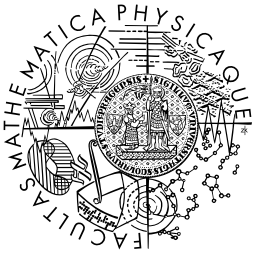


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Matyáš Brabec

Podobnostně-grafový prohlížeč databáze strukturovaných entit

Katedra softwarového inženýrství

Vedoucí bakalářské práce: prof. RNDr. Tomáš Skopal, Ph.D
Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2021

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Rád bych tímto poděkoval svému vedoucímu práce prof. Tomáši Skopalovi za jeho pomoc při psaní textu a návrhu aplikace, bez jeho vhledu by se tato práce neobešla. Nemenší díky patří také Petru Paščenkovi za jeho vedení práce ve firmě Profinit a za předané zkušenosti, bez kterých by práce nemohla vzniknout. Dále bych rád poděkoval Profinitu za poskytnutá data o jeho zaměstnancích, konkrétně Bohumíru Zoubkovi za předzpracování poskytnutých dat. V neposlední řadě bych chtěl také poděkovat své rodině a blízkým za podporu při psaní.

Název práce: Podobnostně-grafový prohlížeč databáze strukturovaných entit

Autor: Matyáš Brabec

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: prof. RNDr. Tomáš Skopal, Ph.D, Katedra softwarového inženýrství

Abstrakt: Práce se věnuje alternativnímu způsobu vyhledávání na bázi podobnosti a následnému zobrazování výsledků. Při klasickém dotazování do SQL databáze, Excel tabulky nebo i online obchodu se uživatel musí seznámit s datovým modelem a následně popsat, jak má výsledek vypadat v jazyce atributů entit (pomocí rovností, nerovností, intervalů, regulárních výrazů apod.). Na rozdíl od tohoto přístupu má podobnostní hledání velkou výhodu, uživatel nemusí vědět o entitách skoro nic. Jediné, z čeho se dotaz skládá, je jedna nebo více entit, na jejichž základě aplikace zobrazí nejpodobnější množinu jako interaktivní graf. Cílem je koncept prozkoumat na doméně zaměstnanců softwarové firmy. Této společnosti potom bude výsledná aplikace sloužit na doporučování vhodných osob na projekty.

Klíčová slova: podobnostní vyhledávání, grafový prohlížeč, podobnostní metriky

Title: Similarity-graph based browser of database of structured entities

Author: Matyáš Brabec

Department: Department of Software Engineering

Supervisor: prof. RNDr. Tomáš Skopal, Ph.D, Department of Software Engineering

Abstract: Work is dedicated to an alternative way of searching entities based on their similarity and comprehensively viewing them. A disadvantage of a classical SQL search is that a user has to be familiar with a real-world entity as well as a data model entity. Only then he can formulate a query based on attributes of the entity (e.g. using equality, intervals, regular expressions). Similarity search does not force the user to know the structure of the data. The only thing the user puts into the query is one or more entities and he obtains a result consisting of entities similar to the given entity (or entities) as an interactive graph. The main goal is to explore the concept of similarity browser on employees of the software company Profinit. The final product will serve Profinit as a searching tool during filling up projects with people.

Keywords: similarity search, graph browser, similarity metrics

Obsah

1. Úvod

V současné době shromažďujeme o světě okolo nás mnoho informací, ty následně skladujeme, učíme pomocí nich algoritmy, popřípadě je zpracováváme mnohými dalšími způsoby. Ovšem co děláme v podstatě se všemi daty je, že v nich vyhledáváme a právě tomu je věnována následující práce.

Ať už je potřeba vyhledat v SQL databázi, Excel tabulce nebo třeba v online obchodu, tak je nutné popsat, jak mají atributy (tj. vlastnosti) entit vypadat nebo co mají splňovat (pomocí rovností, nerovností, regulárních výrazů apod.). Tento typ vyhledávání ovšem vyžaduje seznámení se jak s matematickým/datovým modelem entity, tak i s entitou reálného světa, jenž model modeluje. Pojďme si koncept představit na příkladu. Mějme zákazníka, jehož cílem je koupit novou televizi. První co musí udělat je zjistit, jak televize vypadá a čím se od sebe jednotlivé kusy liší (tj. seznámit se entitou v reálném světě). Například se dozví, že televize jsou různých velikostí, že mají různé vstupy a výstupy, že jsou vybaveny chytrými funkcemi a tak dále. Poté, co dokončí svůj průzkum a rozhodne se, jak má jeho budoucí televize vypadat, se může začít rozhlížet v konkrétních obchodech. Zjišťuje, jaké informace obchodník poskytl a podle čeho umožnil filtrovat (tj. seznamuje se datovým modelem).

V této práci bude zkoumán a následně implementován jiný způsob, kterým je podobnostní vyhledávání. To má oproti klasické metodě jednou velkou výhodu, uživatel (tj. vyhledávající) se nebude muset starat o to, jak vypadá reálná entita natož potom matematický/datový model (i když znát reálnou entitu může jistě pomoci). Princip je snadný, místo vyhledání na základě popisu vlastností lze vyhledávat na základě podobnosti s jinou entitou (nebo případně více entitami). Zůstaňme u příkladu s televizemi, opět je uživatel postaven před problém nákupu nové televize, ale tentokrát si nechce komplikovat život zdlouhavým průzkumem. Jednoduše by chtěl televizi podobnou té, kterou vlastní jeho známý.

Je jisté, že zrovna v doméně televizí by bylo takové dotazování spíše komplementární ke klasickému filtrování, ale existují i domény, kde by tyto podobnostní dotazy mohly dominovat. Kupříkladu pokud uvážíme nakupování vína, u kterého existuje nepřeberné množství atributů, které ho popisují, tak zákazník amatér nemá šanci se v problematice vyznat. Přitom jeho cíl může být poměrně snadno definovatelný, rád by si koupil láhev, která chutná podobně jako ta, kterou již měl a ví o ní, že mu chutnala.

Jak by se dalo čekat, oproštění uživatele od nutnosti znalosti domény není zadarmo. Nelze již databázi daty pouze naplnit, vytvořit indexy nad atributy a prohlásit práci za hotovou. Je nutné, po konzultaci s odborníkem v dané doméně (např. vinařem), sestavit vhodnou metriku/podobnost, která bude dostatečně dobře popisovat vztahy mezi entitami (např. víny nebo televizemi).

Další neméně důležitou částí práce je prohlížení výsledku hledání. Jednou z možností je pouze vypsat výsledné entity, podobně jako to praktikují například online obchody. Tento způsob je perfektní právě pro klasická „SQL“ hledání, nicméně se ukazuje, že pro výsledek podobnostního hledání nemusí být tato metoda příliš vhodná. Důvod je poměrně prostý, v tabulce se zcela ztrácí informace o podobnostech mezi vyhledanými entitami. Další, zde zvolenou, možností je zobrazovat výsledek jako graf, což umožnuje podobnosti vyjádřit jako hrany. S tím

přicházejí další otázky jako, jak velký má graf být, jaké hrany mají být obsaženy, jaký algoritmus použít pro samotné vykreslení grafu na obrazovku monitoru atd.

Dalším cílem práce je vytvořit *proof of concept* aplikaci pro softwarovou firmu Profinit, která bude implementovat úlohy řešené v práci (tj. podobnostní a vizualizační model). Aplikace bude sloužit k usnadnění vyhledávání mezi jejími zaměstnanci, konkrétně bude možno aplikaci použít například v případě, že vznikne nový projekt, pro který bude potřeba najít vhodné zaměstnance. Personalistovi potom bude stačit znát pouze několik zaměstnanců, o kterých ví, že by byli ideální na otevřenou pozici a pomocí aplikace najde další možné kandidáty. Důvodem proč vyhledávat další programátory, když už ví o ideální osobě, může být například neschopnost daného programátora na projekt nastoupit. Kromě toho aplikace rozšíří výběr personalisty, který nebude nucen pozici obsadit prvním adeptem, na kterého si vzpomněl.

Aplikace musí být schopna zpracovávat dva typy dotazů, v prvním případě uživatel zadá jednoho zaměstnance a program zobrazí vhodným způsobem jemu podobné zaměstnance.

Druhým požadavkem je možnost zadat zaměstnanců více. Zde je očekávaným výsledkem vizualizace společného okolí všech zadaných zaměstnanců.

V obou případech je nutno umožnit uživateli změnit parametry podobnosti, například personalista může požadovat zaměstnance podobných znalostí a již mu nebude prioritně záležet na délce pracovního poměru v Profinitu.

Jak již bylo uvedeno výše, pro podobností hledání potřebujeme vhodná data, pomocí kterých bude možno modelovat podobnost mezi jednotlivými zaměstnanci. Tato data jsou z několika interních zdrojů firmy a jejich relevance (pro modelovanou podobnost) byla diskutována s budoucími uživateli.

2. Modelování deskriptorů

Jak bylo zmíněno v úvodu, data zaměstnanců byla získána z mnoha interních zdrojů, zde budou představeny jednotlivé atributy zaměstnanců, ze kterých bude následně vytvořena podobnostní metrika.

Pravděpodobně nejzajímavějším souborem dat, který byl k dispozici, je matice znalostí. Jedná se o tabulku, jejíž řádky reprezentují zaměstnance a sloupce jednotlivé znalosti. Tyto informace si každý ze zaměstnanců vyplňuje při nástupu do firmy samostatně a posléze má povinnost ji udržovat aktuální. Bohužel tato povinnost není ze strany firmy kontrolována, a tedy znalosti mohou být neaktuální i několik měsíců. Tvůrci matice navrhli celkem 21 kategorií znalostí, jako jsou například Data management, QA & Testing, Operační systémy, Infrastruktura, Middleware, DBMS, Big data, Data science, Doménové znalosti, Teorie, Programovací jazyky a nástroje. Každá z 354 možných znalostí potom spadá právě do jedné z kategorií. Pro každého zaměstnance nabývá daná znalost celočíselných hodnot od 0 (žádná znalost) do 3 (expertní znalost).

Příkady kategorií a některých jejích znalostí v tabulce:

Kategorie	**Příklady znalostí**
Programovací jazyky, nástroje	Java SE, C, C#, R, JSON, Haskel ...
Aplikační frameworky a nástroje	Spring, RPC, Net Beans, Vue.js ...
DBSM	MS SQL, Teradata, PostgreSQL ...
Data management	Data Analytics, Data Anonymization ...
Big data	Hadoop, Apache Kafka, Apache Hive ...
Data science	Časové řady, Graph Mining, Statistika ..
Teorie	Algoritmy a složitost, Regulární výrazy...
QA & Testing	Junit, Selenium, Watir, TestNG ...
...	

Tabulka 2.1: Příklady kategorií znalostí

Dále je u zaměstnance známa jeho pozice, která je rovna jedné z pěti hodnot: Junior Consultant, Consultant, Experienced Consultant, Principal Consultant.

Poslední data potřebující vysvětlení jsou supersegment a buddy. Supersegment je jiný název pro divize firmy, tj. jedná se o interní dělení zaměstnanců. Buddy je osoba na kterou se zaměstnanec obrací s dotazy týkajícími se firmy nebo projektu. Atribut buddy poskytuje další náhled, kde zaměstnanec ve firmě pracuje.

V neposlední řadě jsou k dispozici roky nástupů do firmy, velikosti úvazků (částečný/plný), historie projektů v rámci firmy a univerzity, které zaměstnanci studují nebo studovali.

Shrnutí výše uvedeného v následující tabulce:

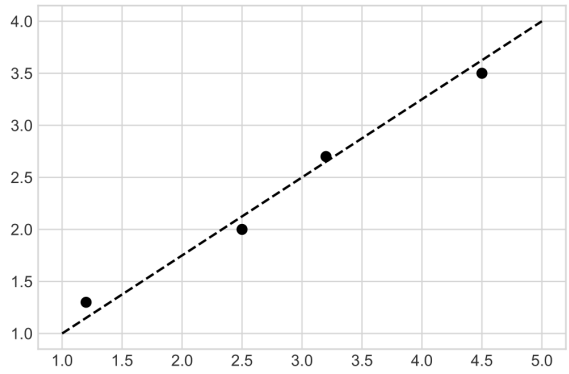
Data o zaměstnanci	Typ dat
Znalosti	354 atributů nabývajících hodnot 0-3
Historie projektů	String obsahující názvy projektů
Současný projekt	String s názvem projektu
Škola	Název vystudované školy
Datum nástupu	String s rokem nástupu
Pozice	Výčtový typ mající pět různých hodnot
Supersegment	ID supersegmentu (jiný název pro divizi firmy)
Buddy	ID buddyho
Úvazek	Dvě možné hodnoty - plný/částečný

Tabulka 2.2: Souhrn dostupných dat o zaměstnanci

2.1 Redukce dimenzionality matice znalostí

V předchozích odstavcích bylo nastíněno, jak matice znalostí vypadá a jakým způsobem vzniká. Zde budou vysvětleny důvody redukce dimenzionality matice znalostí, bude představen použitý algoritmus a bude vysvětleno, z jakého důvodu není vhodné dělat redukci na základě již existujících kategorií.

Nejprve však definujme, co zde redukce dimenzionality znamená. Mějme matici $\mathbf{V}$ tvaru $n \times R$, tj. matice má n řádkových vektorů každý o dimenzi R. Potom dále předpokládejme, že vnitřní dimezionalita matice je r, ta není předem známa. Pod pojmem vnitřní dimezionalita je myšlena dimenzionalita podprostoru ve kterém leží všechny řádkové vektory matice nebo jsou tomuto prostoru blízko.



Obrázek 2.1: Ukázka prostoru o nižší vnitřní dimenzi

Na obrázku 2.1 lze vidět několik vektorů dimenze 2, nicméně jejich vnitřní dimenze je 1, neboť všechny leží v prostoru (nebo blízko prostoru) dimenze 1. Na obrázku je tento prostor znázorněn tečkovanou přímkou.

Matici $\mathbf{W}$ $(n \times r)$ označíme za výsledek redukce dimenzionality matice $\mathbf{V}$, pokud jsou v co nevětší míře zachovány relativní pozice řádkových vektorů vůči sobě.

Ještě je nutné poznamenat, že neurčitá použitá spojení jako „být blízko“ nebo „v co nevětší míře zachovány“, nedávají přesný návod úmyslně. Záleží čistě na

daném datasetu a následnému využití redukované matice, co označíme za dobrou či špatnou redukci.

Redukce dimenzionality realizuje proces *feature selection*, který je shrnut v článku Srivastava a kol. (2014). Kromě metodologie jsou v něm také obsaženy příklady, kde se problematika vyskytuje. Hlouběji se tématem *feature selection* zabývá práce Hall (1999), která se vztahuje zejména na algoritmy strojového učení, kde je vysoká dimenzionalita vstupních dat častým problémem.

2.1.1 Důvody redukce

Důvodů proč redukovat dimenzionalitu matice znalostí lze najít několik (několik jich je uvedeno v Srivastava a kol. (2014)). Začněme s problémem výkonu výsledné aplikace. Jak již bylo zmíněno v předchozí kapitole, tak matice má 354 znalostí, tj. každý zaměstnanec by měl alespoň takový počet atributů. To by výrazně zvýšilo výpočetní čas na zjištění vzdálenosti mezi dvěma jedinci. Byť není problém výkonu nijak tíživý, neboť moderní počítače jsou velice rychlé a počet zaměstnanců není příliš velký, tak je redukcí vyřešen.

Daleko větší opodstatnění má redukce z hlediska závislosti sloupců. Existují znalosti, které jsou mezi sebou silně korelované, například pokud zaměstnanec ovládá Java frameworky, tak jistě umí používat i Javu a do určité míry i obráceně (aby člověk mohl programovat pro firmu v Javě, tak musí nějaké Java frameworky znát). Na druhou stranu, pokud si vezmeme znalost německého jazyka, tak ta nikterak nesouvisí s tím, jak dobře zaměstnanec ovládá Javu. V této chvíli by se mohlo zdát, že se stále jedná o první problém, tedy že o zaměstnanci uchováváme zbytečná data, která nepříznivě ovlivňují výkon. Nicméně tato závislost sloupců má i jiné implikace z hlediska správnosti metriky nad maticí znalostí. Jak je vidět v následujícím příkladu.

Mějme skupinu .NET technologií (C#, F#, ASP .NET atd.) a Java technologií (Java, JUnit, Idea atd.) a pro jednoduchost si rozdělme programátory na „dotneťáky“ a „javisty“. Je zřejmé, že dotneťák bude ovládat většinu .NET technologií a skoro žádné Java technologie (nebo jenom v omezené míře), pro javistu je situace obrácená. Problém nastává, když je .NET technologií výrazně méně, než Java technologií. V takovém případě to znamená, že javisti mají více společných atributů (znalostí) a tedy nehledě na použitou metriku si mezi sebou budou vždy podobnější než dotneťáci.

Příklad by se jistě dal vyřešit jednoduše tak, že by byly manuálně vytvořeny kategorie Java a .NET a posléze agregovány znalosti, které do konkterétní kategorie spadají, například pomocí průměru. Toto by fungovalo dobře pro kategorie, které jsou zřejmé, ale jistě je spousta závislostí, které nejsou na první pohled viditelné. Nebo by mohl vzniknout opačný problém, kdy omylem sloučíme dvě kategorie do jedné. Určitě by se mohlo stát, že dotneťaci se dělí ještě na dvě kategorie weboví inženýři a windows forms programátoři. Toto je také důvod, proč není vhodné použít již definované kategorie na matici znalostí, jak je ostatně popisováno v následující podkapitole.

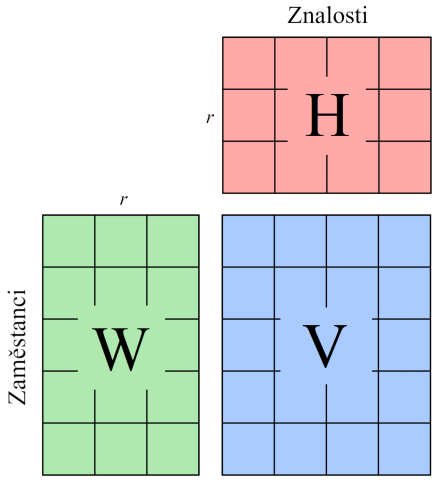
Z výše uvedeného vyplývá, že je vhodnější nechat redukovat dimenze algoritmus, jenž závislosti objeví bez naší intervence.

2.1.2 Zvolený způsob redukce

K odhalení závislostí v datech a následnému snížení dimenzionality je známých mnoho algoritmů. Zde byl zvolen nezáporný maticový rozklad (NMF), který je popsán v článku Berry a kol. (2007). Jeho ukázková aplikace při extrakci rysů obličeje je zpracována autory (Lee a Seung, 1999), kde je NMF srovnáno i s dalšími možnými algoritmy. Článek Yu a kol. (2020) potom dále rozšiřuje NMF a řeší odhalování závislostí v datech, které klasické NMF přehlíží.

NMF aproximuje původní datovou matici pomocí součinu dvou nových matic[1], jak je vidět na obrázku 2.2.

$$V \approx W * H$$



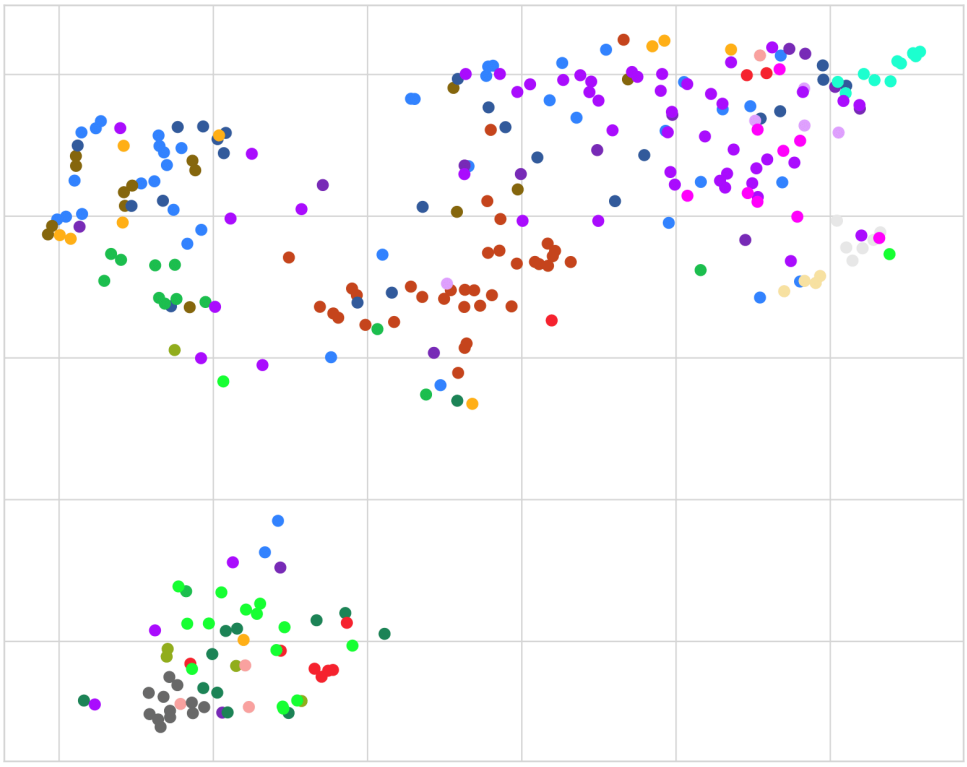
Obrázek 2.2: NMF - ilustrace rozkladu

Důvodem jeho požití je obzvláště možnost analyzovat výsledek redukce na znalostech. To je v tomto případě obzvláště důležité, protože pro zkoumání smysluplnosti redukce na zaměstnancích je potřeba osoba, která se vyzná ve většině zaměstnanců. Ta bohužel nebyla k dispozici po celou dobu vývoje aplikace.

K usnadnění analýzy byl použit UMAP (Uniform Manifold Approximation and Projection) (McInnes a kol., 2018), který promítne data podle předepsané metriky do prostoru nižší dimenze. UMAP se snaží v co největší míře zachovat topologii promítaných vektrů,tedy pokud jsou si dva vektory relativně blízko v původním prostoru, tak v projekci si budou také blízko, to samé platí i pro klastry vektorů. V tomto případě byla použita kosinová vzdálenost, důvody jsou vysvětleny v následující kapitole, a promítáno bylo na 2D rovinu.

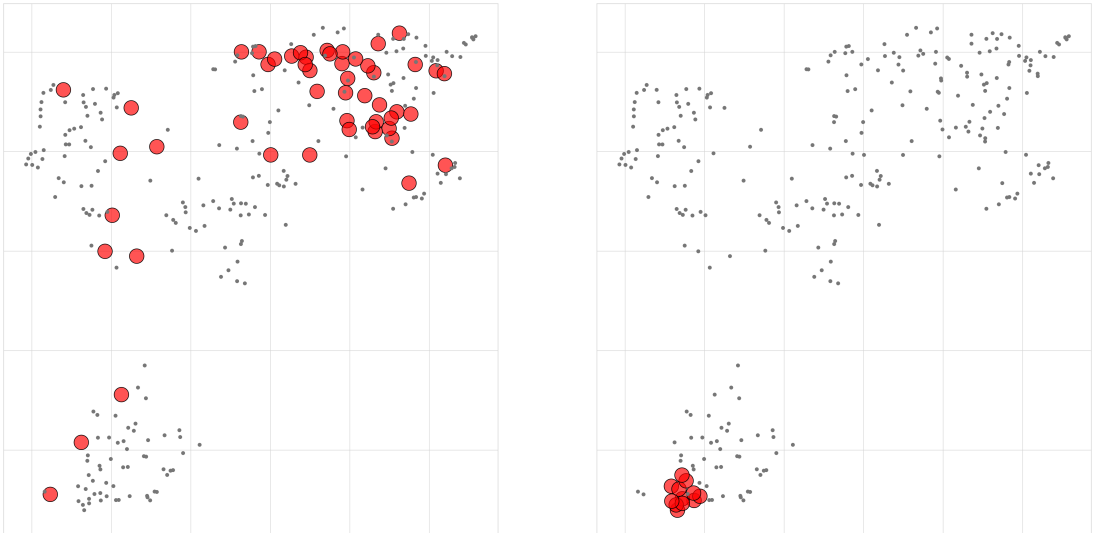
Nejprve se pojďme podívat na vlastnosti neredukovaných znalostí a také na vlastnosti jejich kategorií. Na obrázku 2.3 můžeme vidět promítnuté znalosti původní dimenze.

[1]Matice **W** je redukcí původní matice **V**. Matice **H** je redukcí matice $\mathbf{V}^{\mathrm{T}}$, čehož je při analýze využito. Obě dvě redukce mají dimenzi r.



Obrázek 2.3: Projekce neredukovaných znalostí - znalosti obarveny dle kategorií

Každý bod reprezentuje konkrétní znalost obarvenou dle její kategorie. Znalostí je zde myšlen sloupcový vektor matice znalostí, tedy vektor má stejně dimenzí, jako je zaměstnanců v matici. Na obrázku 2.4 jsou stejně promítnuté znalosti s rozdílným obarvením.



Obrázek 2.4: Projekce neredukovaných znalostí - dvě zvýrazněné kategorie

Na prvním z podobrázků jsou červeně zabarveny všechny znalosti spadající do kategorie Programovací jazyky a nástroje, na druhém jsou potom obarveny

znalosti označeny jako Data management. Rozdíl je patrný na první pohled, v prvním případě je kategorie rozmístěna po celém prostoru a v druhém se naopak dostala do jednoho klastru. První rozmístění je způsobeno tím, že každý typ programátora umí z kategorie Programovací jazyky a nástroje jinou podmnožinu, což je v případě této kategorie poměrně očekávatelné, neboť například data scientista bude umět jiné jazyky než Javista. Oproti tomu znalosti Data managementu očividně ovládá pouze jeden typ zaměstnanců, což způsobuje, že se všechny znalosti shlukly k sobě.

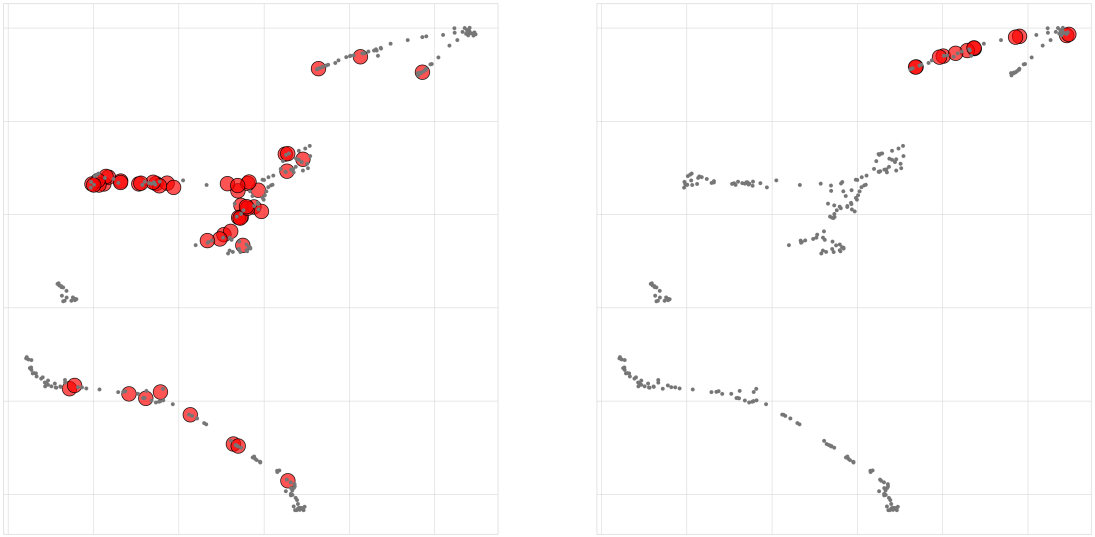
Z výše uvedeného vyplývá, že není vhodné redukovat dimenzi agregací znalostí podle zavedených 21 kategorií.

Vedlejším produktem analýzy by mohlo být předefinování stávajících kategorií, tak aby se znalosti seskupovaly dle jejich skutečné podobnosti. Nicméně k tomuto účelu by se lépe hodily algoritmy aglomerativního shlukování (Müllner, 2011; Gowda a Krishna, 1978), které nepotřebují znát předem dimenzi do které redukovat. Vzniklé shluky znalostí by definovaly nové kategorie a následná redukce by probíhala agregací znalostí dle nově vzniklých kategorií (například pomocí průměru).

Ještě poznamenejme, že v příloze A.2 je tato projekce v interaktivní podobě, tam si lze prohlédnout, kam se promítly jednotlivé znalosti.

Vzhledem k tomu, že se algoritmu NMF musí předem zvolit dimenze do jaké redukovat, tak bylo nutné si takto několik možných redukcí promítnout a zkoumat, kam se jednotlivé znalosti umístily. Cílem bylo určit co nejmenší dimenzi takovou, v níž se jednotlivé znalosti smysluplně shlukují. Po této analýze bylo rozhodnuto, že první dimenze splňující tyto požadavky je pátá dimenze.

Na obrázku 2.5 lze vidět projekci již redukovaných znalostí (tj. sloupcových vektorů matice **H** z obrázku 2.2). Opět jsou obarveny stejné kategorie znalostí jako na obrázku 2.4 (tj. Programovací jazyky a nástroje na prvním podobrázku a Data management na podobrázku druhém). Přesně jak bychom očekávali, tak se první kategorie opět dostává do několika klastrů a druhá zůstává relativně blízko u sebe v jednom klastru. Interaktivní verze je opět k dispozici v příloze A.2.



Obrázek 2.5: Projekce redukovaných znalostí

Díky této projekci víme, že nezanikla struktura dat. Pokud se tedy vrátíme k příkladu s dotnetáky a javisty, tak si můžeme být jistí, že každá ze skupin bude mít svůj klastr, kde vzdálenosti uvnitř klastrů budou podobné. V tomto příkladu budou všechny .NET technologie agregované v jedné nové dimenzi a Java technologie v další nové dimenzi.

3. Podobnostní model zaměstnance

Problematika, kterou se zde budeme zabývat, se nazývá podobnostní hledaní (similarity search). K uvedení do kontextu problému dobře poslouží kniha Zezula a kol. (2006), jenž zpracovává nejenom podobnostní metriky, ale i možnosti ze stran efektivního ukládání dat, v nichž je vyhledáváno. Ve zde vytvářeném vyhledávači ovšem není předpokládáno takové množství dat (zaměstnanců), aby se všechna nevešla do paměti a tedy bylo možné vynechat jakýkoliv typ indexace. Velkým oborem, kde se podobnostní vyhledávání používá, je webové vyhledávání. To je široce popsáno v práci Halevy a kol. (2004), kde jsou také podány důvody, proč není vhodné vyhledávání pomocí klíčových slov (keyword search).

3.1 Model zaměstnance

Nejprve zaveďme formálně model zaměstnance.

$$M = (desc_M,\, dist_M,\, w_M)$$

$$desc_M = \{\ desc_i \mid i \in I\ \}$$

$$dist_M(x,\, y) = \sum_{i \in I} w_i * dist_i(x_i,\, y_i)$$

$$w_M = custom_M \times predefined_M$$

$$I = \{know, hist, proj, school, workage, pos, seg, buddy, worktime\}$$

Model se skládá z deskriptoru, což jsou všechny informace, které jsou o zaměstnanci známé, tedy jeho znalosti (již redukované), historie projektů, současný projekt, vzdělání, služební stáří, pracovní pozice, supersegment, buddy, úvazek.

Dále se skládá z agregované metriky, která váženě sečte všechny výsledky dílčích metrik.

V neposlední řadě jsou v modelu obsaženy zmíněné váhy složené ze dvou položek (*custom* a *predefined*), kterým je věnována jedna z následujících podkapitol.

3.2 Dílčí modely

V této kapitole (resp. v jejích podkapitolách) bude představen každý z dílčích modelů, tj. bude uveden deskriptor a bude zavedena dílčí podobnostní metrika.

3.2.1 Model znalosti

$$M_{know} = (desc_{know}, dist_{know})$$

$$desc_{know} = [x_1, ..., x_5] \in \mathbb{R}^5$$

$$dist_{know}(x, y) = dist_{cos}(x, y)$$

V případě redukované matice znalostí byla použita kosinova vzdálenost. Ta přijímá na vstupu dva vektory a jejich výsledná vzdálenost je závislá na úhlu, který mezi sebou svírají, čím menší úhel tím nižší hodnota.

Abychom pochopili proč je tento přístup vhodný, musíme nejdříve pochopit, jakým způsobem matice znalostí originálně vzniká. Jak bylo avizováno, znalosti jsou vyplňovány samotnými zaměstnanci. Což znamená, že míra expertízy jedné znalosti zaměstnance je relativní vzhledem k ostatním znalostem tohoto zaměstnance a nikoliv k znalostem jiných zaměstnanců. Její absolutní hodnotu tedy nelze srovnávat s hodnotou u ostatních zaměstnanců.

Tento fenomén bude nejlépe vidět na příkladu. Mějme dva zaměstnance, jeden z nich je právě nastupujícím studentem a druhý je seniorním programátorem. Při vyplňování matice znalostí se student ohodnotí velice nízce v technologicky zaměřených znalostech a naopak si přiřkne vysoké skóre v případě teoretických znalostí, protože se v nich cítí daleko jistější (právě úspěšně složil několik zkoušek). V případě seniora bude situace o poznání jiná, ten si je, díky svým zkušenostem, jistý v technologiích a naopak v teorii bude mít čísla nižší a to i v případě, že ve skutečnosti ovládá teorii lépe než student. Oba dva tedy hodnotili své znalosti v kontextu svých vlastních znalostí a ne v kontextu znalostí ostatních.

Došli jsme k tomu, že nelze za pomoci matice znalostí srovnávat zaměstnance absolutně, ale co lze srovnat je jejich zaměření, neboli dává smysl zkoumat pouze směr vektoru znalostí. Z tohoto důvodu je třeba použít metriku, která je založena na směru vektoru a nehledí na jeho délku, což v podstatě přesně definuje, jak funguje kosinova vzdálenost.

3.2.2 Model historie projektů

$$M_{hist} = (desc_{hist}, dist_{hist})$$

$$desc_{hist} \in Strings$$

$$dist_{hist}(x, y) = Jaccard(keywords(x), keywords(y))$$

Deskriptorem je řetězec nad písmeny abecedy běžného jazyka. Hlavním problémem je, že údaj byl vyplněn jako text lidmi. Respektive je problém, že neexistuje jednotný formát názvů projektů, tj. jeden projekt může mít několik různých slovních reprezentací. Kromě toho se také v řetězci vyskytovala různá hloubka informace o projektu, například jeden zaměstnanec má v historii položku „KB“

(komerční banka), druhý „KB analýza“ a třetí „KB testing“. Tedy snadno nahlédneme, že se v řetězci krom názvu pracoviště v některých případech vyskytuje role, jakou zaměstnanec na pracovišti zastával. Tato informace nebyla z dat nijak odstraňována, neboť po konzultaci s budoucími uživateli bylo rozhodnuto, že může výslednou podobnost akorát vylepšit.

Po dobu vývoje projektu nebyl k dispozici člověk, který by znal veškeré projekty firmy. Tedy nebylo možné správně sjednotit jejich názvy a bylo nutné k problému přistoupit jinak než přímým porovnáváním řetězců.

Celý záznam o historii byl rozdělen na jednotlivá slova, která byla potom brána jako množina klíčových slov[1], ve vzorci se jedná o funkci *keywords*. Následně byla na příslušných dvou množinách použita množinová metrika, známá jako Jaccardův index.

3.2.3 Model současného projektu

$$M_{proj} = (desc_{proj}, dist_{proj})$$

$$desc_{proj} \in Strings$$

$$dist_{proj}(x, y) = Jaccard(keywords(x), keywords(y))$$

V případě současného projektu nastává problém nejednotného formátu názvů projektů, stejně jako tomu bylo v případě historie projektů. Tedy byl zvolen naprosto stejný postup - Jaccardův index.

3.2.4 Model vzdělání

$$M_{school} = (desc_{school}, dist_{school})$$

$$desc_{school} = [desc_{uni}, desc_{field}]$$

$$dist_{school}(x, y) = \frac{1}{2} * \sum_{i \in \{uni,\ field\}} Hamming(x_i, y_i))$$

Velice důležitým faktorem při volbě zaměstnance na projekt je jeho vzdělání. Bohužel stejně jako v případě deskritoru historie projektů, tak i zaměstnancovo vzdělání bylo vyplněno personalistou, tj. chyběl jednotný formát. Jeden zaměstnanec měl vyplněnou informaci, že studoval „UK MFF“, další „UK MFF OI“ apod. Jednou z možností bylo vyřešit problém jako v případě projektů (tj. keywordy v kombinaci s Jaccardovým indexem), nicméně přicházela v úvahu varianta projít všechny zaměstnance ručně a sjednotit data do čitelné podoby. Vzhledem k tomu, že firma zatím nedosahuje takové velikosti, byla zvolena možnost ruční transformace. Z dostupných údajů byly vytvořeny dva sloupce, jeden z nich obsahuje samotnou školu (UK MFF, ČVUT FIT, ČVUT FEL atd.), s tím, že méně

[1] Tato množina byla očištěna o spojky a podobné nepotřebné elementy.

obvyklé školy spadají do společné kategorie „jiná“. Druhý sloupec obsahuje jeden z devíti oborů, jako je například „softwarové inženýrství“, „matematika, statistika a umělá inteligence“, „fyzika“, „ekonomie“ a další. Obory byly designovány takovým způsoben, aby lidé mající shodné hodnoty měli tendenci pracovat na stejných, nebo podobných typech projektů. Například lidé co vystudovali matematiku, statistiku nebo umělou inteligenci často pracují na data science projektech pro banky a tedy nebylo nutné je řadit do tří disjunktních oborů.

Konečná metrika potom vrací sumu Hammingovi vzdálenosti obou dvou hodnot (normalizovanou tak, aby maximální možná vzdálenost byla rovna jedné).

3.2.5 Model služebního stáří

$$M_{workage} = (desc_{workage}, dist_{workage})$$

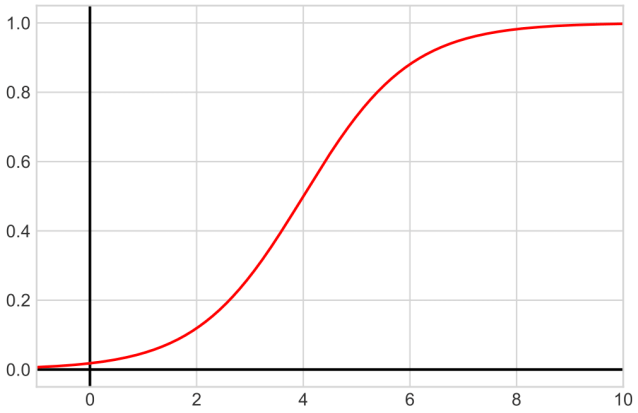
$$desc_{workage} \in \mathbb{R}^{+}$$

$$dist_{workage}(x, y) = |trans(x) - trans(y)|$$

Hodnota deskriptoru reprezentuje počet let ve firmě. Nabízela se možnost vytvořit metriku odečtečím příslušných let a prohlásit výsledek za vzdálenost. Nicméně tento přístup nemusí být tak vhodný, jak by se na první pohled mohlo zdát. Nejprve si ovšem uvědomme, co služební stáří nese za informaci o zaměstnanci. Nejde u něj tolik o množství programátorských zkušeností, jako spíše o loajalitu k firmě, tj. člověk, který je ve firmě čtvrt století, je výrazně loajálnější nežli student s rokem praxe na prvním projektu. Tato loajalita je důležitá zejména při plnění zahraničních projektů, kde jsou služebně mladší zaměstnanci náchylnější na přechod do konkurenční firmy v dané zemi.

Problém prostého rozdílu je, že pro dva lidi se služebním stářím 10 a 15 let vrací stejnou vzdálenost jako pro dva lidi, co jsou ve firmě 0 a 5 let. Je zřejmé, že v prvním případě by měl být rozdíl zanedbatelný a v druhém naopak daleko výraznější. Toto bylo vyřešeno tak, že se před samotným rozdílem služební věk vloží do logistické funkce s následujícím předpisem (graf na obrázku 3.1):

$$trans(age) = \frac{1}{1 - e^{-age+4}}$$



Obrázek 3.1: Logistická funkce transformující služební stáří

3.2.6 Model pracovní pozice

$$M_{pos} = (desc_{pos}, dist_{pos})$$

$$desc_{pos} \in \{JC, C, EC, SC, PC\}$$

$$dist_{pos}(x, y) = \mid val(x) - val(y) \mid$$

V případě pozic zaměstnance[2] nebylo třeba nic dlouze řešit. Díky tomu, že na nich existuje zřejmé lineární uspořádání (od juniora po zkušeného principal konzultanta), bylo možné jednotlivým pozicím přiřadit hodnotu z oboru reálných čísel. Tyto hodnoty se potom ve výpočtu metriky jednoduše odečtou. Pozice byly rovnoměrně rozmístěny na intervalu 0-1, ve vzorci výše tyto hodnoty přiřazuje funkce *val*.

3.2.7 Modely supersegment, buddy a úvazek

$$M_{seg} = (desc_{seg}, dist_{seg})$$
$$M_{buddy} = (desc_{buddy}, dist_{buddy})$$
$$M_{worktime} = (desc_{worktime}, dist_{worktime})$$

$$desc_{seg} \in \mathbb{N}$$
$$desc_{buddy} \in \mathbb{N}$$
$$desc_{worktime} \in \{full\ time, part\ time\}$$

[2]JC = Junior Consultant, C = Consultant, EC = Experienced Consultant, SC = Senior Consultant, PC = Principal Consultant

$$i \in \{seg, buddy, worktime\} : dist_i(x, y) = Hamming(x, y)$$

V neposlední řadě se dostáváme k údajům, u kterých v podstatě nelze použít jinou vzdálenost než Hammingovu.

3.3 Volba vah pro model zaměstnance

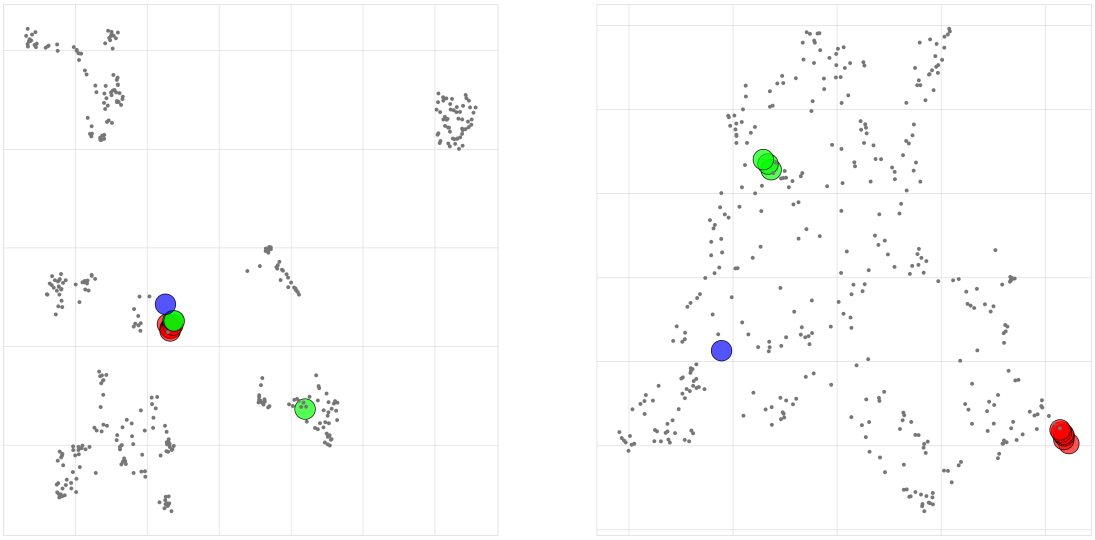
Jak bylo řečeno, tak ke sloučení všech výše zmiňovaných metrik byl použit vážený součet, z čehož plyne, že bylo potřeba vhodně zvolit hodnoty vah.

$$w_M = custom_M \times predefined_M$$

3.3.1 Predefined váhy

První složkou vah je *predefined*, ta byla experimentálně určena způsobem popsaným níže.

K validaci těchto vah (resp. finální metriky) byl opět využit UMAP, který dle dané metriky promítl zaměstnance do dvoudimenzionálního prostoru. Sám jsem schopen validovat jen velmi omezeně, znám pouze jeden tým ve firmě a tedy bylo nutné přenechat validaci na někom, kdo se v zaměstnancích orientuje. Postup vytváření a následné validace byl následující. Nejprve byly (skoro náhodně) zvoleny váhy metriky a byla vytvořena interaktivní projekce zaměstnanců pomocí UMAP na základě těchto vah. Dále jsem zkoumal, kam se promítli zaměstnanci mě známého týmu. Podle toho bylo možné určit, zda mají dané váhy potenciál figurovat ve finální metrice či nikoliv. Posledním krokem bylo předat návrhy kolegovi, který byl díky svým znalostem firmy schopen vybrat nejlepší z navrhovaných vah. Na obrázku 3.2 vidíte příklad dvou projekcí. Každá z nich je vykreslena pomocí jiné sady vah.



Obrázek 3.2: UMAP zaměstnanců dle dvou různých sad vah finální metriky

Na obou podobrázcích 3.2 body reprezentují zaměstnance a zvýrazněni jsou členové týmu, dle kterého jsem se řídil. Tým se skládá z několika studentů tj. juniorů (označeni červenou), tří zkušených programátorů (označeni zelenou) a jedné analytičky (označena modrou). Tedy bylo bezpodmínečně nutné, aby v distribuci byly viditelné tři klastry (junioři, senioři a analytička). Váhy z levého podobrázku 3.2 se do dalšího kola výběru nedostaly, protože metrika oddělila pouze jediného zaměstnance. Naopak napravo nastává ideální situace, tato distribuce byla vytvořena pomocí vah, které se nakonec dostaly do finální aplikace. Jejich hodnoty lze vidět v následující tabulce.

Váha	**Hodnota**
w_{know}	3
w_{hist}	1
w_{proj}	0.5
w_{school}	1.5
$w_{workage}$	0.9
w_{pos}	2
w_{seg}	0.5
w_{buddy}	0.5
$w_{worktime}$	0.8

Tabulka 3.1: Hodnoty predefined složky vah

3.3.2 Custom váhy

Jak bylo zmíněno v úvodu, tak je nutné dát uživateli možnost metriku upravit dle toho, co je pro něj prioritní. K tomuto účelu slouží druhá složka vah *custom*[3], kterou může uživatel v aplikaci definovat.

Ještě nutno dodat, že byť bylo nakonec rozhodnuto, že možnost úpravy metriky má v aplikaci být, tak se nepředpokládá, že by byla příliš používána. Primárním cílem bylo navrhnout základní metriku dostatečně výkonnou pro použití v případě výběru zaměstnanců na projekty.

3.4 Validace podobnosti

Důležitou součástí vytváření podobnosti je ověření, zdali metrika opravdu modeluje požadované vztahy mezi zaměstnanci.

Vzhledem k tomu, že nejsou k dispozici žádné údaje, dle kterých metriku validovat (všechny relevantní informace hrají roli v metrice), tak je situace poněkud ztížená. Ideální by bylo, kdybychom měli například definované, jak se mají zaměstnanci klastrovat a dle toho metriku hodnotit. Jak bylo řečeno, tak taková data nemáme a tedy byl zvolen jiný způsob.

Validaci bychom mohli provádět ve dvou krocích. První z nich se účastní pouze vývojáři metriky, v tomto případě se jedná o autora bakalářské práce a

[3]V aplikaci jsou některé váhy agregovány do jedné proměnné pro zjednodušení uživatelského interface. Například w_{pos}, w_{seg} a w_{buddy} jsou ovládány pomocí jednoho posuvníku *Pracovní pozice*.

vedoucího projektu ve firmě Profinit. Následující postup probíhal z největší části během určování *predefined* složky vah, kterému je věnována kapitola 3.3.1.

Nejprve je vytvořena/upravena metrika, což, kromě volby vah, můžou být i úpravy dílčích metrik (například může být změněna logistická funkce v případě služebního staří apod.). Dále jsou zaměstnanci promítnuti na dvoudimenzionální prostor pomocí UMAP. Následně je na základě rozložení jednoho konkrétního týmu Profinitu[4] rozhodnuto, zdali metrika postoupí do dalšího kola výběru. Druhé kolo potom spočívá v tom, že vedoucí práce v Profinitu, který je firmě již delší dobu, vybírá z navrhovaných metrik nejlepší z nich.

Pro přesnější validaci by mohl být proveden i druhý krok, který zahrnuje vedení firmy Profinit. Ten nebyl z časových důvodů a omezenému rozsahu bakalářské práce proveden, tedy následuje pouze diskuze nad možnými postupy.

První možností je uživatelům předložit k posouzení stejnou projekci zaměstnanců, jako byla použita v předchozí části. Výhodou je, že jsou všichni zaměstnanci zobrazeni najednou a tedy je ověření poměrně rychlé, na druhou stranu se netestuje správnost podobnostních dotazů.

Další variantou je předat uživatelům hotovou aplikaci a nechat je vyzkoušet, zdali výsledky dotazů jsou takové, jaké by očekávali. Nevýhodou je, že je tento proces časově náročný.

Ovšem je možné předchozí dva přístupy zkombinovat. Nejprve si uživatelé vyberou jednu z několika projekcí. Následně se metrika, pomocí které projekce vznikla, použije v aplikaci, kde ji uživatelé otestují. V případě, že uživatelům nebude metrika vyhovovat, se vybere jiná z nabízených projekcí, popřípadě se vytvoří nové metriky a nové projekce na základě poskytnuté zpětné vazby.

3.5 Demonstrace výsledku vyhledávání

Posuďme výhody podobnostního hledání na příkladu konkrétního zaměstnance a jeho výsledného okolí. Zaměstnanec má následující vlastnosti:

know pos	**hist seg**	**proj buddy**	**school worktime**	**workage**
[0.43, 0.42, 0, 0.36, 0] Junior Consultant	"" s1	"p1" b1	[UK MFF, null] part-time	1 year

Tabulka 3.2: Příklad dotazu - data vyhledávaného zaměstnance

V následující tabulce se nacházejí dva zajímaví zaměstnanci, kteří byli vráceni finální aplikací a nacházeli se v jednom klastru, tj. byli spojeni hranou.

[4]Jedná se o stejný tým a požadavky na distribuci jako při určování vah metriky.

know **pos**	**hist** **seg**	**proj** **buddy**	**school** **worktime**	**workage**
[0.30, 0.70, 0.02, 0.45, 0] Junior Consultant	"dpl DS,p2" s1	"dpl DS" b1	[UK MFF, AI] part-time	1 y
[0.45, 0.91, 0.05, 0.60, 0] Junior Consultant	"dpl A" s1	"dpl A" b2	[UK MFF, AI] part-time	1.5 y

Tabulka 3.3: Příklad dotazu - data dvou zaměstnanců z výsledku dotazu

Zde je potřeba vysvětlit anonymizovaná data a tím obhájil správnost výsledku. Hledaným zaměstnancem je autor bakalářské práce a výslední zaměstnanci jsou kolegové studenti pracující ve firmě, kteří také pracují na svých závěrečných pracích v podobném oboru, tedy z pohledu výběru na projekt jsou si všichni tři podobní. Ovšem SQL dotaz, který příslušné dva kolegy vrátí (a odfiltruje zbylé zaměstnance) založený na informacích o autorovi, je v podstatě neformulovatelný.

4. Vizualizační model

Posledním článkem aplikace je zobrazování výsledků. Zde bude diskutováno, jaké požadavky uživatelé na vizualizaci měli, jaké omezení to vytváří a posléze bude pro každý ze dvou druhů dotazů navržen samostatný model.

4.1 Požadavky

Vzhledem k tomu, že máme dva typy dotazů (podobní jednomu zaměstnanci a podobní více zaměstnancům), bude nutné navrhnout dvě různé vizualizace. Nicméně požadavky uživatelů jsou v obou případech totožné.

Nejdůležitější bylo vyhnout se prostému výpisu podobných zaměstnanců. Vidíme, že na zaměstnance a podobnosti mezi nimi se dá nahlížet jako na graf s váženými hranami (váha je určena metrikou). Cílem bylo zobrazit výsledek jako přehledný graf, tedy je nutno vyřešit jaké zaměstnance a hrany zobrazovat, kolik jich zobrazovat, aby neutrpěla přehlednost a samozřejmě také jakým způsoben graf vykreslit na monitor počítače.

Dalším, byť okrajovým, požadavkem bylo kromě jmen, zobrazit také fotky uživatelů. To je důležité zejména z hlediska množství vykreslených zaměstnanců, neboť jich nelze zobrazit velký počet najednou.

4.2 Návrh layoutu

Možnosti vykreslování zajímavě shrnuje článek Vaderna a kol. (2015), který diskutuje známé algoritmy a Java knihovny, jenž je implementují. Na základě tohoto článku a požadavků uživatelů byla pro účel bakalářské práce vybrána varianta forced layoutu, která je podrobněji představena v textu Dwyer a kol. (2006).

4.2.1 Graf pro jednoho zaměstnance

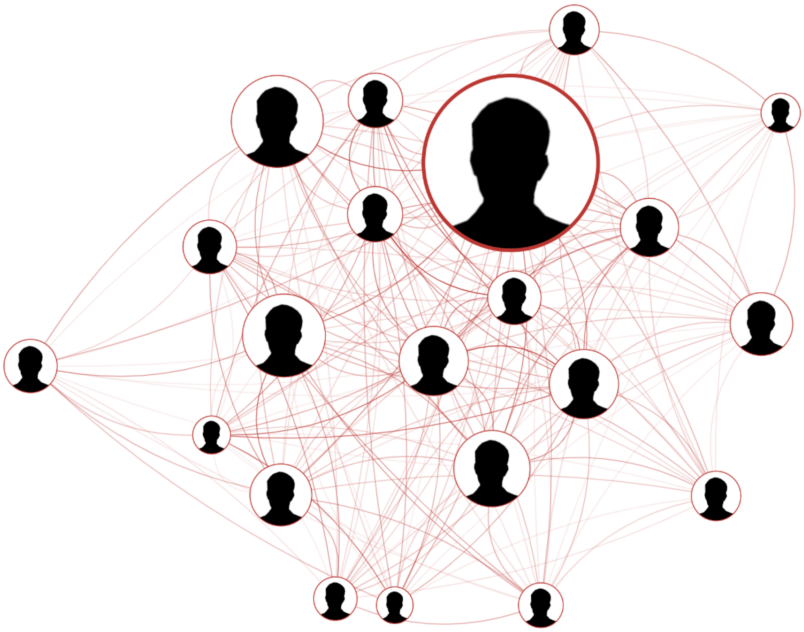
Nejprve se podívejme na to, jaké zaměstnance zahrnout do výsledku. V podstatě se zde nabízejí dvě možnosti, jednou z nich je vzít právě všechny zaměstnance z koule o fixním poloměru (středem této koule je zaměstnanec, na jehož okolí se uživatel dotazuje).

$$result(center) := \{\ e \mid e \in Employees\ \wedge\ dist(e, center) \leq threshold\ \}$$

Tento přístup má ovšem výraznou nevýhodu, může se totiž stát, že dotaz nevrátí vůbec nic (nebo příliš mnoho). Takový výsledek je uživatelsky nepříjemný, a tedy bylo od tohoto způsobu upuštěno.

Druhou variantou je jednoduše vrátit konstantní počet nebližších zaměstnanců. První výhodou je, že není třeba vymýšlet hranici (tj. poloměr zmiňované koule). Druhou výhodou je přesná kontrola nad velikostí výsledného grafu, která umožnila do aplikace přidat další parametr, pomocí kterého může uživatel určit počet zobrazených zaměstnanců.

Dalším krokem je vytvořit vážené hrany grafu, u kterých je také hlavním problémem počet. Nejprve byla zvolena hranice podobnosti[1], tedy hrany budou vytvořeny pouze pro ty dvojice, jenž jsou si dostatečně blízko. Nicméně nyní nastává podobný problém jako s vrcholy. V případě příliš mnoha hran se stává graf nepřehledný, jak lze vidět na obrázku 4.1.

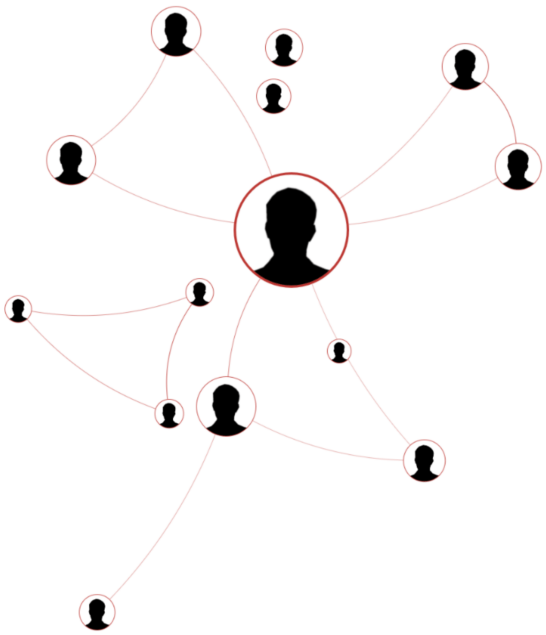


Obrázek 4.1: Vykreslený graf s příliš mnoha hranami

Aby situace nenastávala, tak byl zvolen maximální počet vykreslovaných hran, respektive maximální počet hran na jeden vrchol (nesmíme zapomínat, že počet vrcholů je proměnný).

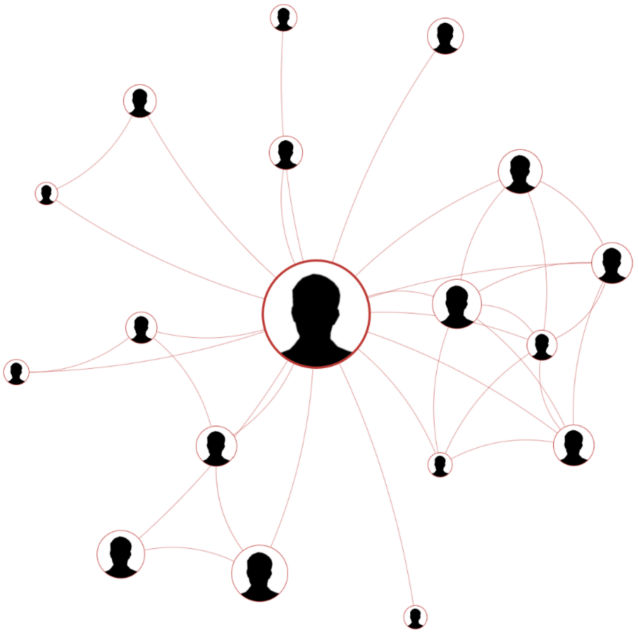
Tím byl odstraněn jeden problém, ale stále není vyřešena situace tak malého množství hran, že se graf stává nesouvislým, jak je patrné z obrázku 4.2.

[1]Hranice byla zvolena experimentálně. Každá možná hranice byla testována v aplikaci procházením grafu zaměstnanců. V případě, že byl graf řídký (tj. v okolí dotazovaného zaměstnance se netvořily klastry), tak byla zvýšena, v opačném snížena. Tento postup byl opakován, dokud se nedokonvergovalo k uspokojivým hodnotám.



Obrázek 4.2: Příklad problému mála hran ve výsledném grafu

Místo zkoumání souvislosti grafu a následného přidávání vhodných spojujících hran byl problém vyřešen jednoduchým způsobem. Do každého výsledného grafu jsou přidány všechny hrany spojující středového zaměstnance s ostatními zaměstnanci a to bez ohledu na vzdálenost mezi nimi. Středový zaměstnanec je ten, jehož okolí zobrazujeme, na obrázcích ho reprezentuje vrchol s největším poloměrem. Výsledek je vidět na obrázku 4.3, kde je zapojeno jak omezení maximálního počtu hran, tak přidání hran se středovým zaměstnancem.



Obrázek 4.3: Graf s vhodným počtem hran

Další pozitivum těchto hran je, že se středový zaměstnanec dostává do středu vykresleného grafu a výsledek začíná opravdu působit jako jeho okolí. Také je dobře viditelné, co se v tomto okolí děje, například na obrázku 4.3 je dobře patrných hned několik klastrů.

V poslední řadě byla přidána závislost velikosti vrcholu na jeho vzdálenosti od středového zaměstnance. Čím je zaměstnanec podobnější středovému zaměstnanci, tím je větší. Dále byla podobná závislost přidána v případě hran, zde kratší hrana a její sytější barva znamená, že jsou si daní dva zaměstnanci blíže. Obě závislosti jsou použité na obrázcích 4.1, 4.2 a 4.3.

4.2.2 Graf pro více zaměstnanců

Stejně jako v případě dotazu s jedním zaměstnancem je i zde na zobrazení výsledku použit forced layout, tam nicméně podobnost končí. První odlišností je, že zaměstnanci z dotazu jsou zafixovaní na kružnici a algoritmus forced layoutu je neovlivňuje.

Další velkou otázkou bylo, jaké zaměstnance by měl výsledek dotazu obsahovat, na rozdíl od předchozího případu to není tak zřejmé. Po konzultaci s budoucími uživateli bylo rozhodnuto, že je důležité pokusit se nalézt takové zaměstnance, kteří jsou podobní co největšímu počtu zaměstnanců z dotazu.

S touto informací již nebylo náročné vytvořit nad zaměstnanci lineární uspořádání. První byla zvolena hranice podobnosti[2], pomocí které je možné rozhodnout, kolika zaměstnancům z dotazu je daný zaměstnanec podobný. Tento počet je potom použit jako primární klíč k řazení. Sekundárním klíčem je součet vzdáleností od všech zaměstnanců z dotazu. Do výsledného grafu je vybráno n nejmenších zaměstnanců[3] ve smyslu tohoto uspořádání. Před tím, než se dostaneme k hranám grafu pojďme formalizovat výše uvedené uspořádaní.

$$Q_{empls} := množina\ zaměstnanců\ z\ dotazu$$

Nejprve definujme dva dvoudimenzionální vektory reprezentující dva zaměstnance (x_M a y_M). První souřadnicí je počet zaměstnanců z dotazu, kterým je daný zaměstnanec podobný. Druhou souřadnicí je suma vzdáleností od zaměstnanců z dotazu.

$$x = (\ |\{\ e \mid e \in Q_{empls} \wedge dist_M(x_M,\, e) \leq threshold\ \}|,\ \sum_{e \in Q_{empls}} dist_M(x_M,\, e)\ )$$

$$y = (\ |\{\ e \mid e \in Q_{empls} \wedge dist_M(y_M,\, e) \leq threshold\ \}|,\ \sum_{e \in Q_{empls}} dist_M(y_M,\, e)\ )$$

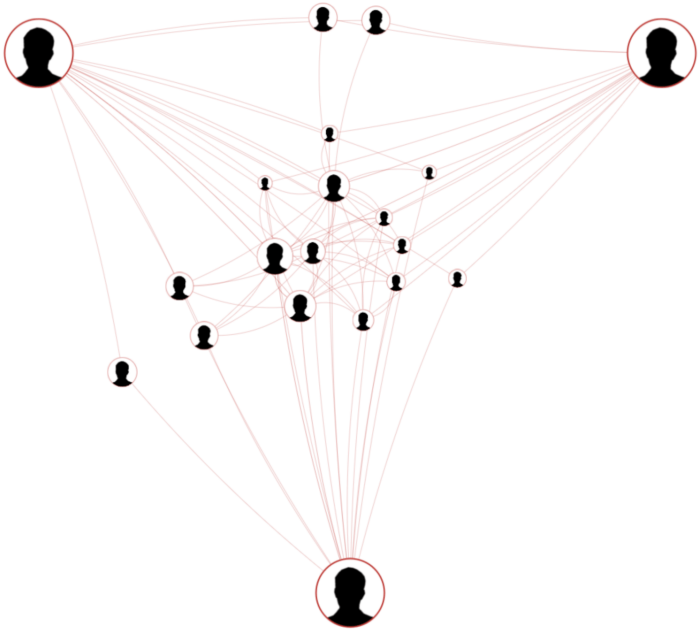
Na dalším řádku vidíme samotnou definici lineárního uspořádání.

$$x_M \leq_{empl} y_M \iff x_0 > y_0 \vee (x_0 = y_0 \wedge x_1 \leq y_1)$$

Nyní zbývá zaměstnance propojit hranami. Bylo upuštěno od všech hran, jejichž koncový vrchol není některý ze zaměstnanců z vyhledávání. Jak je vidět na obrázku 4.4, kde se nacházejí i ostatní hrany, tak je takový graf nepřehledný.

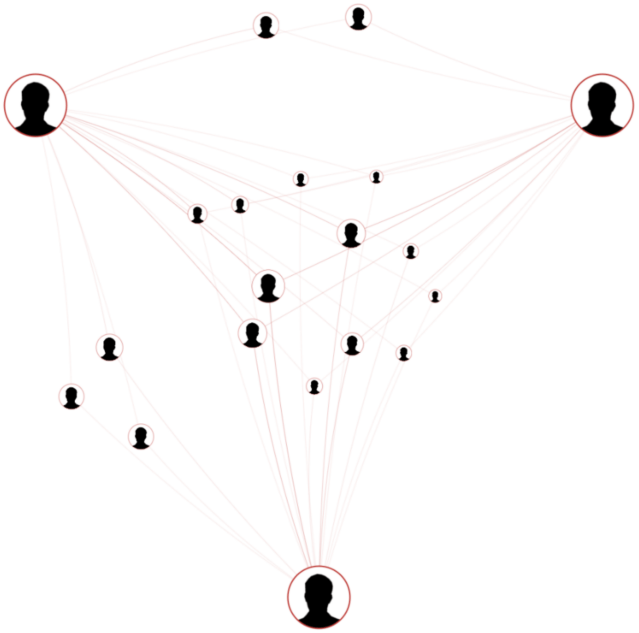
[2]Hranice byla zvolena experimentálně procházením grafu zaměstnaců stejně jako v případě dotazu s jedním zaměstnancem.

[3]Toto n je parametrem dotazu. Tedy je určené uživatelem.



Obrázek 4.4: Graf dle dotazu více zaměstnanců s interními hranami

Z grafu 4.4 se ztrácí možnost nahlédnout, v jakých poměrech jsou jednotliví zaměstnanci podobní zaměstnancům z dotazu, jelikož s touto informací interferují vztahy s dalšími zaměstnanci. Po odebrání interních hran se situace výrazně zlepšila, jak je patrné z obrázku 4.5.



Obrázek 4.5: Graf dle dotazu více zaměstnanců bez interních hran

Díky tomu, že ideální délka hrany je závislá na vzdálenosti zaměstnanců (jako předtím), tak dle umístění zaměstnance je nyní možno z grafu určit, jakým zaměstnancům z dotazu je podobný a v jakých poměrech. Tedy čím více se vrchol

blíží středu kružnice, tím více je zaměstnanec podobný všem zaměstnancům z dotazu.

Zde je velice důležitá i velikost vrcholu, neboť ta je určena součtem vzdáleností od všech zaměstnanců z dotazu, čím menší součet vzdáleností, tím větší vrchol. To je důležité zejména, pokud bude potřeba srovnávat dva zaměstnance, kteří v grafu skončili na přibližně stejném místě.

5. Aplikace

V této kapitole se dostaneme k programátorské stránce projektu. Budou definovány požadavky na aplikaci (ze strany budoucích uživatelů), bude diskutován návrh uživatelského rozhraní a v neposlední řadě bude řeč o zvolené architektuře a případně o použitých technologiích.

Zdrojové kódy aplikace s návodem na spuštění jsou dostupné v příloze A.1.

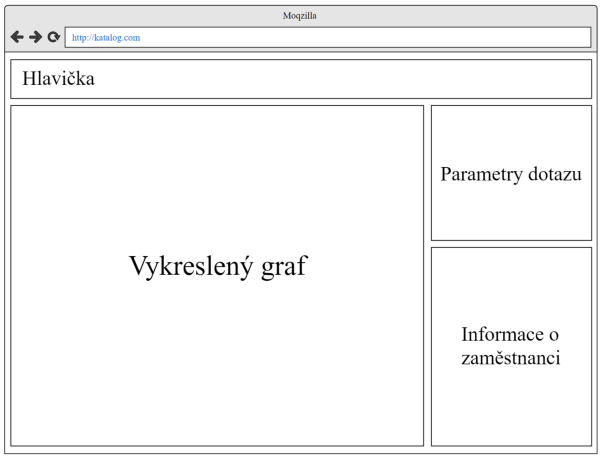
5.1 Požadavky

Nejduležitějším požadavkem bylo, aby aplikace byla webová, a to z důvodu, že desktopové aplikace vyžadují instalaci a následné updatovaní. Významně uživatelsky přívětivější je navštěvovat internetovou stránku, kde se nachází nejčerstvější verze jak aplikace, tak dat.

Další omezení bylo designové. Bylo požadováno, aby interface obsahoval co nejmenší počet obrazovek. Ideální by bylo, kdyby se celý uživatelské rozhraní vešlo na jednu stránku. Zde je argumentace podložena tím, že se novým uživatelům bude aplikace lépe používat a nebudou se ztrácet v moři nepřehledného obsahu.

5.2 Uživatelské rozhraní

Nejprve se podívejme na design uživatelského rozhraní a interakce s ním. Dle požadavků byl navržen hrubý layout elementů, který můžete vidět na obrázku 5.1.



Obrázek 5.1: Návrh layoutu aplikace

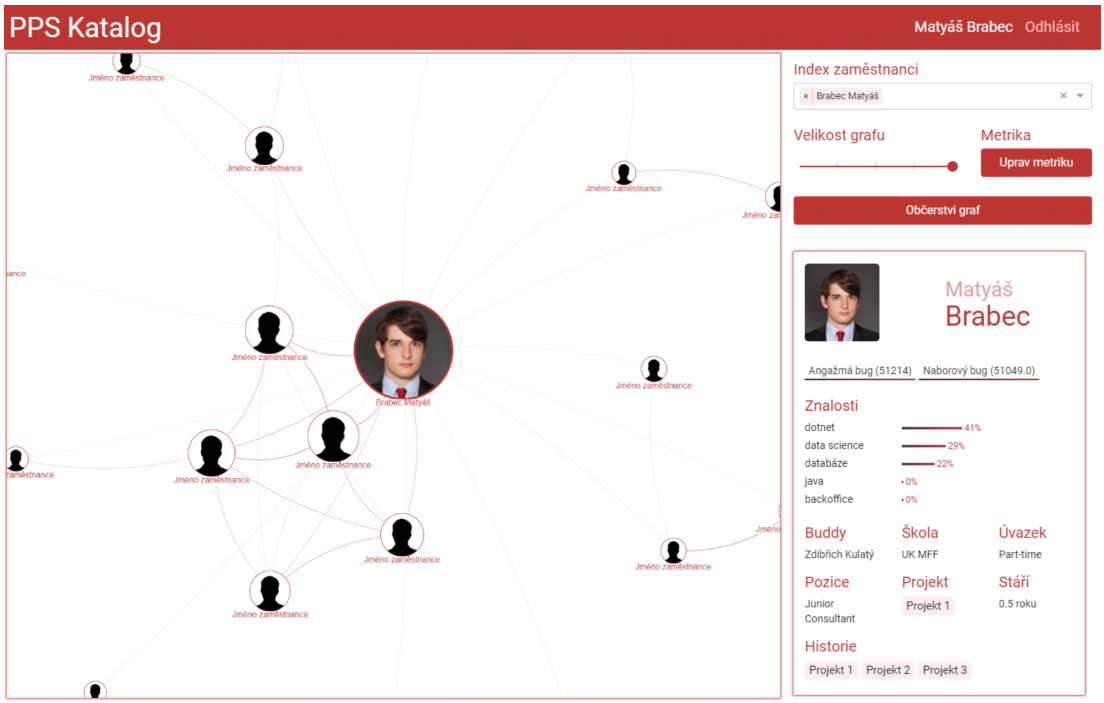
Na horní straně se nachází hlavička webové stránky, ta bude obsahovat oficiální název aplikace, přihlášeného uživatele a možnost odhlášení se z aplikace.

Hlavní a největší částí je canvas pro samotný graf, který se na obrázku rozkládá v levé dolní časti.

Další je interface umožňující parametrizovat dotaz (na obrázku „Parametry dotazu“). Tento box musel obsahovat zadávání index uživatelů, velikosti výsledku (počet vrcholů grafu) a kustomizaci podobnostní metriky.

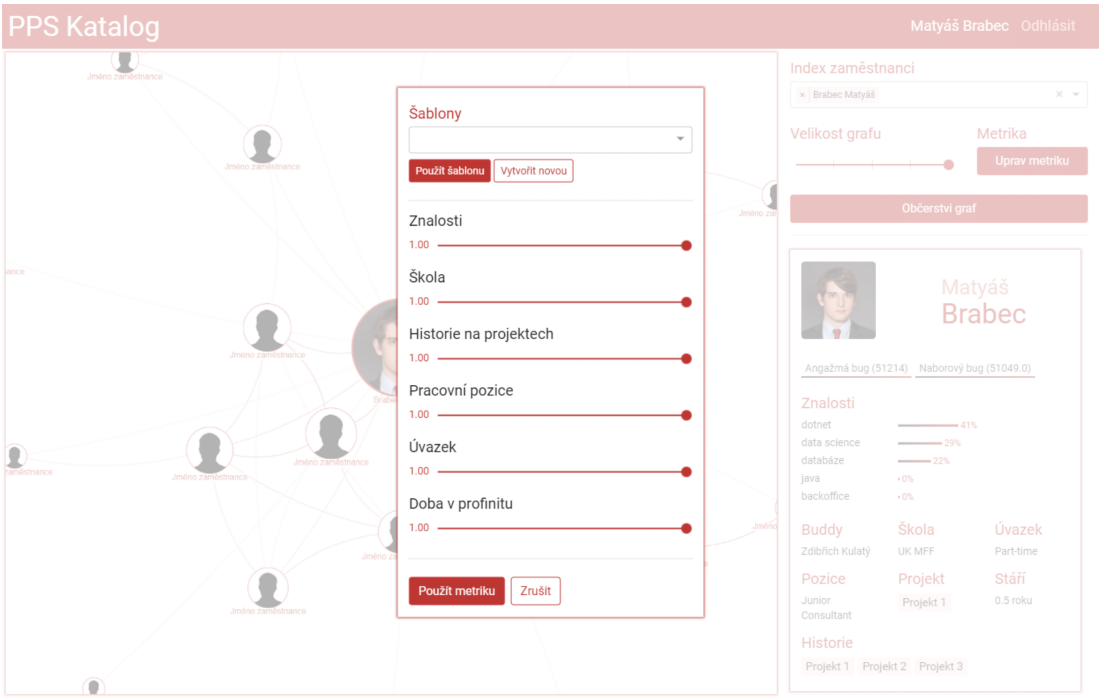
Zbývajícím elementem layoutu jsou informace o zaměstnanci, tam budou vykresleny všechny dostupné informace, jako jsou znalosti, aktuální pracovní pozice, škola atd.

Na obrázku 5.2 je již vidět finální podobu aplikace. Graf byl anonymizován, tedy místo obrázků reálných uživatelů je silueta a nahrazeny byly také citlivé informace v informačním boxu.



Obrázek 5.2: Výsledné uživatelské rozhraní aplikace

Bohužel se na tuto jednu obrazovku nevešla zmíněná parametrizace metriky. Tedy bylo nutné pro ní vytvořit samostatné vyskakovací okno, které lze vidět otevřené na obrázku 5.3.



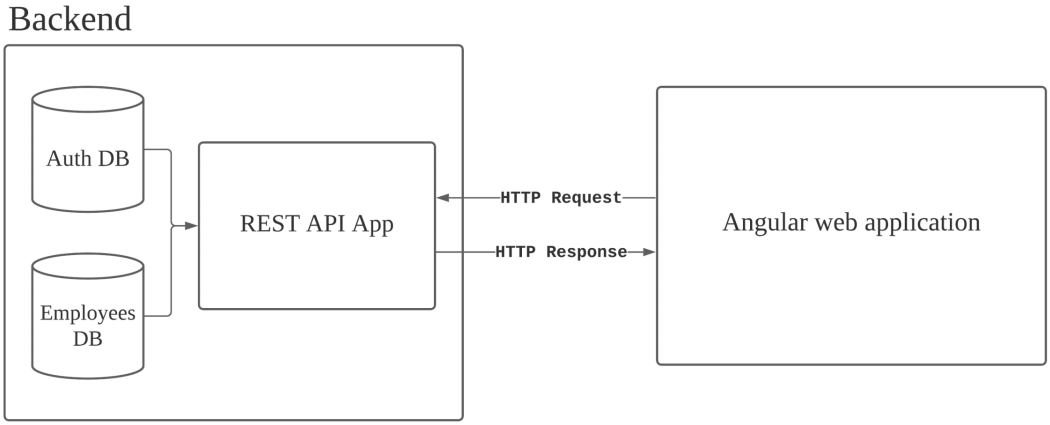
Obrázek 5.3: Okno kustomizace metriky

Okno, kromě samotného nastavování metriky, umožňuje vytvořit, upravovat a odstraňovat šablony, což jsou uložené hodnoty pro dílčí metriky.

Poslední uživatelská interakce, která zatím nebyla zmíněna, je možnost kliknout na libovolného ze zaměstnanců z grafu a zobrazit si tak jeho okolí. Vykreslí se nový graf, kde středovým zaměstnancem je kliknutá osoba.

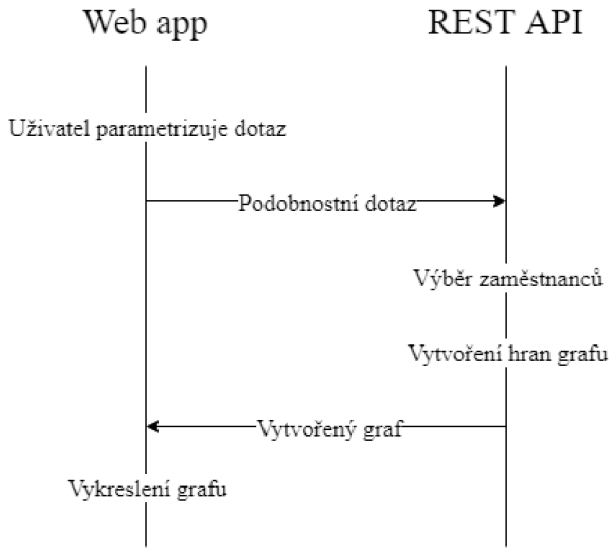
5.3 Architektura aplikace

Jak bylo avizováno v požadavcích, tak aplikace je webová, což podmiňuje její dvě části – backend a frontent. V případě backendu se jedná o REST API ve frameworku Flask (python) a frontend je SPA aplikace fremeworku Angular2+. Na diagramu 5.4 je tato architektura naznačena.



Obrázek 5.4: High level architektura aplikace

Celý proces dotazování probíhá následovně. Nejprve uživatel vytvoří pomocí výše zmíněných prostředků dotaz. Webová část aplikace ho odešle GET metodou na server (parametry jsou obsaženy v query stringu). REST API projde databázi zaměstnanců, vybere vhodné osoby a vytvoří graf (včetně vážených hran). Frontendová část potom pouze vykreslí obdržený graf. Zde popsaný proces je znázorněn na obrázku 5.5.

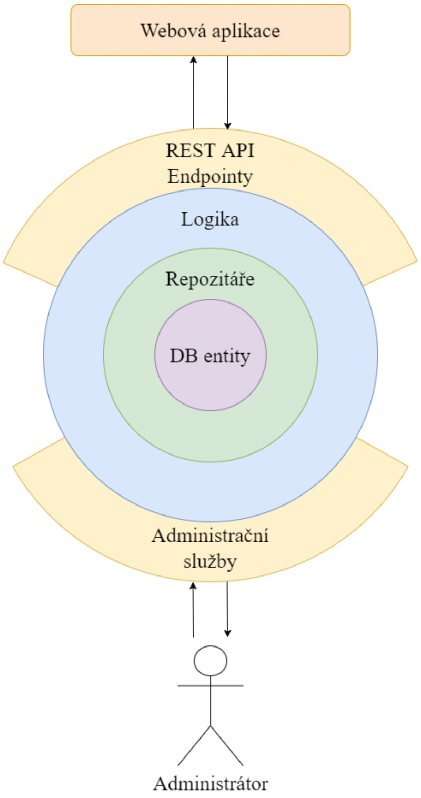


Obrázek 5.5: Proces vytváření grafu

Aplikace také obsahuje méně zajímavé dotazy, jako je získání informací o daném uživateli nebo stahování fotek zaměstnanců. Každý dotaz na server je chráněn pomocí JWT, který musí být přítomen v hlavičce dotazu, na jehož získání je implementován endpoint.

5.3.1 Backend

Myšlenka celé serverové části je vyobrazena na obrázku 5.6. Architektura je inspirována známým onion patternem.



Obrázek 5.6: Architektura serverové části aplikace

V úplném jádru se nachází databázová vrstva, ta samotná se dělí na dvě separátní databáze. Jenda z nich obsahuje zaměstnance, ve kterých vyhledáváme (při současné implementaci se jedná pouze o CSV tabulku). Druhá neméně podstatná databáze uchovává informace o uživatelích aplikace. Databáze byla rozdělena na dvě nezávislé databáze z důvodu, že v budoucnu je plánováno zcela změnit zdroj zaměstnanců - server se bude dotazovat do několika firemních systémů. Tedy je vhodnější tuto část zcela oddělit a posléze se nebude muset vůbec zasahovat do autorizační a autentizační části aplikace.

Další vrstvou jsou repozitáře, jejichž hlavním úkolem je abstrahovat tak, aby se vyšší vrstvy nemusely starat o zdroj dat a jejich přesnou strukturu.

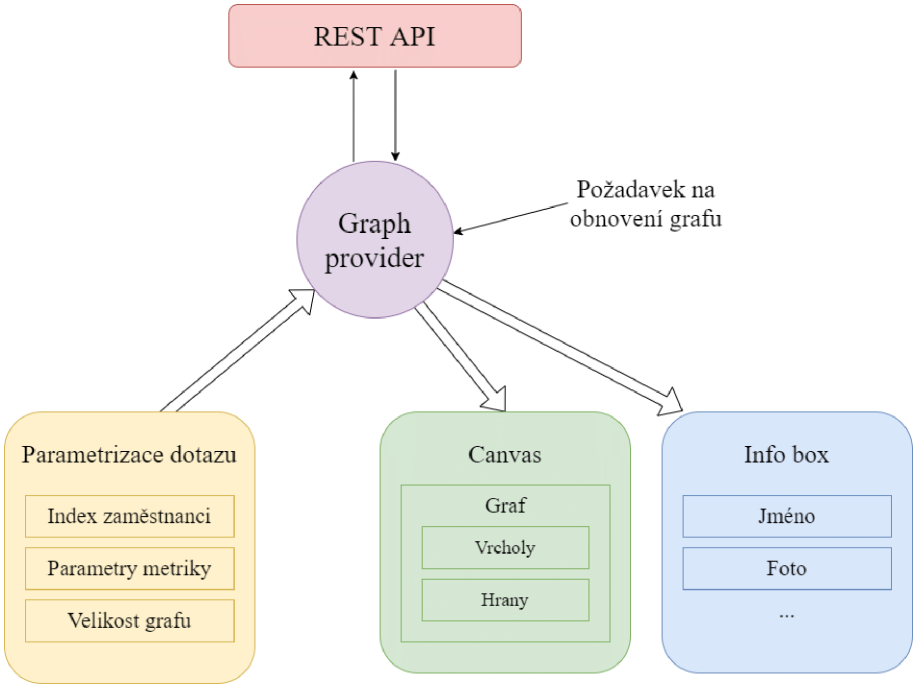
Mozek celé aplikace, se nachází ve vrstvě logika. Jedná se o samotné algoritmy na vytváření grafů, autentizaci, přidávání nových uživatelů aplikace atd.

Poslední vrstva by se také dala nazvat interakční (na obrázku 5.6 jsou to žluté části REST API Endpointy a Administrační služby). Všechny přítomné části zprostředkovávají komunikaci s vnějším světem. REST API endpoity dodávají data webové aplikaci, což je kromě samotného grafu také možnost přihlášení, získávání informací o uživatelích apod. V případě administračních služeb se jedná hlavně o správu uživatelů aplikace. Je implementován CLI, pomocí kterého lze jednoduše přidávat nové uživatele. Služby nemají webový interface z toho důvodu, že se v budoucnu integruje autentizace a autorizace s centrálním firemním systémem a tedy je opět vhodnější oddělit tuto část od samotné aplikace.

5.3.2 Frontend

Zvolený framework ve frontendové části aplikace (Angular2+) vynucuje modulární strukturu. Stránka samotná je poskládána z jednotlivých komponent, které jsou sami tvořeny jednotlivými podkomponenty a takto rekurzivně dále. Nejvyšší komponenty v podstatě odpovídají obrázku 5.1 (tj. canvas grafu, parametrizační komponenty, informační box), další dělení komponent zde není nutné komentovat.

Podstatným rysem frameworku jsou servisy, které tvoří komunikační kanály mezi jednotlivými komponenty, interakce v rámci aplikace jsou naznačeny na obrázku 5.7.



Obrázek 5.7: Diagram interakcí na frontendu

Diagram ilustruje tok dat v případě, že nějaký event požádá servis Graph provider o obnovení grafu. Zmíněný event může být například změna zaměstnanců dotazu, klinutí na nějaký vrchol grafu nebo i první načtení aplikace.

Jak je vidět, tak servis přečte aktuální data z komponent týkajících se parametrizace dotazu. Pomocí těchto informací formuluje dotaz na server, který vrátí graf zaměstnanců. Posléze servis naplní obdrženými daty komponenty canvas a info box.

6. Use Cases

V této kapitole bude představena dvojice příkladů, jeden bude využívat podobnostní dotaz založen na jednom zaměstnanci a druhý bude založen na více zaměstnancích.

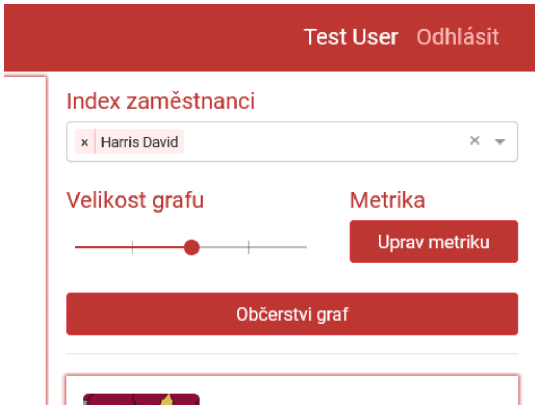
Data reálných zaměstnanců byla anonymizována, aby nebyly odhaleny citlivé informace jak samotných zaměstnanců tak firmy Profinit. Následující příklady pracují s již takto upravenými osobami.

Elektronicky podepsané povolení k publikaci od firmy Profinit se nachází v příloze A.3.

6.1 Dotaz na okolí jednoho zaměstnance

V prvním usecasu řeší personalista následující problém. Z týmu odešel klíčový zaměstnanec a tedy je potřeba za něj najít náhradu.

Po otevření aplikace personalista zadá jméno zaměstnance do searchboxu s labelem „Index zaměstnanci“[1], v případě na obrázku 6.1 je jeho jméno David Harris. Pro potvrzení výběru není třeba klikat na tlačítko „Občerstvi graf“[2], graf je obnoven automaticky při jakékoliv změně parametrů dotazu.



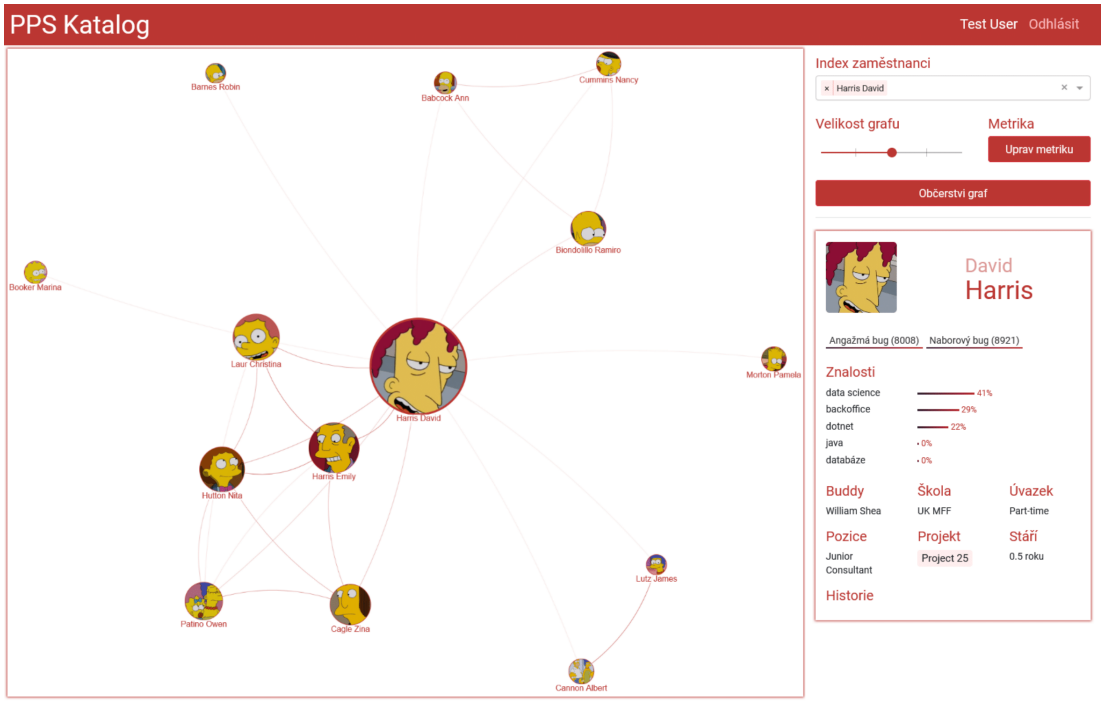
Obrázek 6.1: Zadání jména do searchboxu aplikace - jeden zaměstnanec

Aplikace nyní najde a vykreslí okolí tohoto zaměstnance. Také se zobrazí jeho informace spolu s odkazy do interních systémů[3] . Oboje je vidět na obrázku 6.2.

[1] searchbox při vyhledávání našeptává možná jména

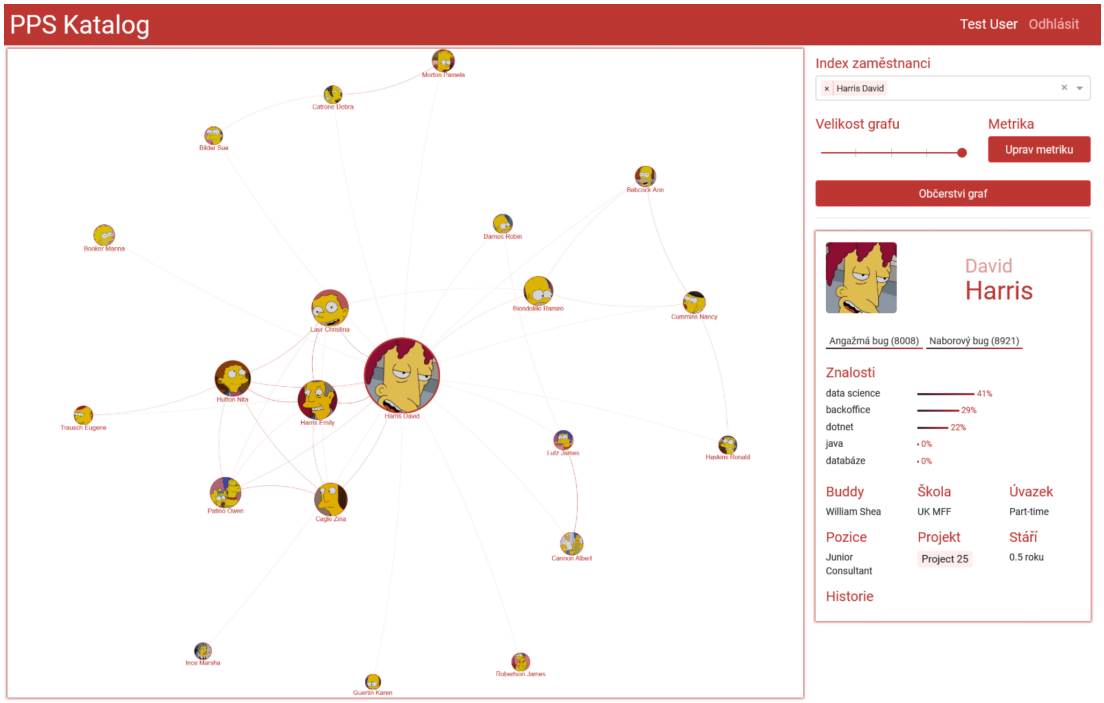
[2] tlačítko slouží primárně k navrácení do původního stavu v případě, že personalista prochází graf příliš hluboko

[3] V aplikaci se jedná o odkazy *Angažmá bug* a *Naborový bug*. *Náborový bug* je záznam o zaměstnanci, který vzniká při jeho nástupu a nachází se v něm například školy, které zaměstnanec vystudoval. V *Angažmá bugu* se zaznamenávají aktivity zaměstnance, jako například přestupy na projekty, aktuální výše úvazku atd.



Obrázek 6.2: Výsledek hledání dle zaměstnance jménem David Harris

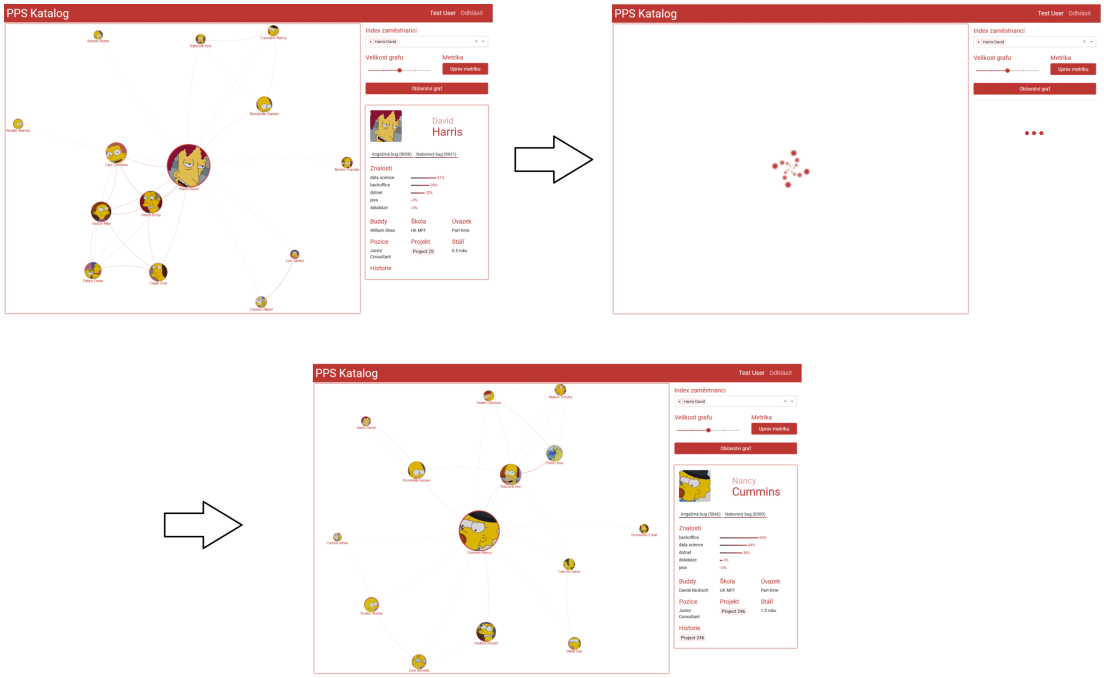
Je možno změnit velikost grafu pomocí posuvníku s labelem „Velikost grafu“ (opět není třeba potvrzovat velikost explicitně tlačítkem). Graf bude překreslen s vyšším počtem vrcholů, což umožní personalistovi nahlížet na širší okolí zaměstnance. Výsledek lze vidět na obrázku 6.3.



Obrázek 6.3: Vykreslený graf s maximálním počtem vrcholů

Personalista může procházet graf klikáním na jednotlivé vrcholy. Kliknutí způsobí změnu středového zaměstnance a aplikace vykreslí jeho okolí a zobrazí jeho informace. Tato akce je zcela ekvivalentní ručnímu zadání jména kliknutého zaměstnance do searchboxu.

Situace je naznačena na obrázku 6.4, kde personalista kliknul na zaměstnance jménem Nancy Cummins.



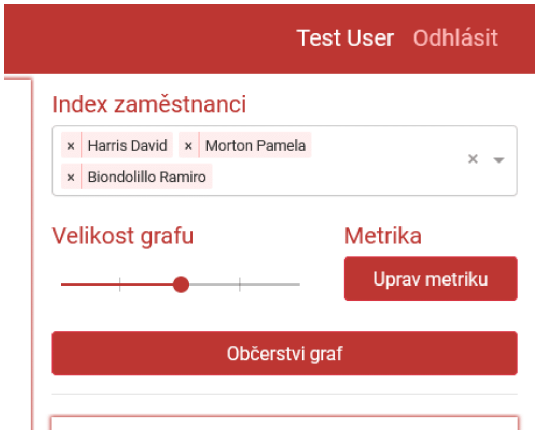
Obrázek 6.4: Průběh kliknutí na vrchol grafu - jeden zaměstnanec

Personalista tímto prohledáváním grafu nachází možné kandidáty za odchozího zaměstnance.

6.2 Dotaz na okolí více zaměstnanců

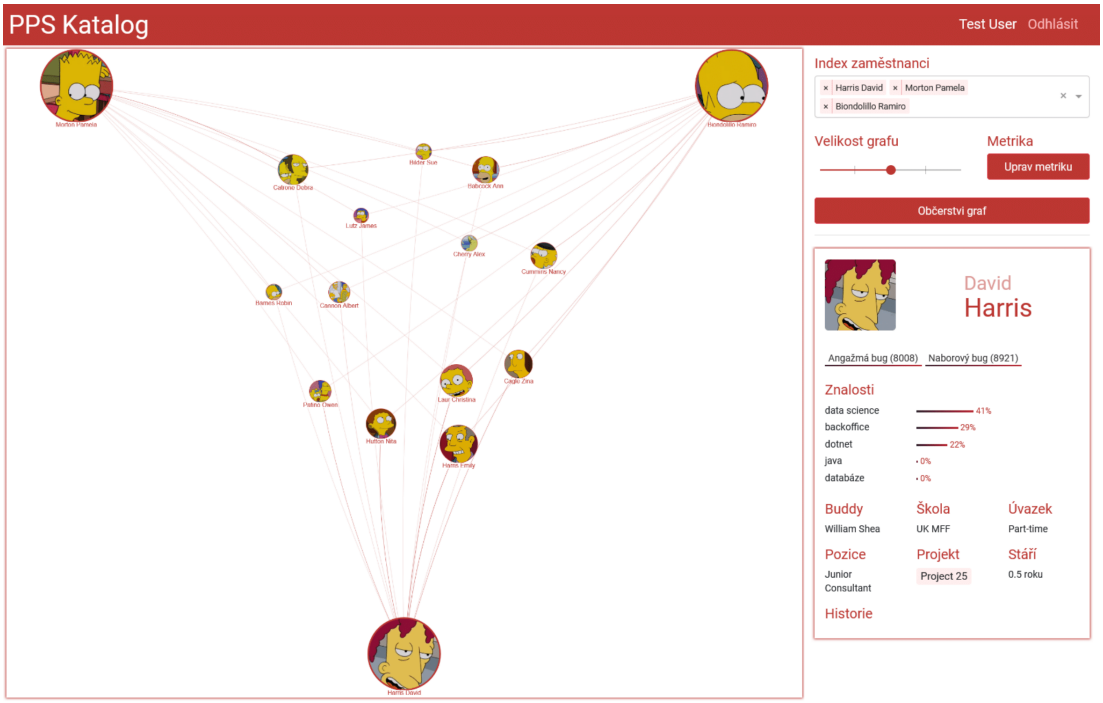
V tomto případě je třeba obsadit místo na projektu s tím, že personalista má tři kandidáty, kteří by byli vhodní. Místo toho, aby personalista procházel okolí každého z nich samostatně, tak se aplikace lze dotázat na okolí všech tří najednou.

Nejprve personalista zadá do searchboxu jména všech tří zaměstnanců. Na obrázku 6.5 můžeme vidět, že v příkladu se jedná o zaměstnance David Harris, Pamela Morton a Ramiro Biondolillo.



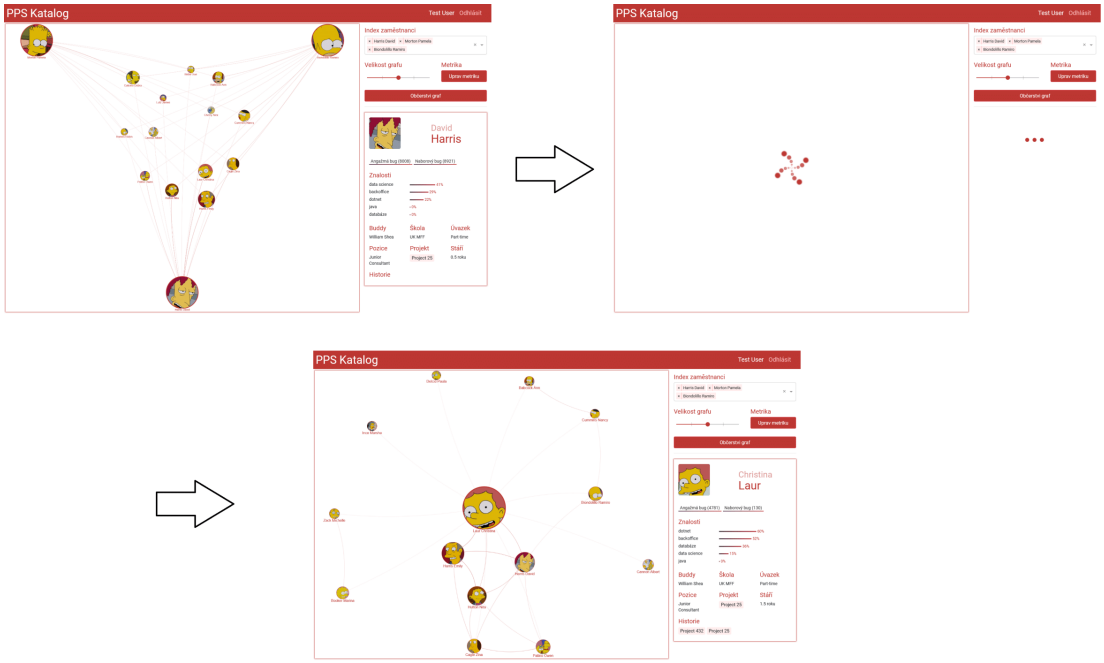
Obrázek 6.5: Zadání jmén do searchboxu aplikace - více zaměstnanců

Aplikace nyní najde a vykreslí společné okolí všech tří zaměstnanců. Zobrazené informace se vztahují k prvnímu ze zaměstnanců v searchboxu. Výsledek je vidět na obrázku 6.6.



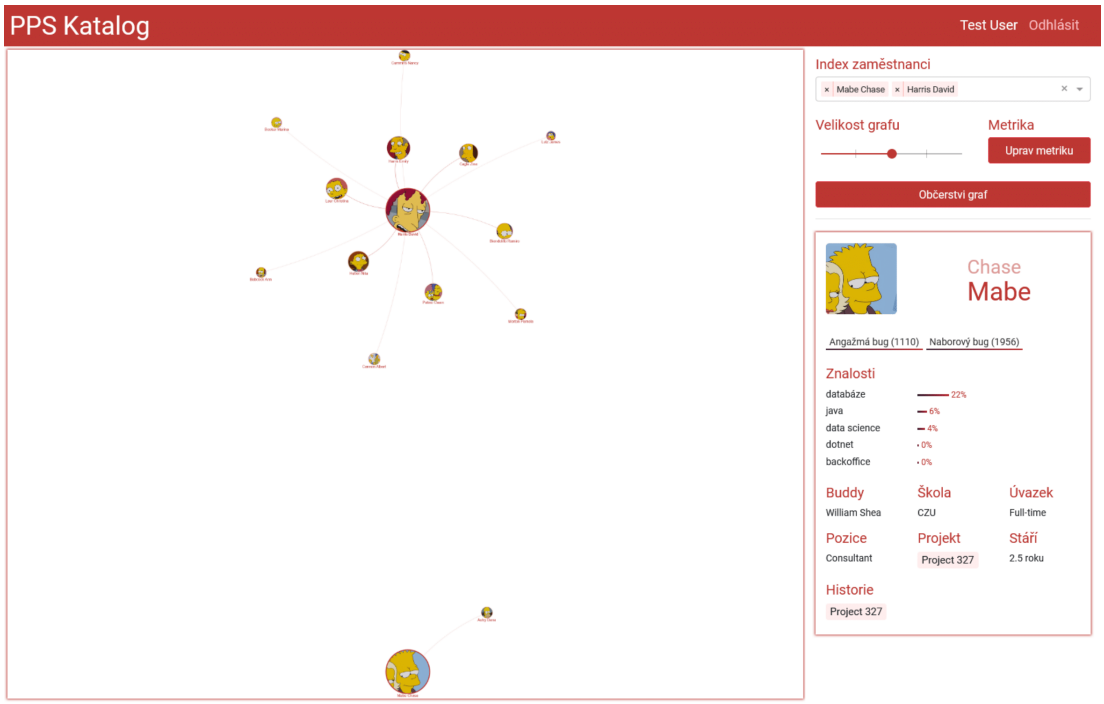
Obrázek 6.6: Výsledek hledání dle tří zaměstnanců

Personalista stejně jako v prvním případě může klikat na libovolné vrcholy grafu. Po kliknutí se zobrazí okolí jednoho zaměstnance, nevzniká tedy nový dotaz s více zaměstnanci. Na obrázku 6.7 je vidět průběh kliknutí na zaměstnance Christina Laur.



Obrázek 6.7: Průběh kliknutí na vrchol grafu - více zaměstnanců

Personalista pomocí tohoto dotazu nahlédne, jestli se ve firmě nachází zaměstnanci podobní všem hledaným či nikoliv. V případě, že neexistují zaměstnanci podobní všem, tak nastává situace z obrázku 6.8. Na něm jsou k nalezení zaměstnanci spojeni hranou pouze s jedním z index zaměstnanců.



Obrázek 6.8: Výsledek hledání dle dvou zaměstnanců - žádné spoolečné okolí

I tento výsledek je validním a chtěným výstupem aplikace. Říka personalistovi,

že firma nemá k dispozici osobu podobnou oběma zaměstnancům.

Závěr

V práci byl popsán proces vzniku aplikace pro podobnostní vyhledávání mezi zaměstnanci softwarové firmy Profinit. Byla představena dostupná data o zaměstnancích a následně byl popsán způsob jejich zpracování. Posléze byla namodelována a částečně validována podobnostní metrika, sloužící k porovnávání lidí na základě jejich vhodnosti nastoupit na stejný projekt. Dále byl diskutován vhodný layout výsledku podobnostního hledání, tj. graf kde vrcholy jsou zaměstnanci a hrany reprezentují podobnost mezi nimi. Pro oba dva typy dotazů (dle podobnosti jednomu zaměstnanci a dle podobnosti víceru zaměstnancům) byl navržen způsob výběru správných vrcholů grafu a samozřejmě výběru vhodných hran, to vše s ohledem na přehlednost výsledku. V neposlední řadě byla navržena a implementována samotná aplikace prohlížeče. Byla popsána její architektura jak na serverové tak na klientské části.

Aplikace má mnoho směrů, kam se nyní může rozvíjet. Jedním z nich je implementovat editor superzaměstnance a následné vyhledávání dle této umělé entity. Nezřídka kdy se může stát, že se objeví pozice na projektu, ale personalista nezná žádného vhodného adepta. V takovém případě samozřejmě nelze použít aplikaci v současném stavu, protože ta je schopna vyhledat podobné osoby pouze na základě existujících zaměstnanců. Buď se nabízí možnost dovolit uživateli definovat fiktivního člověka (tj. superzaměstnance) nebo navrhnout několik archetypů programátorů (např. zkušený Javista, junior data scientist atd.). Oba přístupy mají své výhody a nevýhody, například první nabízí více flexibility, ale na druhou stranu tím nutíme uživatele znát datový model zaměstnance, což je něco, čemu jsme se chtěli touto aplikací vyhnout. Tedy pravděpodobněji by se uvažovalo o druhé možnosti, která se více shoduje s filosofií projektu.

Další možností jak aplikaci rozvinout je přidat nová data o zaměstnancích. Například velkým kandidátem na další deskriptor je životopis zaměstnance. Takové rozšíření bude vyžadovat zpracování psaných textů a vytvoření z nich sérii atributů. Data by se ovšem také dala rozšířit jiným směrem a to přidáním jiných než interních programátorů. Tady by byla varianta získat volně dostupná data o externích programátorech a pokusit se vytvořit metriku, která bude schopna porovnat interního zaměstnance a externí osobu nepracující v Profinitu.

Určitě by se dal také rozvinout layout výsledku. Existuje mnoho dalších algoritmů na vykreslování grafů. Dále by bylo možné zcela opustit koncept klasického grafu a výsledky zobrazovat jako hierarchický seznam apod.

V neposlední řadě bude možno rozšiřovat samotnou aplikaci. Jak již bylo v některé z předešlých kapitol zmíněno, tak je plánováno změnit zdroj dat současné implementace, která čerpá pouze ze statického souboru. Bylo by možné aplikaci integrovat ze všemi interními systémy tak, aby data byla dynamická a aktuální. Preferovaným způsobem bude jednou za určitý čas spustit proces, který obejde všechny zdroje a načte z nich aktuální data. Nebylo by tedy nutné příliš zasahovat do zdrojových kódů, spíše by se přidala nová část. Další možností je integrovat správu uživatelů aplikace s centrálním přihlašovacím systémem. To by bylo velice vítané zejména ze strany personalistů, kteří by si tak nemuseli pamatovat další heslo do dalšího systému.

Seznam použité literatury

Berry, M. W., Browne, M., Langville, A. N., Pauca, V. P. a Plemmons, R. J. (2007). Algorithms and applications for approximate nonnegative matrix factorization. *Computational statistics & data analysis*, **52**(1), 155–173.

Dwyer, T., Koren, Y. a Marriott, K. (2006). Ipsep-cola: An incremental procedure for separation constraint layout of graphs. *IEEE Transactions on Visualization and Computer Graphics*, **12**(5), 821–828.

Gowda, K. C. a Krishna, G. (1978). Agglomerative clustering using the concept of mutual nearest neighbourhood. *Pattern recognition*, **10**(2), 105–112.

Halevy, A., Nemes, E., Dong, X., Madhavan, J. a Zhang, J. (2004). Similarity search for web services. In *Proceedings of the 30th VLDB Conference*, pages 372–383.

Hall, M. A. (1999). Correlation-based feature selection for machine learning.

Lee, D. D. a Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, **401**(6755), 788–791.

McInnes, L., Healy, J. a Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

Müllner, D. (2011). Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378*.

Srivastava, M. S., Joshi, M. N. a Gaur, M. (2014). A review paper on feature selection methodologies and their applications. *IJCSNS*, **14**(5), 78.

Vaderna, R., Milosavljević, G. a Dejanović, I. (2015). Graph layout algorithms and libraries: Overview and improvements. In *ICIST 2015 5th International Conference on Information Society and Technology Proceedings*.

Yu, N., Wu, M.-J., Liu, J.-X., Zheng, C.-H. a Xu, Y. (2020). Correntropy-based hypergraph regularized nmf for clustering and feature selection on multi-cancer integrated data. *IEEE Transactions on Cybernetics*.

Zezula, P., Amato, G., Dohnal, V. a Batko, M. (2006). *Similarity search: the metric space approach*, volume 32. Springer Science & Business Media.

Seznam obrázků

Seznam tabulek

A. Přílohy

A.1 Zdrojové kódy aplikace

A.2 Interaktivní projekce znalostí

A.3 Povolení publikace bakalářské práce