnoho funkčností, a proto by každá nová funkcionalita měla být pro uživatele co nejjednodušší, aby nebyl nucen si pamatovat složité postupy. Požadavkem zadavatele bylo také, aby způsob vyhledávání v grafu zapadl do konceptu celé aplikace, tzn. aby byl vizualizovaný. Základní okno je vidět na obrázku 4.5. Většinu okna zde zabírá samotný graf, po stranách dále může být umístěn panel rychlých nástrojů, který zde není vyznačen. K vyhledávání bude přistupováno stejně jako k většině dalších nástrojů přes nabídku *Okno* na horní liště. Na obr. 4.6 je znázorněn návrh rozložení oken pro vyhledávací modul. V pravé části plochy se objeví paleta s entitami, které existují v načteném grafu. Tato paleta bude navíc obsahovat také entitu, představující

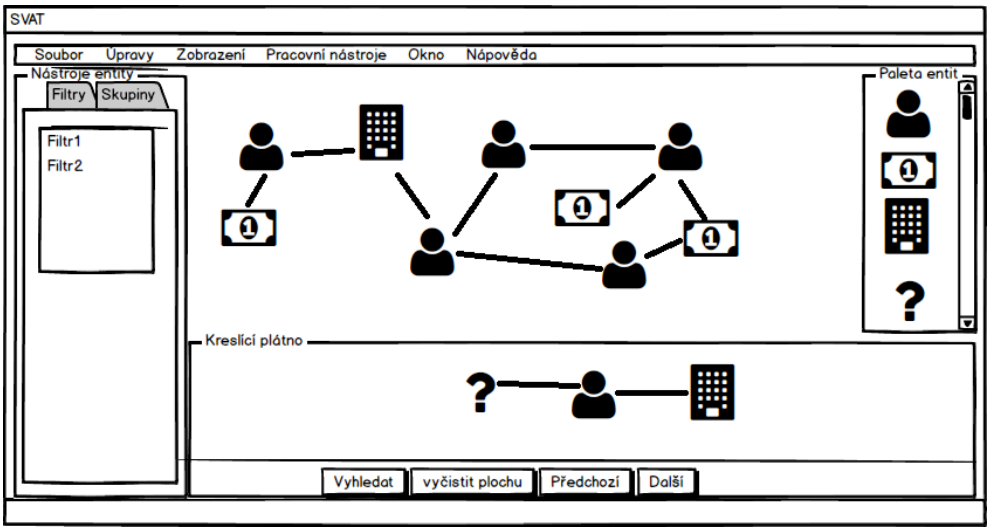
náhodný uzel (na obrázku znázorněno ikonou otazníku). V dolní části plochy bude poté plátno, na které bude uživatel přetažením z palety kreslit svůj hledaný graf. Pod plátnem bude několik tlačítek, které budou představovat funkce hledání grafu, vymazání plochy a přepínání mezi řešeními (pokud jich je více). Tlačítka pro přepínání mezi řešeními budou nedostupná, dokud uživatel nepoužije vyhledávací tlačítko. Přidávání atributů k uzlům v hledaném grafu je znázorněno na obr. 4.7. Po kliknutí pravým tlačítkem na uzel a vybrání možnosti editace uzlu se objeví tabulka, ve které bude možné vybrat atribut, který uzel umožňuje mít a přiřadit mu určitou hodnotu, podle které pak bude algoritmus daný uzel hledat v prohledávaném grafu. Poslední ná-



Obrázek 4.4: Případy užití modulu

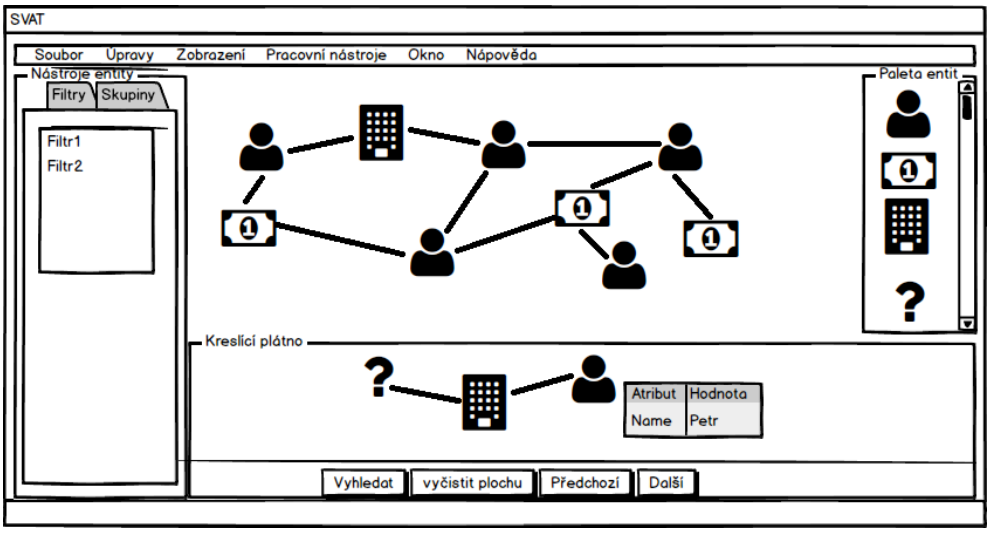


Obrázek 4.5: Obrazovka před otevřením nástroje pro vyhledávání

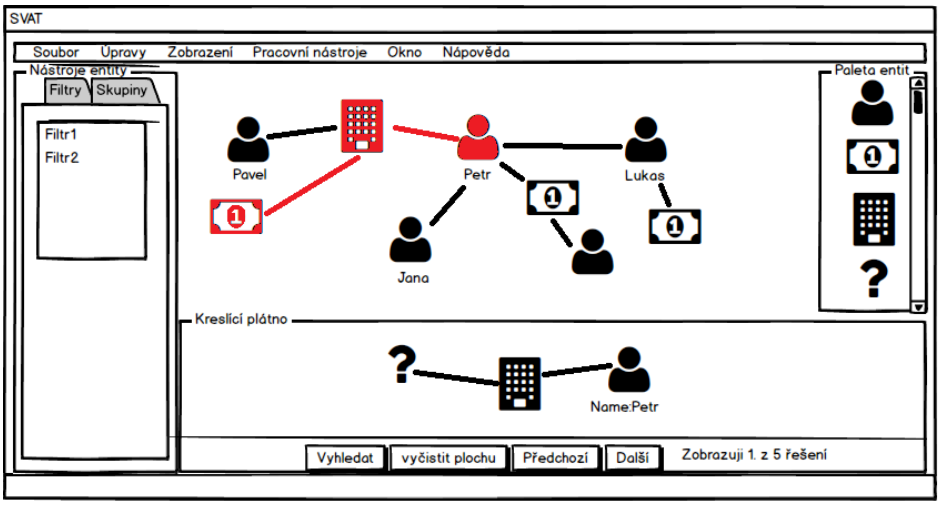


Obrázek 4.6: Základní rozložení oken pro vyhledávání

vrh ukazuje jakým způsobem bude zobrazováno řešení. Pokud jich bude více, tlačítka pro přepínání mezi řešeními se stanou dostupnými a vedle nich bude vypsané, kolik řešení algoritmus našel a u jakého řešení se právě nacházíme. V prohledávaném grafu je dané řešení zároveň vyznačeno4.8.



Obrázek 4.7: Přidávání atributu k uzlům



Obrázek 4.8: Výsledek vyhledávání

Kapitola 5

Implementace

5.1 Implementace modulu pro vyhledávání

Jak bylo řečeno výše, celý projekt SVAT je založen na NetBeans platformě, což je modulární a rozšířitelný aplikační framework. Největší výhodou tohoto frameworku je právě modularita, která zajišťuje snadné přidávání nových komponent k existující aplikaci nebo například zjednodušuje aktualizace již nainstalovaných prvků. Existují základní 3 typy modulů pro NetBeans platformu:

- NetBeans Platform Application - jedná se o projekt, který sdružuje skupinu modulů a library wrapper modulů, které jsou mezi sebou závislé a dovoluje je nasadit jako jeden prvek.
- Library Wrapper Module - už z názvu vyplývá, že se jedná o modul který sdružuje knihovny. Tento modul neobsahuje žádný kód. Slouží k tomu, že pokud některý z modulů využívá externí knihovny, Library Wrapper Module je použit k tomu, že třídy z těchto externích knihoven bude možné použít za běhu.
- Module - projekt, který obsahuje funkcionalitu, business logiku a uživatelské rozhraní aplikace založené na NetBeans platformě

5.1.1 Vyhledávací modul

Z popisu výše vyplývá, že rozšíření nástroje o vyhledávání bude realizováno pomocí samostatného modulu. Tento modul je dále rozdělen na několik balíčků, které jsou dále stručně popsány:

- balíček visualsearch - Tento balíček obsahuje vizuální komponentu, na které se vytváří hledaný graf, dále obsahuje grafické komponenty, které se na plátně mají vykreslit a v poslední řadě obsahuje implementaci samotného vyhledávácího algoritmu

- balíček palette - paleta s jednotlivými prvky grafu, která umožňuje drag and drop svých elementů do hlavního okna pro vyhledávání
- balíček entitygraph - tento balíček představuje datový model pro vyhledávací modul, jsou v něm obsaženy třídy představující vyhledávaný graf a jeho uzly. Jelikož SVAT dovoluje mít pouze atributy přiřazené k uzlům, není potřeba, aby zde byla třída představující hranu. Implementace hran mezi jednotlivými uzly je řešena za pomoci matice sousedností ve třídě EntityGraph

5.1.2 Použité knihovny a implementace scény s grafem

Scéna na kterou uživatel přetahuje jednotlivé prvky z palety je implementována pomocí NetBeans Visual Library, což je knihovna která poskytuje sadu nástrojů pro práci s grafem.

Základními prvky této knihovny jsou tzv. *Widgety*. Jedná se o prvky, které v sobě mají vestavěné různé vlastnosti, jako například akce, které s nimi je možné provádět, layouty a jiné. Tato knihovna je ideální pro řešení problému, kdy je podmínkou, aby uživatel měl vizuální přehled o tom, jaký graf vyhledává.

Základní okno s plátnem je realizováno jako *TopComponent*, což je komponenta která umožňuje interakci se systémem oken v NetBeans. Interakcí je myšleno to, že ji jde stejně jako kterékoli okno v NetBeans přesouvat, aktivovat nebo dokovat na různé pozice podle toho jak to danému uživateli vyhovuje.

Na toto okno je poté umístěna komponenta, která dědí od třídy *GraphScene*, což je v podstatě kořenový uzel co se týče stromové hierarchie widgetů. Tato komponenta kontroluje celou scénu, její view nebo také překreslování a validaci dalších komponent na ní umístěných. Tato komponenta dále umožňuje například také přiblížení nebo oddálení celé scény. Na této komponentě jsou poté umístěny tři tzv. *Layer Widgety*. Každá z těchto tří vrstev má na starosti některé akce:

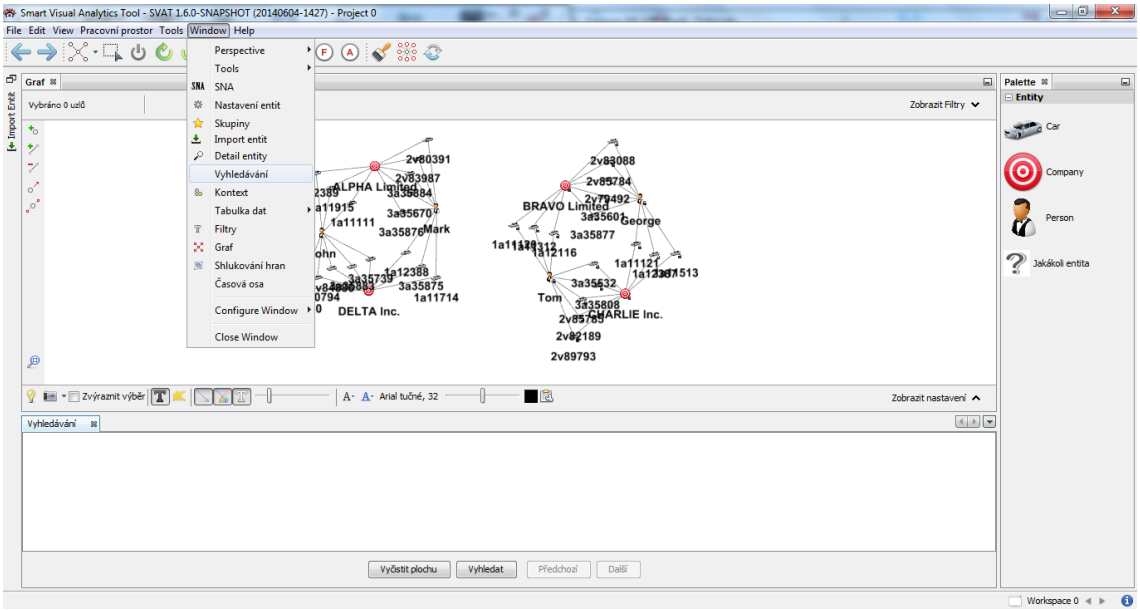
- Jako první je zde *mainLayer* což je vrstva, na kterou jsou umísťovány jednotlivé uzly, které jsou přetáhnuté z palety.
- Druhou je *connectionLayer* což je vrstva na které jsou vykreslované hrany mezi jednotlivými uzly.
- Poslední je *interactionLayer*, která zajišťuje, že daný uzel a jeho hrany budou správně vykreslovány v průběhu drag&drop.

Jednotlivé uzly jsou instancí třídy *EntityWidget*, která dědí od třídy *Icon NodeWidget*. Z názvu vyplývá, že se jedná o widgety, které jsou reprezentované jejich ikonou. Jelikož každá entita na paletě má jinou ikonu, je použití tohoto typu entity výhodné. K dostupným metodám třída *EntityWidget* má oproti svému předkovi přidaných několik metod a parametrů. Nejdůležitějšími

jsou atributy daného uzlu a dále objekt *LabelWidget*, což je další widget, který je s tímto spárovaný a slouží ke zobrazení atributů pod každou entitou.

5.1.3 Uživatelské rozhraní

Od začátku byl kladen velký důraz na uživatelskou přívětivost. Jelikož nástroj SVAT je vizuálně-analytický, jednou z podmínek pro vyhledávacího modulu bylo to, že vyhledávání musí být prováděné graficky. Je to z toho důvodu, že uživatelé, kteří tento nástroj používají, nejsou obvykle příliš zběhlí v práci se složitými programy a upřednostňují intuitivní ovládání.



Obrázek 5.1: Obrazovka po spuštění modulu pro vyhledávání

5.1.3.1 Paleta s entitami

Seznam entit, které v grafu existují, je zobrazen na paletě v pravé části hlavního okna. Podobnost s paletou, na kterou jsou zvyklí uživatelé IDE NetBeans není náhodná, jedná se totiž o tu samou komponentu, která stejně jako ta v IDE umožňuje funkci drag&drop. Jednotlivé entity jsou reprezentovány svojí ikonou a názvem, což je z uživatelského hlediska pravděpodobně nejpříjemnější přistup, jelikož má vizuální kontrolu, že chce opravdu vyhledat danou entitu.

Může se stát, že entita v grafu nemá přidělenou žádnou ikonu. Pro tyto případy takovou entitu na paletě představuje ikona s nápisem N/A (not available) a její název, který naopak musí vždy existovat.

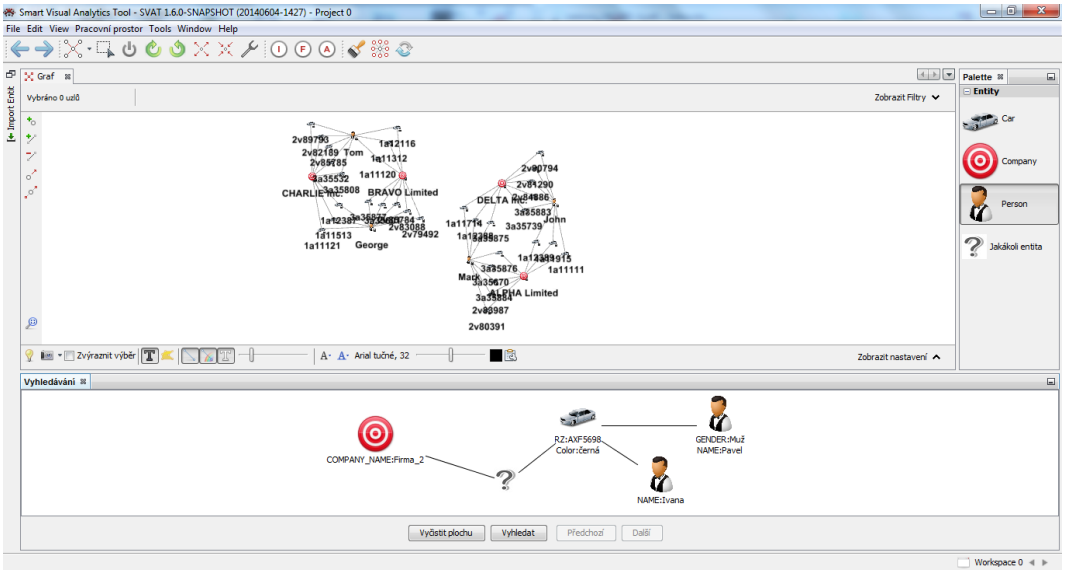
Speciálním případem entity, která v žádném grafu reálně neexistuje, ale je možné ji vždy použít je tzv. *Neznámá entita*. Tato entita představuje jakýkoli uzel v grafu, nemá žádné atributy (jelikož nevíme co hledáme, nemůžeme tomu přiřadit žádný atribut ani název). Jak bylo řečeno v sekci 4, reálné grafy obsahují implicitní vztahy. Z tohoto důvodu je zde tato ikona, kterou pokud uživatel použije, je jí možné nahradit jakýmkoli uzlem v grafu. Příkladem může být např. pokud chce uživatel zjistit zda existuje mezi dvěma firmami nějaké blízké spojení, ať už přes jinou firmu, přes majitele či zaměstnance nebo například přes transakce mezi jejich účty. V tomto případě uživatel použije ikonu neznámé entity jak je ukázáno na obr. 5.2. Na obrázku uživatel hledá cesty délky 3 mezi firmami $Firma_1$ a $Firma_2$, jenž může například představovat vztah mezi majiteli obou firem.

Obrázek 5.2: Ukázka použití neznámé entity

5.1.3.2 Okno pro tvorbu hledaného grafu

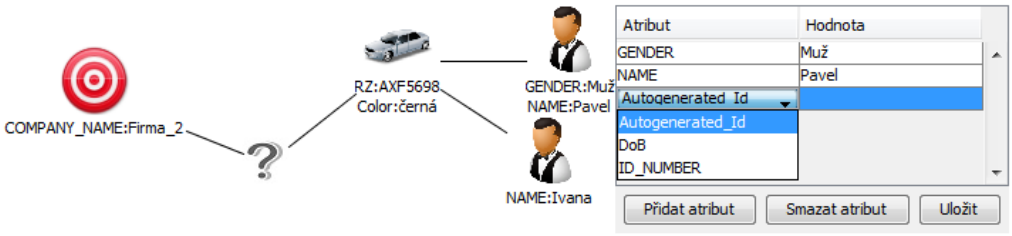
Okno které slouží pro tvorbu hledaného podgrafu je zobrazeno v dolní části na obr. 5.3. Jednotlivé uzly jsou na plátno získaný z palety entit, která se nachází na levé straně pomocí funkce drag&drop. Po přetažení se na plátně objeví pouze ikona představující danou entitu. Po kliknutí pravým tlačítkem na ikonu uzlu se objeví roletkové menu, kde uživatel buď může editovat daný uzel ve smyslu, že chce přiřadit hodnotu nějakému jeho atributu nebo ji případně upravit či smazat. Druhá položka v menu slouží ke smazání daného uzlu. Pokud uživatel smaže uzel, jsou automaticky smazány všechny hrany na něj napojené, jelikož graf nepovoluje, aby hrany jen tak 'visely' ve vzduchu.

Hrany mezi jednotlivými uzly uživatel kreslí při stisku klávesy CTRL a přetažením myší mezi zdrojovým a cílovým uzlem (na pořadí nezáleží, jelikož se jedná o neorientovaný graf). Stejně jako u uzlů, po kliknutí pravým tlačítkem myši na uzel se objeví menu, kde lze danou hranu pouze smazat. Editace není potřebná z toho důvodu, že SVAT v současné podobě nepodporuje přiřazení atributů hranám. Přiřazení atributu jednotlivým uzlům je znázorněno na obrázku 4.7. V podstatě se jedná o dynamicky generovanou tabulku, kde je v prvním sloupci na výběr z jednotlivých atributů a do druhého sloupce uživatel k danému atributu přiřadí hledanou hodnotu. Pokud uživatel už jednou vybral některý atribut a chce přidat další, udělá tak kliknutím na



Obrázek 5.3: Ukázka okna sloužícího pro vyhledávání

tlačítko *Přidat atribut*, čímž se v tabulce objeví další řádek s tím, že na výběr bude pouze z těch atributů, které uživatel dosud nevybral. Analogicky tlačítko *Smazat atribut* slouží ke smazání označeného řádku v tabulce. Po kliknutí na tlačítko *Uložit* se dané natavení atributů uloží. Opět je vše vidět na obrázku 5.4. Ke zvýšení uživatelské přívětivosti je také možné se všemi existujícími



Obrázek 5.4: Přiřazení atributu jednotlivým uzlům

komponentami na plátně pohybovat, což se provádí jednoduše klinutím na uzel a přetažením na jiné místo. Zároveň je zde implementována funkce 'pravítka', která dokáže jednotlivé uzly zarovnat jak vedle sebe tak nad sebe viz. obr. 5.5. Samotné vyhledávání je realizováno kliknutím na tlačítko *Vyhledat* v dolní části okna pro vyhledávání. Pokud existuje více řešení, vedle tlačítka *Vyhledat* se zobrazí jejich počet a zároveň se zpřístupní tlačítka pro přepínání

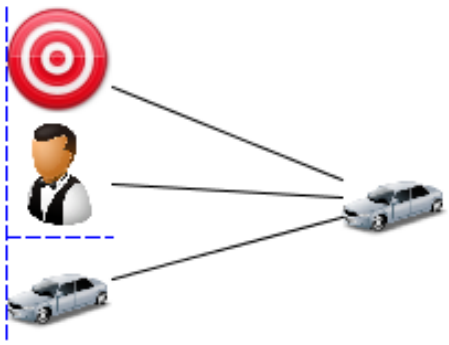
mezi nimi. Jednotlivé uzly řešení jsou poté v programu zvýrazněny a zároveň, jelikož se obvykle jedná o velké grafy, které se nevejdou celé na plátno s grafem, je na tyto uzly zaostřeno. Příklad je uveden na obrázku 5.6. Konkrétně v tomto případě existuje 32 možných řešení, jelikož žádná z entit nemá přiřazený žádný atribut a tedy algoritmus nalezne všechna řešení bez závislosti na nich.

5.1.4 Vyhledávací algoritmus a jeho optimalizace

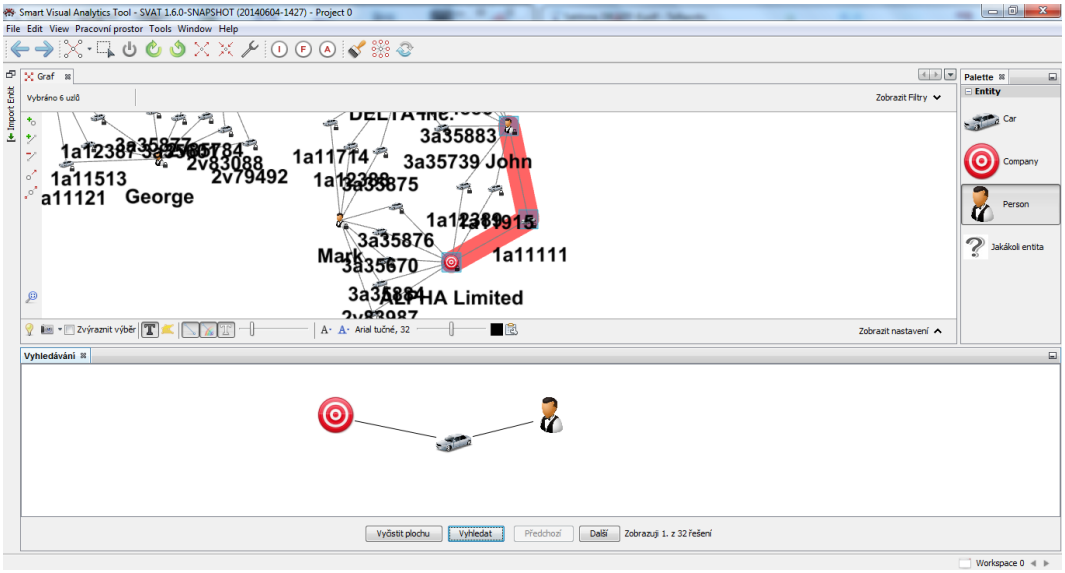
Z důvodů, jenž byly zmíněny v části 3.1.5 byl vybrán algoritmus Vflib jako vhodný pro implementaci vyhledávání v grafu. Oproti originálnímu algoritmu je zde nicméně jeho implementace ořezána o podmínku, že hledaný podgraf musí být indukovaný. Stačí pouze pokud se jedná o podgraf částečný. Tato podmínka je odstraněna z toho důvodu, že pro uživatele používající tento nástroj není podstatné, zda graf obsahuje jimi přesně hledaný podgraf v tom smyslu, že v něm musí být jednotlivé uzly spojené hranami stejně, jako jsou v prohledávaném grafu.

Dalším rozdílem oproti originální implementaci je to, že algoritmus musí najít všechna řešení, z čehož vyplývá, že si v paměti ukládá všechny částečné izomorfismy a postupně zkouší všechny možné větve ve vyhledávacím stromu, dokud buď nenalezne celkové řešení nebo dokud není možné v daném řešení dále pokračovat z důvodu nesplnění jedné z podmínek vflib algoritmu.

Algoritmus sám o sobě je dobře optimalizovaný, přesto bylo přidáno několik heuristických vylepšení ke zvýšení jeho rychlosti. Aby bylo vyhledávání co nejrychlejší, před spuštěním algoritmu proběhne předzpracovací fáze, ve které jsou určeny typy uzlů hledaného podgrafu. Typem uzlu se rozumí, že se jedná například o Auto nebo Osobu. Poté je pro každý z těchto typů určena jeho četnost v hlavním grafu a algoritmus začíná prohledávat toho typu uzlu, jehož četnost je nejmenší. Tento způsob značně omezí prohledávaný prostor hned na



Obrázek 5.5: Ukázka funkce pravítka



Obrázek 5.6: Ukázka vyhledání daného podgrafu

začátku, protože například v grafu, který představuje transakce mezi účty osob je typicky několikanásobně více uzlů typu transakce mezi uzly reprezentujícími bankovní účty.

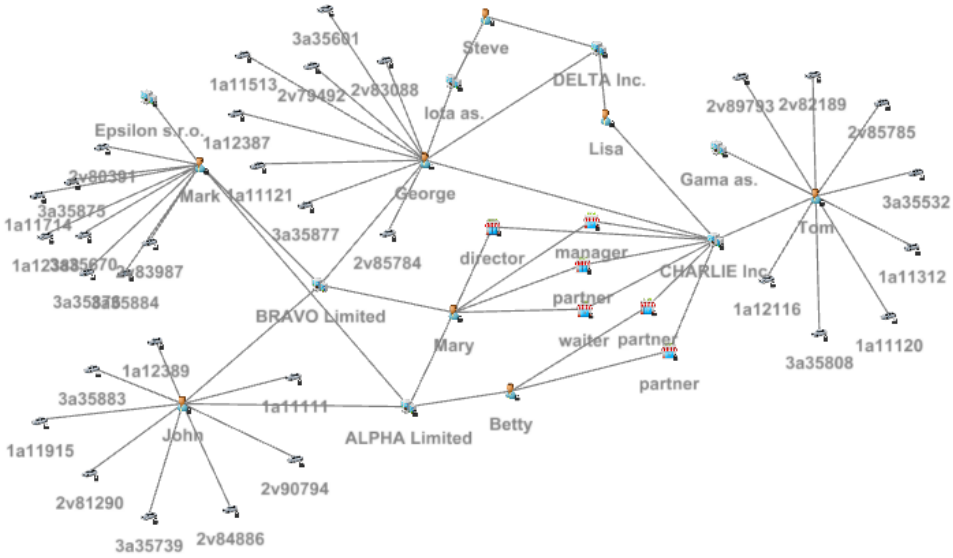
Je zřejmé, že pokud uživatel hledá transakce na jakýkoli cizí účet, jdoucí z určitého účtu, je výhodnější, aby algoritmus začal vyhledávat nejdříve konkrétní účet a až poté jeho případné transakce.

Jednotlivé uzly mohou (ale nemusí) mít další atributy, na základě kterých chce uživatel provádět vyhledávání. Tyto atributy jsou dané typem uzlu a nelze je nijak měnit. Příkladem atributu může být u uzlu typu Auto například barva nebo poznávací značka. Tato vlastnost uzlů je rovněž zohledněna v předzpracovávací fázi tím způsobem, že algoritmus vybere jako vstupní bod uzel, který má buď nejmenší četnost (za předpokladu, že žádný z uzlů v hledaném grafu není definován dalšími atributy) a nebo ten uzel, ke kterému je přiřazeno nejvíce atributů). Tato podmínka je zde díky předpokladu, že čím více je toho o daném uzlu známo, tím je specifičtější a tudíž by se měl v grafu vyskytovat co nejméněkrát, čímž se už v prvním kroku hledání vyhledávací prostor ořeže na možné minimum. Pokud je takových uzlů se stejným počtem specifikovaných atributů více, opět je vybrán ten, jehož typ má nejmenší četnost.

5.1.5 Testování správnosti algoritmu

Testování správnosti implementace bylo složeno ze dvou částí a to otestování, zda algoritmus vrací správné výsledky při hledání různých typů vzorů a druhou částí bylo testování rychlosti algoritmu. Tato kapitola se zabývá ověřením

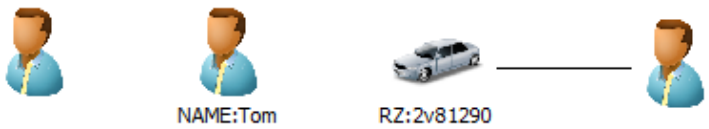
správnosti vyhledávacího algoritmu. Graf, na kterém byla testována správnost algoritmu je zobrazen na obr. 5.7. Jak je vidět z obrázku, graf, na kterém byla korektnost algoritmu testována, nemá příliš mnoho uzlů ani hran. Je to z toho důvodu, že nebylo možné algoritmus ověřit jiným způsobem než vizuálně, protože řešení nebylo známé. U větších grafů by byl poté problém například najít ručně všechna správná řešení, pokud by jich existovalo více.



Obrázek 5.7: Graf, na kterém byla kontrolována správnost vyhledávacího algoritmu

5.1.5.1 Jednoduché vzory

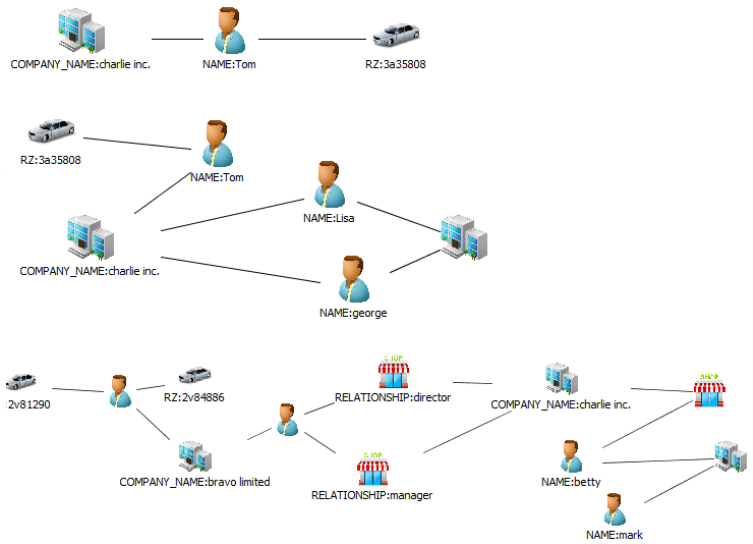
Jako první byl algoritmus testován na jednoduché vzory. V tomto kontextu je jednoduchým vzorem myšlen samostatný uzel, samostatný uzel s parametrem a jednoduchá vazba dvou uzlů, která má jednoznačné řešení viz. obr. 5.8. Na těchto vzorech algoritmus fungoval bez problémů a vždy našel správné řešení.



Obrázek 5.8: Skupina jednoduchých vzorů pro testování správnosti algoritmu

5.1.5.2 Složité vzory s jednoznačným řešením

Druhou skupinou vzorů, kterými byl algoritmus testován, byly takové kombinace, které se skládaly z 3 a více uzlů, ale zároveň měly jednoznačné řešení. Nejsložitější vzor, který byl hledán obsahoval 12 uzlů. Příklad hledaných vzorů je na obr. 5.9. Při testování této skupiny byla objevena chyba v algoritmu, která se projevila při vyhledávání vzorů, které obsahovaly cykly. Tato chyba byla způsobena tím, že algoritmus už jednou projitý uzel při dalších iteracích ignoroval, a proto v případě cyklu nenalezl žádné řešení.



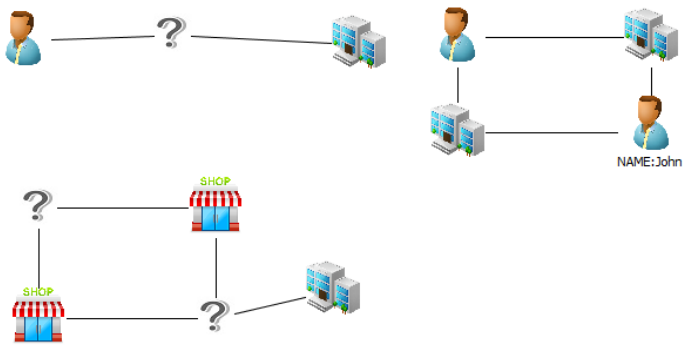
Obrázek 5.9: Složitější vzory pro testování správnosti algoritmu

5.1.5.3 Vzory s více řešeními

Jednou s podmínek zadavatele byla schopnost algoritmu nalézt všechna řešení pro daný vzor a proto poslední skupinou vzorů, které byly testovány, byly takové, pro které existuje více možných řešení. V této chvíli byla zároveň testována také implementovaná funkčnost náhodné entity, která reprezentuje jakýkoli uzel v grafu. Příklad testovaných vzorů je možné vidět na obr. 5.10.

5.1.6 Testování rychlosti vyhledávání

K prokázání kvality implementovaného algoritmu bylo nutné ověřit, že je schopný vyhledat řešení v co nejkratší době. V první fázi byl vygenerován graf o menší velikosti (cca 500 uzlů). Na tomto grafu byla testována správná funkčnost vyhledávání. Na takto malém grafu si algoritmus vedl ve všech ohledech dobře, protože řešení nalezl vždy okamžitě. Pro druhou fázi testování byl



Obrázek 5.10: Příklad vzorů s více řešeními

vygenerován graf, který obsahoval přibližně 65 000 uzlů a 200 000 hran. Testování bylo prováděné na následující konfiguraci:

- Procesor: Intel(R) Core(TM) i5-2410M 2.30 GHz
- Operační paměť RAM: 8,00 GB
- Operační systém: Windows 7 Home Premium

V této fázi je nutné podotknout, že samotný import entit do nástroje trval přibližně 5 minut. Vyhledávání bylo rozděleno na několik případů:

- Vyhledání samotné entity
- Vyhledání podgrafu o velikosti 2 s atributy
- Vyhledání podgrafu o velikosti 3 s atributy
- Vyhledání podgrafu o velikosti 4 s atributy
- Vyhledání podgrafu o velikosti 5 s atributy
- Vyhledání obecného podgrafu o velikosti 2
- Vyhledání obecného podgrafu o velikosti 5

To, že je v některých případech podgraf obecný, znamená, že pro vyhledávání bylo prováděno pomocí Neznámych entit. Výsledky měření jsou znázorněny v následující tabulce: Z naměřených výsledků vyplývá, že implementovaný algoritmus bez problémů obstojí při vyhledávání ve velkých grafech. Je předpokladem, že uživatel který program využívá, hledá nějaké konkrétní řešení (nehledá obecný podgraf). V tomto případě bylo řešení nalezené prakticky

	Počet výsledků	Doba vyhledávání
Unikátní entita	1	0.60
Velikost 2 s atributy	8	0.68 s
Velikost 3 s atributy	1	0.70 s
Velikost 4 s atributy	2	0.65 s
Velikost 5 s atributy	3	0.92 s
Velikost 2 obecně	568200	3 min 28 s
Velikost 5 obecně	-	nedokončeno

okamžitě. Z hlediska uživatelů se dá taktéž předpokládat, že budou vyhledávat maximálně grafy o velikosti 5 nebo 6, a proto nebyly vyšší hodnoty testovány.

Doba vyhledávání podgrafu o velikosti 5 s atributy (řádek 3) je oproti vyhledávání podgrafu o velikost 2 s atributy (řádek 2) je relativně vyšší (avšak z pohledu uživatele tedy absolutně je tento rozdíl nulový) i přesto, že bylo nalezeno méně řešení. Je to dáno tím, že algoritmus nezvolil správně výchozí bod pro vyhledávání - zvolil entitu s nejvyšším počtem atributů a s nejnižším počtem výskytů, nicméně ostatní uzly hledaného podgrafu mohly počáteční strom prořezat více.

Tomuto jevu by se dalo předejít tím, že algoritmus na začátku vyzkouší počáteční mapování na každý uzel z grafu a poté vybere jako počáteční bod ten, na který je možné namapovat co nejméně uzlů v prohledávaném grafu. Za tohoto předpokladu je ovšem otázkou, zda cena vyzkoušení mapování každého z uzlů vykompenzuje cenu, kterou by algoritmus případně ušetřil při prohledávání stavového prostoru.

Algoritmus začal mít problémy v tom případě, že existuje velké množství možných výsledků. Nicméně pro problémy, na které je toto vyhledávání určeno se předpokládá, že počet výsledku bude velmi malý a v tomto ohledu je spolehlivost algoritmu velmi dobrá.

Poslední fáze testování byla už na reálných datech, které byly dodané zadavatelem. Tyto grafy obsahovaly od několika desítek po přibližné 5 000 uzlů. Stejně jako v sekci 5.1.5. byly vyhledávány takové vzory, u kterých se dá očekávat, že budou hledány uživateli programu. Na těchto datech algoritmus řešení nalezl vždy okamžitě.

5.1.7 Testování zákazníkem

Poslední test, kterým implementovaný modul prošel, bylo testování zadavatelem práce, tedy firmou Profinit. Jako první byla testována splnitelnost požadavků definovaných zadavatelem. To znamená, že vyhledávání musí být grafické (uživatel si nakreslí, co chce hledat), dále jednoduchost celého řešení a v neposlední řadě korektnost a rychlost implementovaného algoritmu.

Co se týče vizuální stránky, ta byla po celou dobu implementace průběžně se zadavatelem konzultována a byla tedy přijmuta bez výhrad. Otestování funkčnosti prováděl interní testovací tým firmy Profinit a také během něj nebyly objeveny žádné chyby. Zároveň i rychlost algoritmu je dle zadavatele pro účely, ke kterým bude vyhledávání sloužit více než dostačující.

5.1.8 Výsledky testování

Z výše provedených testů byl učiněn závěr, že modul je možné nasadit do ostrého provozu. Jedinou problémovou částí je rychlost algoritmu při vyhledávání velmi obecných vzorů nad obrovským vstupním grafem. Vzhledem k tomu, že celý program slouží k detekci podvodného jednání, zadavatel nepředpokládá, že bude takové vyhledávání potřeba.

5.2 Zdrojové kódy

Vzhledem k tomu, že nástroj SVAT je komerční software, není možné zveřejnit zdrojové kódy ani testovací verzi. Na přiloženém CD jsou pouze zdrojové kódy implementovaného modulu, které ale nejsou samostatně spustitelné. Implementační část práce je z toho důvodu psána takovým způsobem, aby si čtenář udělal představu o tom, jak implementovaný modul vypadá.

Kapitola 6

Možnosti rozšíření

Možností na rozšíření této práce se nabízí celá řada. Prvním z nich by mohlo být vylepšení algoritmu v tom smyslu, že by dokázal obstojněji pracovat s obecnými datovými strukturami, i když to není prioritou při vyhledávání konkrétních věcí, jak bylo řečeno výše. Zajímavou možností rozšíření algoritmů pro vyhledávání by mohla být implementace některého z algoritmů popsaných v sekci 3.2. Díky tomuto způsobu vyhledávání by například mohla být odstraněna ikona neznámé entity, kde je pro uživatele problém určit, pokud například hledá vazbu mezi dvěma osobami, kolik takových entit mezi nimi musí být. Místo ikony neznámá entita by bylo tedy možné použít přibližné vyhledávání, které by uživateli jasně ukázalo jak hodně se od sebe liší jím hledaný podgraf a ten, který ve skutečnosti v prohledávaném grafu existuje. Vhodným kandidátem by mohl být konkrétně algoritmus 3.2.1, jenž umožňuje měřit chybu v závislosti na operacích přidání a ubírání hran nebo uzlů.

Dalším možným rozšířením, které nebylo v rámci této práce nebylo realizováno je možnost vyhledávání přímo nad databázemi, kde by uživatel nemusel nejdříve danou databázi překonvertovat do formátu vhodného ke grafovému znázornění. Tento problém je velmi komplexní z důvodu množství různých databází, na které je možné nástroj napojit a dále z toho důvodu, že data v těchto databázích často nejsou v jednotné formě, a proto by se takové vyhledávání mohlo ukázat jako neefektivní. V současné době vzniká nová verze, která bude napojení na databáze umožňovat a vyhledávání přímo nad databázemi je řešeno jako součást dalšího grantu.

Závěr

Na základě zadání byla práce zaměřena na prozkoumání metod pro vyhledávání ve velkých grafech. Následně byla vybrána jedna metoda, která se jeví jako nejlepší a byla použita pro vyhledávání v grafu ve vizualizačním nástroji SVAT, který primárně slouží pro detekci podvodného jednání. Důležitou podmínkou implementace tohoto vyhledávacího modulu byla uživatelská přívětivost v tom smyslu, že vyhledávání bude prováděno graficky a bude uzpůsobeno mimo jiné pro uživatele, kteří nemají tolik technických zkušeností pro práci se složitými programy. V první části byl představen problém vyhledávání v grafech obecně na několika demonstračních aplikacích, z čehož poté vyplynuly některé vlastnosti reálných grafů, které bylo při implementaci nutné zohlednit. V navazující části byl poté problém vyhledávání představen v rovině teorie grafů, jejíž pochopení bylo nutné ke správné implementaci. Byl představen seznam několika algoritmů sloužících pro vyhledávání v grafech, které byly rozděleny do několika skupin na základě jejich vlastností. Z těchto skupin algoritmů se jako nejvhodnější jevily ty, které slouží k přesnému vyhledávání. Na základě studii o těchto algoritmech byl vybrán jako nejlepší algoritmus vflib, jehož implementace se jevila jako nejvhodnější z požadovaného hlediska rychlosti vyhledávání. Dalším krokem poté byla implementace modulu pro vyhledávání do vizualizačního nástroje SVAT, který měl splňovat podmínky dané zadavatelem této práce. V této části bylo nutné algoritmus mírně upravit tak, aby vyhovoval potřebám uživatelů a v další řadě provést optimalizace vedoucí k jeho ještě většímu zrychlení. V poslední fázi byly otestovány teoretické předpoklady implementovaného algoritmu z pohledu jeho korektnosti a rychlosti. Na základě výsledků těchto testů spolu s výsledky testů zadavatele se stal modul pro vyhledávání součástí nástroje SVAT a v současné době je třetí rok v provozu.

Literatura

[1] *World Wide Web*. `https://cs.wikipedia.org/wiki/World_Wide_Web`.

[2] Johnson, T.: Community detection in quantum networks. 2014.

[3] Krebs, V.: Mapping networks of terrorist cells. 2001.

[4] D. Conte, F. C. S., P; Vento, M.: Thirty years of graph matching in pattern recognition. 2004: s. 266–283.

[5] Zampelli, S.: A Constraint Programming Approach to Subgraph Isomorphism. 2008: s. 35–45.

[6] Wu, Y.: Extending Graph Homomorphism and Simulation for Real Life Graph Matching. 2010: s. 4–8.

[7] Bharat, K.; Broder, A.: Mirror, Mirror on the Web: A Study of Host Pairs with Replicated Content. 1997.

[8] Terveen, L.; McDonald, D. W.: Social matching: A framework and research agenda. 2005.

[9] Ullmann, J.: An Algorithm for Subgraph Isomorphism. *J. Assoc. for Computing Machinery*, ročník 23, 1976: s. 31–42.

[10] Luigi Cordella, C. S. F., Pasquale Foggia; Vento., M.: Graph matching: A fast algorithm and its evaluation. *ICPR*, ročník 2, 1998: str. 1582.

[11] L. P. Cordella, C. S., P. Foggia; Vento., M.: Performance evaluation of the vf graph matching algorithm. *ICIAP*, ročník 2, 1999: str. 1172.

[12] Messmer, B.: Efficient Graph Matching Algorithm for Preprocessed Model Graphs. 1995.

[13] B. Messmer, H. B.: Efficient subgraph isomorphism detection : a decomposition approach. *Pattern recognition*, 1998: s. 1979–1993.

[14] A. C. M. Dumay, R. J. v. d. G.: Consistent inexact graph matching applied to labelling coronary segments in arteiograms. *TProc. Int. Conf. Patter Recc.*, 1992: s. 439–442.

[15] Peter E. Hart, N. J. N.; Raphael, B.: A Formal Basis for the Heuristic Determinination of Minimum Cost Paths. *IEEE Trans. Syst. Man Cybern.*, ročník 4, 1968: s. 100–107.

[16] Zavlanos, M. M.; Pappas, G. J.: A Dynamical Systems Approach to Weighted Graph Matching.

[17] Jiang, J.: An introduction to spectral graph theory.

[18] P. Foggia, C. S.; Vento, M.: A performance comparison of five algorithms for graph isomorphism. *Proc. 3rd IAPR-TC15 Workshop Graph-Based Representations in Patt. Recc.*, 2001: s. 188–199.

[19] Baskararaja, G. R.; Manickavasagam, M. S.: Subgraph Matching Using Graph Neural Network. *Journal of Intelligent Learning Systems and Applications*, ročník 4, 2012: s. 274–278.

[20] Auwatanamongkol, S.: Inexact graph matching using a genetic algorithm for image recognition. *Pattern Recognition Letters*, ročník 28, 2007: s. 1428–1427.

[21] Linkurious. `http://linkurio.us/product/docs/`, verze z 8.4.2016.

[22] Maltego. `https://www.paterva.com/web7/`, verze z 8.4.2016.

[23] Tulip. `http://tulip.labri.fr/TulipDrupal/`, verze z 8.4.2016.

[24] Gephi. `https://gephi.org/`, verze z 8.4.2016.

Příloha **A**

Seznam použitých zkratek

GUI Graphical User Interface

WGM Weighted Graph Matching

SVAT Smart Visual Analytics Tool

BFS Breadth First Search

NASA National Aeronautics and Space Administration

CSV Comma Separated Values

GEXF Graph Exchange XML Format

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF